Formal Security and Privacy in Cryptoeconomic Systems

Chiang, James Hsin-yu

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Chiang, J. H. (2023). *Formal Security and Privacy in Cryptoeconomic Systems*. Technical University of Denmark.

Formal Security and Privacy in Cryptoeconomic Systems

James Hsin-yu Chiang
姜欣宇

Ph.D. Thesis

May 2023

Document compiled on June 1, 2023.

Supervisor:	Alberto Lluch-Lafuente	DTU Compute
Co-supervisor:	Bernardo David	IT University of Copenhagen
Co-supervisor:	Massimo Bartoletti	University of Cagliari
Hosting supervisor:	Ittay Eyal	Technion - Israel Institute of Technology

Technical University of Denmark
Department of Applied Mathematics and Computer Science

Abstract

Cryptoeconomics is an emerging science concerned with the formal investigation of permissionless consensus and the innovative ecosystem of financial applications hosted by such distributed computing platforms. As a field it lies at the intersection of the study of consensus protocols, economic incentive design and advanced cryptography for security and privacy.

Permissionless consensus, first introduced by Bitcoin, realizes a novel notion of serverless[1] computing, since any ephemeral, unauthenticated participant can be replaced anytime, even after sending just a single message. Such protocols achieve global agreement on a history of inputs submitted by clients; prior to the advent of the Bitcoin protocol, unauthenticated agreement was considered impossible.

The permissionless, "anybody can join" nature of such platforms has given rise to an ecosystem of financial applications, called Decentralized Finance (DeFi). In contrast to traditional finance, which is highly asynchronous and loosely coordinated across many institutions and parties, protocols in DeFi are highly composable since interactions with multiple DeFi venues are settled immediately with a single interaction, enabling financial innovations such as risk-free strategies or collateral-free flash loans that are impossible in the traditional setting. The design of financial protocols in DeFi differ significantly from their counterparts in traditional finance, since they require effective incentivization mechanisms for rational, yet unknown participants; furthermore, their implementations are constrained by limited scalability of permissionless consensus, requiring novel approaches to overcome this. However, DeFi suffers from a lack of formalization impeding formal analysis of its security. Furthermore, a general lack of privacy in DeFi exposes the system to economic, front-running attacks which reduce impose a tax on the overall utility of the system.

In this PhD thesis, we investigate the formal security at the intersection of permissionless consensus and financial applications deployed on top; we formalize the semantics and desired economic properties of lending protocols and automatic market makers in decentralized finance and observe that a general lack of privacy enables rational attacks by economic actors, called front-running or Miner-Extractable-Value. In response, we investigate novel, privacy-enhancing techniques for lower-level, permissionless consensus as well as higher-level application protocols to mitigate privacy leakage that enables such economic attacks. We introduce novel realizations of long-running privacy for DeFi applications enabled by secure multi-party computation and extended notions of differential privacy; we believe our contributions open up a novel, practical design space for cryptoeconomic applications which benefit from fine-grained control of user privacy.

This thesis also provides a gentle introduction to input fairness in consensus, security in decentralized finance and privacy in cryptoeconomic systems; we hope this may serve the interested reader looking to explore interdisciplinary research domains in cryptoeconomics.

[1]The term is overloaded. Server-less cloud computing is not permissionless (§3.3.1).

Resumé

Kryptoøkonomi er et område i vækst, der beskæftiger sig med den formelle undersøgelse af tilladelsesfri konsensus og det innovative økosystem af finansielle applikationer der er skabt ovenpå disse distribuerede computerplatforme. Som felt, kombinerer det studiet af konsensusprotokoller med økonomisk incitamentdesign og avanceret kryptografi for sikkerhed og privatliv.

Tilladelsesfri konsensus, som først blev introduceret af Bitcoin, realiserer en ny forestilling om *serverless computing*[1], hvor enhver ny-kommen, uautentificeret klient kan erstattes når som helst i protokollens forløb, selv efter blot at have sendt en enkelt besked. Sådanne protokoller opnår global enighed om en historie af beskeder sendt af klienter. Inden offentliggørelsen af Bitcoin-protokollen blev global enighed, mellem sådanne uautoriserede klienter, anset for at være umuligt.

Den tilladelsesfrie, "alle kan være med-karakter af sådanne platforme har været grobund for et økosystem af finansielle applikationer, kaldet Decentralized Finance (DeFi). I modsætning til traditionel finans, som er asynkron og sparsomt koordineret på tværs af institutioner og parter, er protokoller i DeFi i stand til at samarbejde. Interaktioner mellem DeFi applikationer muliggør finansiel innovation såsom risikofrie strategier eller såkaldte flash loans uden sikkerhedsstillelse, der ellers er umulige i de traditionelle finansielle systemer. Design af finansielle protokoller i DeFi er markant anderledes sammenlignet med deres modparter i traditionel finans. Blandt andet på grund af de indbyggede incitament-mekanismer for rationelle klienter, der er nødvendige, samt deres begrænsede skalerbarhed. Begge er problemer som kræver innovative, om end sikre løsninger indenfor protokoldesign. DeFi økosystemet lider dog af en mangel på formalisering som gør det næsten umuligt at analysere sikkerheden af applikationer. Dertil kommer en generel mangel på privatliv i DeFi systemet som muliggør strategier fra højfrekvenshandel som fx front running og andre "angreb" som udnytter systemet svagheder for egen vindings skyld.

I denne ph.d.-afhandling undersøger vi den formelle sikkerhed i området mellem tilladelsesfri konsensus og de dertilhørende finansielle applikationer; vi formaliserer semantikken og de ønskede økonomiske egenskaber ved låneprotokoller og automatiske markedsskabere i DeFi. Vi observerer at en generel mangel på privatliv muliggør rationelle angreb fra økonomiske aktører såsom front running og Miner-Extractable-Value. For at afhjælpe dette, undersøger vi nye teknikker for at fremme privatliv i tilladelsesfri konsensus samt fjerne den lækage af private informationer på applikationsniveau, der ellers ville muliggøre økonomiske angreb. Vi introducerer nye protokoller med private løsninger for DeFi-applikationer, der er bygget op omkring sikker beregning mellem flere klienter - såkaldt MPC (multiparty computation) - og nye tiltag inden for differential privacy. Dette åbner op for nye muligheder for design af kryptoøkonomiske applikationer, som gør det muligt at finjustere kontrollen med brugernes privatliv.

Denne afhandling giver også en skånsom introduktion til input fairness i konsensus, sikkerhed i DeFi og privatliv i kryptoøkonomiske systemer. Vi håber at dette kan hjælpe den interesserede læser der ønsker at udforske den tværfaglige forskning inden for kryptoøkonomi.

[1]Udtrykket refererede oprindeligt til cloud computing, som ikke nødvendigvis er tilladelsesfri (§3.3.1).

Acknowledgements

This PhD was made possible by a 3-year scholarship awarded by the PhD school of DTU Compute; their support has enabled me to freely investigate an emerging field in computer science. Contributions presented in this thesis were joint work with 12 external researchers[1] across 11 research institutes[2] in Belgium, Denmark, Finland, Israel, Italy and the USA.

A fellowship awarded by the Otto Mønsted foundation and the Innovation Centre Denmark in Tel Aviv supported a research visit to Ittay Eyal at Technion, Israel for 6 months, expanding my research horizons to permissionless consensus design and resulting in an important contribution of this thesis. Ittay has co-authored one of the most cited papers in permissionless consensus and provided generous mentorship during an unforgettable stay in Israel.

I had the fortune to be advised by Alberto Lluch-Lafuente. Without exception, Alberto has encouraged me to pursue research questions most meaningful to me and to initiate external collaboration opportunities to explore new research avenues. Most, if not all, projects in this thesis were made possible with his support and guidance. This includes several fruitful collaborations with co-supervisor Massimo Bartoletti, who has contributed some of the earliest formal languages for smart contracts in cryptoeconomics; I am grateful for the exposure to his formal rigour in the research process, as it has been an invaluable learning experience.

I wish to extend a special thank you to Bernardo David, who co-supervised this PhD project and adopted me as a member of his cryptography research group at the IT University of Copenhagen; Bernardo has contributed some of the most celebrated work in energy-free, permissionless proof-of-stake consensus, and working under his tutelage opened up the investigation of many privacy-related research directions in this project; undoubtably, the role of advanced cryptography in our digital society cannot be underestimated, as it increasingly reconfigures *who can do what, from what*[3].

I am grateful for willing collaborators; in particular, Carsten Baum and Tore Frederiksen were both co-authors and generous with advice and guidance.

Anders Konring kindly provided the meticulous translation of the abstract into Danish.

This thesis is dedicated to my parents, sister and partner.

[1] Massimo Bartoletti, Carsten Baum, Bernardo David, Ittay Eyal, Tore Frederiksen, Tiantian Gong, Christian Janos Lebeda, Mariana Gama, Lorenzo Gentile, Tommi Junttila, Massimiliano Mirelli, Andreas Vandin

[2] Aalto University, Aarhus University, Alexandria Institute, Basic Algorithms Research - University of Copenhagen, IT University of Copenhagen, KU Leuven, Protocol Labs, Purdue University, Sant'Anna School for Advanced Studies, Technion - Israel Institute of Technology, University of Cagliari

[3] The Moral Character of Cryptographic Work, Phillip Rogaway

Contents

Part I

Introduction

1 Cryptoeconomics

1.1 Definition: Cryptoeconomic Layers 1 & 2

The term “Cryptoeconomics” is believed to have emerged from the early years of the Ethereum developer community. Cryptoeconomics is also the title of a rigorous, economic analysis of the Bitcoin protocol grounded in the principles of Austrian economics by Eric Voskuil [Vos22].

Towards defining the emerging *field or discipline* with which this thesis is concerned, we choose to reproduce the following definition from Brekke and Wassim [BA21] for its generality.

Definition 1.1 (Cryptoeconomics). Cryptoeconomics describes an interdisciplinary, emergent and experimental field that draws on ideas and concepts from economics, game theory and related disciplines in the design of peer-to-peer cryptographic systems. Cryptoeconomic systems try to guarantee certain kinds of information security properties using incentives and/or penalties to regulate the distribution of efforts, goods and services in new digital economies. Cryptoeconomics is an embryonic field at present and can be taken to include several areas of focus: information security engineering, mechanism design, token engineering and market design.

We categorize, as in [BA21], following layers of the crypteconomic stack.

- **Layer 1 (L1)** refers to the underlying, permissionless consensus system.
- **Layer 2 (L2)** refers to the token, market or mechanism capacities offered by emerging cryptoeconomic platforms.

We clarify commonly (mis)used terminology. **Blockchain** refers to a data structure, in which ordered transaction batches form an integrity-preserving hash-chain; it is implemented in *permissionless consensus* protocols (§3.3) where finality is probabilistic, and the protocol must permit parties to switch between *alternative chains* or transaction histories. The term *permissioned blockchain* is a misnomer, despite being widely used in marketing communication and even research papers. Consensus or agreement in the permissioned or authenticated setting (§3.1) is final and does not require blockchain data structures. **Smart Contracts** refer to user-defined programs deployed to a distributed virtual state machine realized by parties executing an instance of permissioned or permissionless consensus. The unit of interaction between client and smart contract is a **Transaction** authorized by a digital signature.

1.2 Open research problems

We briefly highlight general problem domains in Cryptoeconomics exposed by the emergence of permissionless consensus protocols in recent years and refer to the future work section in Chapter 6.

(1) Formalization. Cryptoeconomic systems are generally under-formalized in practice. The Bitcoin protocol has no formal specification; the original open-source implementation is simply referred to as the *reference implementation*. A lack of protocol formalization continues on layer 2, where the infamous DAO bug[1] resulted in roll-back of the Ethereum blockchain, violating the spirit of a permissionless “world computer”. Formalization of layer 1 and 2 protocols are a necessary requirement for rigorous security analysis and verification of systems which realize aggregate economic value in the vicinity of the annual gross domestic product (GDP) of small to medium-sized, industrialized countries.

(2) Rationality. In the permissionless setting (§3.3.1), identity and authentication cannot be assumed; this stands in stark contrast to protocol design in the classical setting, where identities are known and communication is authenticated (§3.1.1). Still, permissionless protocols require an honest threshold of participants to behave honestly and according to protocol. In particular, the most effective way to “engineer honest behaviour” without identity is to ensure that it represents the *winning strategy* for rational agents; such a protocol is *incentive-compatible*. However, widely deployed cryptoeconomic protocols have been shown to violate incentive compatibility, such as Bitcoin (§3.3.3). Many urgent questions on the secure composition of incentives arising from cryptoeconomic layers 1 and 2 remain open.

(3) Privacy. Despite the obvious reference to cryptography, permissionless cryptoeconomic layer 1 systems offer *no privacy guarantees* by default; still, *most internet applications require some form of privacy to be meaningful*. The most widely deployed application class in cryptoeconomics is currently called Decentralized Finance; here, the lack of privacy results in systemic front-running of user inputs, taxing the utility of the system for all participants. We provide an overview of privacy-enhancing techniques for cryptoeconomic systems in Chapter 5, and highlight open challenges.

(4) Scalability. Permissionless consensus is synchronous and must be parameterized by the worst-case network delay (§3.3). This implies a constraint on its inherent scalability; proposals to scale throughput often introduce trusted third parties for security and/or liveness.

In this thesis we investigate research questions concerned with problem domains (1), (2) and (3). We motivate the specific research questions addressed by this thesis in Chapter 2.

[1] https://en.wikipedia.org/wiki/The_DAO_(organization)

2 Thesis Overview

All papers in this thesis were authored within the duration of a 3-year Danish PhD, which included a 6 month research visit to Technion in Israel, hosted by Ittay Eyal. In this chapter, we provide a high-level discussion of the research questions addressed by the works in this thesis, and point to relevant background in Part II for domain context.

A snapshot of the publication status of works in this thesis is provided below.

RQ	Manuscript	Status	Venue(s)
1	A theory of Automated Market Makers in DeFi (P1a/P1b)	C: ●	COORDINATION'21
		J: ●	LMCS
	SoK: Lending Pools in Decentralized Finance (P2)	W: ●	WTSC'21
2	Formal Analysis of Lending Pools in Decentralized Finance (P3)	C: ●	ISoLA'22
3	Maximizing Extractable Value from Automated Market Makers (P4)	C: ●	FC'22
4	SoK: Mitigation of Front-running in Decentralized Finance (P5)	W: ●	DeFi'22
5	FairPoS: Input Fairness in Proof-of-Stake with Adaptive Security (P6)	C: ○	-
6	Eagle: Efficient Privacy Preserving Smart Contracts (P7)	C: ●	FC'23
7	Differentially Private Market Mechanisms with MPC (P8)	C: ○ W: ◐	DeFi'23 - prelim results
8	SoK: Privacy-Enhancing Technologies in Finance (P9)	C: ○ W: ◐	DeFi'23 - full paper talk

Conference, Journal, Workshop,
● Paper published, ○ Paper under submission, ◐ Peer-reviewed talk

2.1 Research questions & thesis contributions

Our investigations begin with a novel application class called Decentralized Finance (§4) - or DeFi for short - that first emerged on permissionless consensus protocols (§3.3). Such applications differ significantly from their counterparts in traditional finance, since they must satisfy the computational, storage and communication constraints imposed by permissionless consensus. See §4 for a discussion on how these constraints influence application design in DeFi. Furthermore, such applications deployed to the permissionless setting must incentivize rational third parties to perform indended actions to preserve economic safety. As a consequence, many of these applications feature interesting and novel designs, yet the lack formal specifications of their key functionalities make any security analysis challenging. Thus, we state our first research question as follows.

RQ 1: What are formal semantics & desired safety properties of DeFi?

We answer RQ 1 by providing the first formal models of the two most widely used DeFi applications, namely Automatic Market Makers (P1a/P1b, background in §4.2.2) and Lending Protocols (P2, background in §4.2.3). We formally specify their key functionalities and economic security properties with appropriate abstraction to apply across different implementations; for example, key security properties of our lending protocol model apply to all implementations parameterized with different interest rate functions.

Indeed, applications in DeFi must be appropriately parameterized in order to satisfy their intended economic security properties. In particular, lending protocols in DeFi are exposed to volatile token prices, which make formal security analysis challenging; the security of *liquidations* in lending protocols is highly dependent on the appropriate choice of protocol parameters when exposed to stochastic price behaviour. Thus;

RQ 2: Can lending protocols be securely parameterized with automated formal verification?

Given the probabilistic nature of such systems, we investigate how stochastic model checking can be employed to harden liquidation security of lending protocols in P3 for chosen volatility regimes. This work combines a formal verification environment implementing the lending protocol semantics formally specified in P2, with a recently proposed statistical model checking framework.

Another major security challenge in DeFi arises from the lack of input fairness in permissionless consensus systems (background in §3.4), allowing the adversary to censor, order and inject its own transactions to the blockchain. This permits the block leader in permissionless consensus to extract financial gain from user actions submitted to Automatic Market Makers, motivating our following inquiry.

RQ 3: What is the optimal front-running attack on AMM's?

We answer this research question in P4 (background in §4.3) with surprising results; we introduce the optimal multi-layer sandwich attack that considers all action types that users can submit to automatic market makers and name it the *Dagwood Sandwich*. Our results show that the adversary is never incentivized to include redeem actions, implying potential difficulty for liquidity providing agents to retrieve or reallocate their liquidity. In addition to financial loss for the honest user, front-running introduces *artificial demand* for block space, implying a utility tax on all users (see §4.3). Naturally, this leads us to ask;

RQ 4: How can front-running in DeFi be mitigated?

In P5 we survey and expose short-comings of folklore mitigation techniques such as *commit-and-reveal*, where the adversary can induce selective aborts to its benefit. We survey proposals to introduce input fairness; however, such approaches introduce assumptions from the authenticated setting (background in §3.1.1). Thus, in order to achieve input fairness in the permissionless setting, we ask;

RQ 5: Can fair input ordering be realized in the permissionless setting?

We answer RQ 5 in the affirmative with FairPoS in P6, the first consensus protocol which achieves input fairness in the permissionless setting (background in §3.4); this is accomplished my means of *delay encryption* [BF21] and a novel protocol gadget named "longest *extendable* chain selection" which mitigates adversarial attacks to desynchronize expensive, honest key extraction processes by means of delaying adversarial blocks.

In P5 we also show that input fairness is not sufficient to mitigate front-running in some cases; public balances can reveal the "direction" of a trade in decentralized finance, permitting a class of front-running attacks even when input privacy is guaranteed. More generally, input fairness provides *no privacy guarantees* on state or outputs from smart contracts. "Fully private" smart contracts enabled by a trusted curator have been proposed in Hawk [KMS+16]; however, distributing the trusted curator for security was previously thought to be impractical with secure multi-party computation (MPC), due to high cryptographic overhead imposed on the MPC evaluation, motivating the following line of research;

RQ 6: Can privacy-preserving contracts be realized in a practical manner?

We propose Eagle in P7, the first practical realization of privacy-preserving smart contracts with general expressiveness and support for private state for each user, that succeeds in moving all the cryptographic overhead outside the MPC evaluation. Eagle improves on [KMS+16] with long-running private smart contract instances; our model permits clients to outsource computation to a static MPC committee, and lazily collect their private outputs from the blockchain without server interaction, even if these go offline.

We argue that the practical setting of Eagle opens up a novel design space for cryptoeconomic applications by supporting private interaction with smart contracts and computation on private state fragments. In particular, we revisit *dark pool* designs in traditional finance, which permit traders to execute trades in a privacy-preserving manner; such dark pool designs have recently been implemented in MPC with impressive real-world throughput [dGCSA22, dGCP$^+$22] for the traditional finance setting, but we observe that any market mechanism based on order-matching will always leak information about the counter-party; an executed trade *necessarily implies* an executed trade in the opposing direction. Thus, an adversary actively participating in a thinly traded dark pool can directly infer the execution of the honest trader. Consider an honest user trading a large volume of a given asset over period a fixed period time (time-weighted average price strategy). Here, the adversary observing such an in-progress execution can anticipate and front-run any future honest trades executed periodically. We investigate whether market mechanisms exist which provide "pre- and post-trade" differential privacy [DR$^+$14] over multiple rounds; the classic notion of differential privacy protects the presence of a single trader in each interaction round and provides rigorous guarantees for privacy leakage.

RQ 7: Can differentially-private markets be realized in the setting of Eagle P7?

In "Fuzzy Order Matching" (P8), we introduce the first differentially private market mechanisms (background in §5.3.2) which can be applied equally to the dark pool setting in traditional finance, or privacy-preserving smart contracts with Eagle [BCDF22] in decentralized finance. We introduce an extended definition of differential privacy extended to the novel setting of a "trusted curator" (background in §5.3.1); such a notion must protect the privacy of inputs, but also ensure that *correlated outputs* do not violate chosen privacy parameters. We argue that differentially privacy in the trusted curator model is useful for many other applications involving secure multi-party computations with correlated outputs.

Finally, we survey the landscape in both traditional and decentralized finance and investigate the wider potential of applying privacy-preserving techniques;

RQ 8: What problems can privacy-enhancing techniques solve in finance?

In P9, we survey privacy-enhancing techniques in the following categories in finance; identity, KYC, AML, GDPR, digital asset custody, markets & settlement and future applications. We observe that the full spectrum of zero-knowledge, secure multi-party computation, fully homomorphic encryption, threshold secret sharing primitives are proposed to solve practical, privacy-related challenges in (decentralized) finance; we anticipate increased interest and adoption of such advanced cryptographic techniques in practice, which were previously mostly of theoretical interest less than a decade ago.

Part II

Background

3 L1: Input Fairness in Consensus

This chapter provides background and motivation for the following research question[1];

RQ 5: Can fair input ordering be realized in permissionless consensus?

We highlight fair input ordering as a consensus property with key significance for application security in *layer 2* of the cryptoeconomic stack; a lack of input fairness in consensus or *layer 1* is exploited by economic actors interacting with *layer 2* applications (see background in §4.3).

Notions of input fairness include *encrypt-and-reveal*, where blinded inputs are first finalized before they are unblinded for all parties; we achieve the first permissionless consensus protocol which guarantees such a notion of input fairness in FairPoS (P6). Another notion of input fairness is *receipt-order-fairness*, where the arrival time of inputs at parties executing consensus determines the final ordering; this fairness is only meaningful in the permissioned setting, where a weaker network adversary is assumed that cannot manipulate arrival time of messages received by consensus nodes. Thus, input fairness and its applicability differs between settings of *authenticated* (§3.2) and *permissionless* (§3.4) consensus. Even today, many works do not formally delineate between the two, referring to both classes as “blockchain protocols”, an ambiguous term applicable to either setting.

We dedicate a significant portion of this chapter to providing a minimal introduction to both authenticated (§3.1) and permissionless consensus (§3.3) for the interested reader; the juxtaposition of both settings illustrates trade-offs made by permissionless systems, and may provide valuable context for discussions of input fairness in each model.

The expert reader may wish to jump directly to the discussion of input fairness in the authenticated (§3.2) and permissionless (§3.4) model and our contributions to achieve input fairness in FairPoS (P6) for the permissionless setting.

[1]See Chapter 2 for an overview of research questions addressed by this thesis

3.1 Authenticated consensus

In this section, we state the underlying assumptions of the authenticated setting (§3.1.1) and provide a minimal introduction to protocols which achieve authenticated consensus in both synchronous (§3.1.3) and asynchronous models (§3.1.4). Proposals to achieve *input fairness* in the authenticated setting are described in §3.2.

We hope this introduction to authenticated setting will provide useful context in its relation to the permissionless setting, with respect to differing notions of *input fairness* that may be achieved. Examples from this section are adapted from [Wat17] and [Shi20] for our purposes.

3.1.1 Authenticated setting

In the authenticated setting, the following must hold for each participant.

1. Authenticated channels have been established with all parties
2. Identities and numbers of participants are known
3. A public key infrastructure functionality is available

This is a realistic notion in the context of a data center operated by known entities, but may be too difficult to achieve over the public internet with ephemeral participants which are not known a priori. Still, it is in this authenticated setting that the notion of byzantine fault tolerant state machine replication was first conceived and realized; furthermore, in contrast to permissionless consensus, these can be realized in both the synchronous and asynchronous model; the latter permits the protocol to execute as fast as network latency allows, enabling protocols with much higher throughput.

3.1.2 State Machine Replication

We restate the standard definition of bbyzantine fault-tolerant state machine replication (SMR).

Definition 3.1 (**Byzantine fault-tolerant SMR**)**.** A state machine replication protocol is run by n servers of which f are byzantine corrupted. Honest servers initialize their local state to a common Γ_0 and agree on a sequence of client inputs; for each input x in the finalized order, a deterministic state transition function T is applied to the current local state Γ and input x to obtain both an updated state and output value, namely $(\Gamma', y) = T(\Gamma, x)$. Let $\Gamma_0 \rightarrow^{T_x} ... \rightarrow^{T_{x'}} \Gamma'$ denote a valid history of state updates on Γ_0 induced by input sequence $\mathbf{x} = (x, ..., x')$ and T.

- *Consistency.* For any two honest servers, the local input history of one party is the prefix of the input history of the other party.
- *Liveness.* There exists a polynomial function $\mathsf{Confirm}$, such that if an honest player sees an input, it will be applied to the state update history within $\mathsf{Confirm}(\Delta)$ time, where Δ is the worst network delay induced by the adversary in an execution.
- *Finality.* For any single honest server, there exist a local input history prefix, which is also the prefix of all local update histories in the future.

Byzantine state machine replication protocols (BFT-SMR) are closely related to single-shot Byzantine Broadcast (BB) and Byzantine Agreement (BA) protocols. We restate standard definitions of BB and BA and then sketch how BFT-SMR can be constructed from these primitives in the synchronous model.

Definition 3.2 (**Byzantine Broadcast**)**.** BB is run by n parties, of which f are byzantine corrupted. A single sending party i receives an input to activate the execution, for which the following holds;

- *Agreement.* All honest parties output the same value.
- *Validity.* If the honest sender receives input x, all honest parties will output x.
- *Termination.* If the sender is honest, all honest parties output (and terminate).

Definition 3.3 (Byzantine Agreement). BA is run by n parties, of which f are corrupted. Each party i receives an input to activate the execution, for which the following holds;

- *Agreement.* All honest parties output the same value.
- *Validity.* If all honest parties receive input x, all honest parties will output x.
- *Termination.* All honest parties output (and terminate).

3.1.3 Synchronous model.

We begin with the synchronous model towards realizing authenticated consensus and BFT SMR. We restate the standard definition of the synchronous model.

Definition 3.4 (Synchronous model). Protocols in the synchronous models proceed in rounds; any message sent in a round is delivered by the beginning of the next round. Thus, messages are delivered by the network adversary $\mathcal{A}$ within a finite delay Δ that is *known* to all participants, who have synchronized clocks;

We illustrate a naive attempt at synchronous BB tolerating $f \leq n-1$ for $n \leq 3$.

Example 3.5 (Synchronous BB for $n \leq 3$). In the first round, let the sending party **signs its input** and sends it to each other party. In the second round, all servers forwards the signed message received in round 1 to all other servers. In the final round, a server will output the identical, signed message it received from all other parties, or output nothing. This protocol achieves byzantine broadcast (Definition 3.2) for $n \leq 3$:

- *Agreement.* If $f = 1$, where the sender is corrupt, honest receivers can detect inconsistent, signed messages. If a single receiver is corrupt, it cannot forge signatures and can only remain silent. If $f = 2$, only a single honest server remains; agreement is trivially achieved.
- *Validity.* If the sender is honest, all other honest parties will receive a single signed message only, and output this.
- *Termination.* If the sender is honest, the synchrony of the protocol implies termination.

Example 3.5 fails to achieve byzantine broadcast tolerating $f \leq n-1$ for $n > 3$; consider a colluding sender and receiver for a protocol execution with $n > 3$ parties. In round 1, the sender signs two inconsistent messages and delivers $\mathsf{sig}(m)$ to honest servers and $\mathsf{sig}(m')$ to the colluding server. In round 2, the colluding server forwards $\mathsf{sig}(m')$ to a *subset* of the honest servers only. Thus, in the final round, a subset of honest parties has received inconsistent, signed messages (m, m') and the other subset has only observed a single, signed message (m), breaking agreement.

The Dolev-Strong protocol [DS83] overcomes this attack by requiring each party to append its own signature when forwarding a message in a given round; a valid message in each round must include the sender signature appended with signatures contributed by an additional forwarding party in each prior round. Thus, it is no longer possible to inject inconsistent messages without prior detection in the last round, thereby achieving synchronous BB tolerating $f \leq n-1$ for all n. Dolev-strong [DS83] implies;

Theorem 3.6 (Synchronous BB corruption). *Synchronous BB tolerates up to $f \leq n-1$, assuming signatures.*

Next, we sketch how synchronous agreement can be constructed from synchronous broadcast with the following example.

Example 3.7 (**Synchronous BA**)**.** We construct sync BA from sync BB; upon receiving an input, each party initializes an instance of synchronous BB (Example 3.5) with all other parties. Upon termination of each BB sub-protocol instance, each party outputs the majority output from all BB instances.

Together with [DS83], this implies the following;

Theorem 3.8 (**Synchronous BA corruption**)**.** *Synchronous BA tolerates up to $f < n/2$.*

Proof. (Sketch) Assume an execution of Π_{syncBA} with corruption $f = n/2$. All parties receive an input value 1, but the corrupted parties immediately flip the input bit to 0 after receiving it; all parties then proceed to execute Π_{syncBA} honestly with inputs 0 and 1 respectively. Without an honest majority, protocol Π_{syncBA} has insufficient information to decide an output that is always consistent with the input received by all honest parties. □

Example 3.9 (**Synchronous byzantine SMR**)**.** We can now construct synchronous BFT SMR synchronous setting from BB; upon receiving inputs, each server initializes an instance of BB, such that all honest servers obtain the set of honest inputs. Then, honest parties simply apply a canonical ordering on the inputs output by the BB sub-protocol instances, and update the local state machine accordingly.

3.1.4 Asynchronous model.

Next we introduce the asynchronous network model, as it permits modern asynchronous consensus protocols to execute as fast as network communication permits; parties execute their protocol task upon receiving an input or message, and do not wait for time-outs to perform protocol actions. We restate the definition of the asynchronous model.

Definition 3.10 (Asynchronous model)**.** In the asynchronous model, protocols permit participants to be invoked upon receiving a message from the network, to then perform computation and output messages to send. Participants do not have access to synchronized clocks, and cannot be activated following time-outs.

From [DLS88], we restate;

Theorem 3.11. *Asynchronous BB tolerates up to $f < n/3$.*

Proof. (Partial sketch) We illustrate the impossibility of asynchronous BB for the case $n = 3$ and $f = 1$, where the broadcasting party is corrupt. Towards contradiction, let us assume an instance of asynchronous BB tolerating $f = n/3$. Here, the adversary executes the asynchronous BB protocol, but ensures that the communication between the two honest parties is delayed by a large amount relative to the fast communication between itself and the individual honest parties. It then initializes the BB protocol with input 0 towards one party and with input 1 towards the other honest party.

Since we have assumed a protocol tolerating $f = n/3$, both honest parties must terminate before receiving any messages from each other, as a long delay cannot be distinguished from silence, and communication with the corrupted broadcaster is fast. Thus, one honest party outputs 0, and the other honest party outputs 1, contradicting agreement.

□

Early examples of asynchronous agreement were achieved by Ben-Or [BO83] and Bracha [Bra84]. For completeness, we note deterministic asynchronous BA is impossible requiring randomized agreement algorithms [FLP85]. Efforts to advance the state-of-the-art have not subsided; the field has arguably been reinvigorated by the advent of permissionless consensus protocols. From [DLS88], we restate;

Theorem 3.12. *Asynchronous BA tolerates up to $f < n/3$.*

Any asynchronous state-machine-replication protocol cannot contradict the corruption threshold of Theorem 3.12, as any state-machine-replication trivially realizes the original, one-shot notion of byzantine agreement in Definition 3.3.

We highlight the asynchronous PBFT protocol as the gold standard for asynchronous state-machine-replication, a seminal work of Castro and Liskov [CL$^+$99]. A dedicated server plays the role of a leader, proposing a block of received client inputs. Corrupt leader behavior is detected by honest servers, triggering a view-change sub-protocol, upon which a new view with another leader is initiated.

Recent BFT-SMR protocols Tendermint [BKM18] and Hotstuff [YMR$^+$19] introduce *rotating leaders*, eliminating expensive view-changes when the leader is corrupted; the rotating-leader paradigm is achieved with additional overhead in comparison to the stable leader view.

Leader-based Consensus Since consensus protocols are leader-based, it is important to note that the leader or primary can arbitrarily decide the ordering of each batch of committed inputs. Any biased ordering or censoring of inputs remains undetectable if the primary runs honestly otherwise. Such "dishonest" behaviour motivates the following notions of *input fairness* in authenticated consensus.

3.2 Input fairness in authenticated consensus.

In leader-based consensus, the leader has the power to (1) inject its own inputs, (2) observe all submitted inputs prior to finalization, (3) censor inputs it receives and (4) decide on a final order of the input batch committed to the log.

As we describe in §4.3.1, this power can be exploited by a rational adversary when implementing economic applications on the replicated state machine. We first describe several **folklore solutions** to achieve input fairness in the authenticated setting and highlight their shortcomings.

- *Time-stamped inputs.* Each node may sign its own inputs with the current local time-stamp, requiring each finalized input batch to respect the chronological order of the respective time-stamps. However, a leader colluding with another client can ensure that a malicious input is signed with a (manipulated) time-stamp which can then be ordered before or after any other observed input.
- *Commit-and-reveal.* All parties first individually commit their inputs, and wait until the commitments are finalized. The commitments are then opened to reveal the inputs which are logically ordered according to pre-determined canonical ordering. This approach suffers from *selective abort*; the adversary can decide whether to reveal or abort *after* observing the commitment openings of all other parties.
- *Rotating leaders.* At a cost of protocol complexity, rotating leaders can reduce the number of leaders which exploit censoring and ordering powers, but no input fairness guarantees can be made in any given round.

Several orthogonal notions of input fairness have been proposed in the setting of authenticated consensus, which mitigate the power of the adversary to (1) observe the plaintext of honest user inputs and (2) decide on any arbitrarily input ordering.

1. **Encrypt-and-reveal.** Several lines of work [MXC+16, ACG+18] avoid *selective abort* from commit-and-reveal by means of threshold encryption. Here, the secret key is distributed across its members; clients encrypt their inputs to the threshold public key and decryption follows from an interactive protocol by committee members holding the secret shared key material. If the corruption threshold is respected, no malicious party can prevent the decryption of inputs.

   Alternatively, several proposals realize *encrypt-and-reveal* by piggybacking on the presence of a randomness beacon or agreement protocol (in the authenticated setting). In Fairblock [MGZ23], the presence of a BLS-based randomness beacon is sufficient to realize identity-based-encryption (IBE) [BF01]; the signature of a given round number that is released by the beacon committee enables decryption of any message encrypted to that round number. McFly [DHMW22] enables a similar notion of *encrypt-and-reveal* but only assumes the presence of a committee performing authenticated consensus.

2. **Receipt-order fairness.** Recent works [KZGJ20, CMSZ22, Kur20] implement a fairness notion based on the "receipt time" of client inputs. A naive implementation permits all members of the consensus committee to broadcast their local view of received client inputs ordered by receipt time, and then to apply a deterministic ordering algorithm over all "receipt-order-views" received from committee members to arrive at a final, fair ordering of inputs. The notion of γ**-receipt-order** fairness introduced by [KZGJ20] guarantees agreement on a relative ordering between two inputs if observed by a γ fraction of nodes.

   We note that protocols realizing receipt-order-fairness cannot protect against the adversary controlling the network between clients and servers running agreement. Instead, it is generally assumed in [KZGJ20, CMSZ22, Kur20] that the network between client and committee is not adversarially controlled, restricting the meaningfulness of receipt-order fairness to a very specific setting with a **limited network adversary**.

We emphasize that notions of input fairness in the authenticated setting do not trivially apply to the permissionless setting, introduced in the next section. We highlight our contribution of realizing the first encrypt-and-reveal scheme for permissionless consensus and refer to §3.4 for relevant background and the FairPoS manuscript P6.

3.3 Permissionless Consensus

In this section, we clarify the underlying assumptions of the permissionless setting, in which consensus was previously considered impossible (§3.3.2), and provide an overview of the different branches of permissionless consensus and their inherent trade-offs. In contrast to authenticated consensus, known permisionless protocols are **synchronous**, as they are parameterized by worst-case network delays and therefore do not achieve throughput of modern, asynchronous consensus systems in the authenticated setting. The main branches of permissionless consensus protocols are;

- **Nakamoto proof-of-work** (§3.3.3)
- **Nakamoto proof-of-stake** (§3.3.4)
- **Proof-of-stake + Byzantine Agreement** (§3.3.5)

In section §3.4, we discuss input fairness for the different flavours of permissionless consensus to motivate a key contribution in this thesis; to the best of our knowledge, FairPoS (P6) is the first consensus protocol to achieve input fairness in form of *encrypt-and-reveal* in the permissionless setting.

3.3.1 Permissionless setting

The permissionless setting describes a model of protocols executed on the public internet without the presence or setup of trusted authorities to authenticate users; without identification, (1) sybil entities can be generated arbitrarily and (2) the true number of participants remains unknown. Furthermore, (3) nothing can be assumed about the reliability of participants in any protocol role; they can disappear at any point during the protocol execution and must therefore be replaceable.

1. No authentication of parties.
2. Number of parties are unknown.
3. Player replaceability. Protocol participants are replaceable anytime.

Nakamoto consensus protocols (§3.3.3, §3.3.4) satisfy all criteria for the permissionless setting, whereas Proof-of-stake + BA (§3.3.5) sacrifices (2) to achieve finality of agreement; such a protocol instance must be parameterized by the number of online participants.

3.3.2 Impossibility of unauthenticated agreement

Prior to Bitcoin [Nak08], unauthenticated consensus was deemed impossible. We restate and simplify following theorems from [BCL+05] and [PS17b] respectively.

Theorem 3.13. *Unauthenticated agreement in synchrony is impossible.*

Proof. (Sketch) Without authenticated communication, the adversary can trivially simulate any protocol role as honest parties cannot distinguish honest and adversarial messages. Assuming a protocol featuring a fixed set of roles, the adversary can split any honest set into disjoint subsets, each running “independent” executions without honest majority. Thus, these subsets cannot agree. □

Theorem 3.14. *Unauthenticated agreement in asynchrony is impossible, even if the adversary does not corrupt any parties.*

Proof. (Sketch) In the asynchronous setting, the adversary can delay messages indefinitely; it splits the honest parties into two camps and permits the network delay for messages sent within each camp to be very small. Messages sent between the honest camps are delayed "indefinitely". The two honest camps performing *asynchronous* agreement must terminate and output as "fast" as the communication within a camp permits, as they are asynchronous protocol executions. If the two honest camps receive different inputs, they will therefore output different values before any messages between the two camps are delivered by the adversary, violating agreement. □

These impossibility results are overcome by permissionless consensus protocols in the subsequent section with assumptions such as honest majority of computational power or honest majority of online coin-holders.

3.3.3 Nakamoto Proof-of-Work

As defined in §3.3.1, the permissionless setting assumes that the number of online parties participating in the protocol be unknown. Bitcoin overcomes the impossibility of unauthenticated consensus highlighted in §3.3.2 by means of assuming honest majority of computation or **proof-of-work**; here, a participant must provably expend an expected amount of computation work in order to append a valid input batch or block to a **blockchain**.

We defer to [GKL15, PSS17, Ren19, KRS18] for a full formalization of the Bitcoin consensus or Nakamoto proof-of-work, but emphasize the key protocol properties required for realizing state machine replication (Definition 3.1).

In Nakamoto proof-of-work, parties initialize their local view of the state machine to an agreed upon **genesis block**. Throughout the protocol execution, each generated block containing user inputs must unambiguously point to another block, resulting in blockchains rooted in genesis. Honest block leaders (elected by means of proof-of-work) will always extend the **longest chain** in its local view, which represents the canonical input or transaction history. Dishonest block leaders may extend any (sub)chain of blocks, potentially resulting in chain forks; this is indistinguishable from forks from simultaneous generation of honest blocks or forks resulting from inconsistency in local views caused by delayed delivery of messages, slowing the propagation of new blocks across the network. The security of Nakamoto consensus is over defined by **structural properties** of the local **blocktree** of honest parties.

For an execution of Nakamoto consensus of duration T, we restate from [GKL15] widely adopted notions of consistency (or common-prefix), chain quality and chain growth as security properties over honest blocktree views.

Definition 3.15 (k-Common Prefix)**.** With overwhelming probability (in T and k), at any point, the longest chains of two honest players can differ only in the last k blocks. The shared longest chain prefix is called k-common-prefix.

Definition 3.16 (μ-Chain Quality)**.** With overwhelming probability (in T), for any T consecutive messages in any longest chain held by some honest player, the fraction of messages that were "contributed by honest players" is at least μ.

Definition 3.17 (τ, s-Chain Growth)**.** With overwhelming probability (in T), at any point in the execution, the longest chain of honest players grows by at least $\tau \cdot s$ blocks in the last s rounds; τ is called the chain-growth of the protocol.

Corollary 3.18. *$CP \wedge CQ \wedge CG \implies$ Consistency $\wedge$ Liveness.*

Proof. (Sketch) Chain growth (Def. 3.17) and quality (Def. 3.16) necessarily implies liveness, as honest blocks are guaranteed to be appended to the common-prefix; k-common prefix (Def. 3.15) implies consistency. □

Assuming equal distribution of computational power, let us denote the expected block arrival rate as $\lambda = n \cdot p$, where n is the number of online parties and p the probability for a single party of generating a valid block in each round. Further, let Δ denote the maximum message delay in a synchronous network model. Both λ and Δ are protocol parameters which must be fixed as a function of the honest mining fraction $\alpha > 1/2$ for security (CP, CQ and CG) to hold in Nakamoto proof-of-work.

Theorem 3.19 (**Nakamoto proof-of-work satisfies CP, CQ and CG**). *For any honest mining fraction $\alpha > 1/2$, there exists parameters λ and network delay Δ such that Nakamoto proof-of-work satisfies k-CP, CQ and CG with overwhelming probability in runtime T and k.*

We reproduce Figure 3.1 from [DKT+20] which plots the security thresholds for nakamoto proof-of-work established in the following works [GKL15, PSS17, KRS18, Ren19, DKT+20], which relate α, λ and Δ for a secure parameterization of Nakamoto PoW. Below we highlight the main *proof*

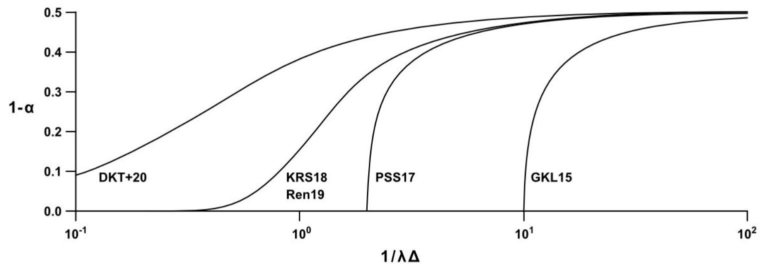


Figure 3.1: Security bounds for Nakamoto PoW from [GKL15, PSS17, KRS18, Ren19, DKT+20] relate the adversarial computational fraction $(1 - \alpha)$ to the block interval normalized by network delay $(1/(\lambda\Delta))$.

techniques from aforementioned works based on (1) honest convergence opportunities and (2) private double-spend attacks.

1. **Convergence opportunities** are a key proof technique deployed in [GKL15, PSS17, Ren19, KRS18]. Let a convergence opportunity be $\mathsf{converge}_{\mathcal{H}} = \mathsf{silence}_{\Delta} :: \mathsf{uniqueBlock}_{\mathcal{H}} :: \mathsf{silence}_{\Delta}$, where $\mathsf{silence}_{\Delta}$ denotes an event where no blocks are generated in period Δ, and $\mathsf{uniqueBlock}_{\mathcal{H}}$ the discovery of a *single* honest block. Such an event guarantees that all honest users converge on the longest chain in their local view following the first silence period of Δ, as it provides sufficient time for honest parties to share local block tree views over the network. Then the isolated honest block occurrence extends this longest chain and is also seen by all honest parties following another silence period of Δ. Informally, such a convergence event can be leveraged to demonstrate consistency, chain growth and chain quality *without reasoning about adversarial strategies*. Assume k distinct occurrences of $\mathsf{converge}_{\mathcal{H}}$ in the absence of adversarial blocks; this implies k-consistency, quality and chain-growth for honest users. To break k-consistency, $\mathcal{A}$ must interrupt each of the k occurrences of $\mathsf{converge}_{\mathcal{H}}$ with an adversarial block. The works above prove this does not occur with overwhelming probability in runtime T and k.
2. **Private double spend attacks** are proven in [DKT+20] to represent the *worst adversarial attack* in Nakamoto consensus, and thus bounding the success probability of such attacks implies bounding probability of violating security. Double spend attacks were conjectured in the original Bitcoin whitepaper [Nak08] to represent the worst adversarial strategy, but this was not formally proven. Thus, the security threshold for Nakamoto PoW demonstrated in [DKT+20] and plotted in Figure 3.1 is optimal.

Nakamoto PoW is permissionless. Nakamoto PoW is a permissionless protocol (§3.3.1). (1) Parties require no authentication; any party can contribute proof-of-work which is publicly verifiable. (2)

The number of online parties is unknown; the protocol is parameterized by total online *hash-rate*, implied by the block arrival rate λ. (3) Any party can contribute a single message (or block) and be replaced by another online party thereafter.

Nakamoto PoW is synchronous. A secure Nakamoto-PoW protocol is parameterized by honest hash-rate fraction $\alpha > 1/2$, block arrival rate λ and maximum network delay Δ (Theorem 3.19). Since parameterizing Nakamoto PoW secure implies knowing Δ, the protocol is synchronous (§3.1.3).

Nakamoto PoW is not incentive compatible. As first shown in [ES18], there exist adversarial strategies for parties executing Nakamoto PoW which would be preferrable to honest behaviour for a economically rational parties. This is particularly challenging in the *permissionless setting* without identities and reputation; specifying honest protocol behaviour which also represents the *best choice* for rational parties is desirable, but remains an open problem in the design permissionless consensus protocols. The Fruitchain protocol [PS17a] extends and improves Nakamoto PoW in this regard. It achieves **δ-approximate fairness** for honest mining behaviour; here, an α honest fraction of computational power receives $(1 - \delta)\alpha$ fraction of monetary reward.

3.3.4 Nakamoto Proof-of-Stake

Ouroborous [KRDO17] introduced the first Nakamoto Proof-of-Stake consensus protocol in the permissionless setting. In proof-of-stake, each coin is granted an equal probability of being elected as block leader in a given slot. The obvious advantage of such an approach is elimination of "wasted compute" and implied energy consumption. Nakamoto PoS is similar to PoW in that honest parties always extend the **longest chain** in their local **blocktree** view; thus, the security of Nakamoto PoS can be defined over the same structural properties of local block trees of honest users, as formalized in definitions 3.15, 3.16 and 3.17.

Leader election in Proof-of-stake requires each party to register a verification key for a *verifiably random function* (VRF); this function behaves similar to a keyed pseudo-random function, but also outputs a proof which allows the owner of the verification key to attest the validity of a VRF output relative to its pre-image.

At each slot or protocol round, each online party, with its private key, computes a VRF on the current block, parent block pointer and current round number, which outputs a pseudorandom value; similarly to PoW, a VRF threshold parameter determines whether the output was "successful"; in this case, the party is elected block leader and is permitted to extend a blockchain in its local view.

Nakamoto PoS assumes availability of stake Informally, Nakamoto PoS can be considered a permissionless consensus protocol in a *weaker form*. In particular, joining the protocol execution requires availabiltiy of "coin stake"; if no other party is willing to transfer or sell coins, then the *player replaceability* property in §3.3.1 cannot hold.

Simulation attacks Since VRF evaluations are "free" and "immediate", nothing prevents the adversary from simulating an alternative protocol execution beginning from the genesis block, indistinguishable from any other honest execution. This is possible because Nakamoto consensus relies on the *longest chain rule*; any longest chain appearing in the local view will be adopted, even if new and old chains diverge significantly. Recent works [DKT21] have been proposed to mitigate such attacks in the permissionless setting with verifiable delay functions, imposing a *temporal cost* to each VRF evaluation, thereby thwarting a simulation attack to produce a new longest chain, as the production of the adversarial chain cannot outpace that of the honest one.

3.3.5 Proof-of-Stake + Byzantine Agreement

Algorand [GHM+17] introduces an alternative permissionless consensus protocol which differs from Nakamoto, longest chain variants, which we denote Proof-of-stake + Byzantine Agreement; instead of electing block leaders in each round of the protocol, a fresh committee is elected to perform each step of a synchronous Byzantine Agreement protocol; player replaceability (in §3.3.1) is achieved be allowing users to emit a single message in each BA role, and then to go silent. Upon completion of the agreement phase, the block of inputs is finalized; in contrast to Nakamoto consensus, finality holds with probability 1 once agreement is completed. We briefly highlight the trade-off made to achieve finality; namely, that the number of honest, online participants must be known in order to ensure safety in Proof-of-Stake + BA, thereby violating condition (2) of the the permissionless setting in §3.3.1.

Number of online parties is known. For simplicity, we assume a single online player for each staked coin. Thus, for a number of n online staking parties, each of the n parties participates in committee election with equal probability of success. We now demonstrate that the number of online parties must be known to safely parameterize a PoS + BA protocol instance.

Note that the BA protocol in PoS+BA must be parameterized by the threshold of votes required for agreement. For an expected committee size of τ and required honesty fraction h, let the number of honest votes required for BA agreement be fixed at $h\tau$.

Towards bounding the probability that the honesty threshold is maintained in any elected committee throughout the execution, we first restate the probability of electing exactly τ members from a set of n online parties below, where the probability of election for each party is given by p;

$$\binom{n}{\tau} p^{\tau}(1-p)^{n-\tau}$$

For the safety of the protocol, we consider the probability of the "catastrophic event" in which fewer than $h\tau$ honest parties are elected to the committee to illustrate the dependency of secure protocol parameterization on the number of online parties. Assuming hn honest online parties participating committee election and election probability $p = \tau/n$, the probability of fewer than $h\tau$ elected members is given by the following probability F.

$$F = \sum_{k<h\tau} \binom{hn}{k} (\frac{\tau}{n})^k (1-\frac{\tau}{n})^{n-k}$$

Note that ensuring a sufficiently low F requires fixing h, τ and n; thus, the number of online players n must be known to securely parameterize Proof-of-Stake + BA. We refer to [GHM+17] (Appendix B.2) for a detailed analysis of honest committee election in the setting of Proof-of-stake.

3.4 Input fairness in the permissionless setting

Since permissionless consensus protocols introduced in this section are all leader-based, we refer the reader to Section 3.2 for an initial description of the (adversarial) leader powers and approaches towards realizing input fairness in the *authenticated* setting.

Several challenges arise when attempting to establish notions of input fairness in the *permissionless* setting where parties are unauthenticated and future role-assignment of parties is unknown (player-replaceability). We highlight the challenges and our contribution FairPoS in P6.

Encrypt-and-reveal with *threshold encryption*. Whilst threshold encryption committee can be deployed in the *authenticated* setting to realize encrypt-and-reveal without selective abort (§3.2), this becomes challenging in the permissionless setting; realizing threshold key generation and threshold key encryption schemes with *player-replaceability* remains an open research question; recent works [GHK+21, CDGK23] have established first theoretical foundations in this direction.

Encrypt-and-reveal with *timed-release cryptography*. A promising approach is to turn to *timed-release cryptography* to implement encrypt-and-reveal in a permissionless setting. First introduced in the form of *time-lock puzzles* [RSW96], timed-release cryptography permits *any* party to extract the encrypted message by performing a fixed number of operations, which resist parallelization. In [RSW96], the extraction of a message from a time-lock puzzle requires the extracting party to perform squarings of a RSA group element in a group of unknown order to obtain the decryption key. Timed commitments [BN00] ensure well-formedness of the extraction task generated by the sender. Timed-release cryptography promises *encrypt-and-reveal* in the permissionless setting since any party can extract the original message thereby denying the adversary any selective abort attacks plaguing commit-and-reveal schemes (See P5). Formal security of time-lock puzzles and timed commitments have been established for the sequential squaring assumption [KLX20] and in the UC model [BDD+21].

We highlight challenges in timed-release cryptography in practice. Fundamentally, the parameterisation of timed-release cryptography to match physical time as intended by the sender naturally assumes the existence and access to the **state-of-the-art extraction machine** that performs each sequential operation in the shortest time possible; in practice, investment in specialized ASIC hardware is costly, and does not preclude unknown or future computing advancements. Thus, relating the intended timed-release duration to physical time in practice remains challenging. Alternative frameworks to formalize this relation remains an open problem.

Nonetheless, if one is willing to accept the uncertainty in delay parameterisation, the state-of-the-art in timed-release cryptography is *Delay Encryption*, first proposed by by De Feo et al. [BF21]; here, clients to encrypt to a single session key in each round. The extraction process only needs to be performed once for each round, thereby revealing the decryption key for all encrypted round inputs. In contrast, naive usage of time-lock puzzles implies that participants must perform an extraction process for *each input* submitted to the permissionless consensus protocol execution. The price paid for this improvement in usability is complexity in cryptographic assumptions; delay encryption and its related verifiable-delay-function (VDF) predecessor [DFMPS19] are based on supersingular elliptic curves isogenenies, which are very memory intensive in practice.

Encrypt-and-reveal in permissionless consensus. In this thesis, we investigate the realization of input fairness in form of *encrypt-and-reveal* with delay encryption in permissionless consensus. In order to relieve clients from performing the costly extraction of delay encrypted inputs, we only require parties participating in *block leader election* to perform this task, and to make extraction keys public by including these in later blocks generated after encrypted inputs are finalized.

- *Nakamoto consensus.* In FairPoS P6, we realize a Nakamoto Proof-of-Stake protocol with input fairness from encrypt-and-reveal with delay encryption. A key challenge in this work is to overcome the network adversary's ability to "delay" the initialization of honest key extraction processes by deferring the broadcast of adversarial blocks containing delay encrypted inputs; in contrast to standard Nakamoto consensus where elected honest leaders can choose to extend any chain branch in their local view, honest leaders extracting decryption keys from prior blocks need to have seen these "on-time", so the decryption keys can be extracted by the designated slot. FairPoS introduces a new protocol rule for Nakamoto PoS called the "longest *extendable* chain selection" rule, which mitigates such synchronization attacks on honest delay extraction processes and restores the honest leader's ability to extend other honest blocks, thereby maintaining liveness and consistency properties despite the presence of a "stronger adversary".

We conjecture FairPoS can be adapted to the proof-of-work setting and consider it an interesting extension for future work.

- *Proof-of-stake + Byzantine Agreement.* We briefly discuss applying encrypt-and-reveal with delay encryption to the proof-of-stake & BA consensus introduced by Algorand [GHM+17]. Here, the same principle from FairPoS can be applied, tasking parties participating in leader and committee election to post decryption keys after completing extraction, but since blocks are finalized by a BA committee in each round, no block withholding attacks can be performed, simplifying analysis.

  In contrast to Nakamoto PoS (and by extension FairPoS), however, note that PoS & BA inherently requires a very high number parties to participate in leader election to guarantee honest BA committees in each round (§3.3.5); applying delay encryption to such protocols would imply that all online participants are required to perform the expensive extraction of decryption keys, even if the probability of getting elected is very low. This dramatically increases the cost of participation, given the relatively rare event of obtaining block rewards in the PoS & BA setting.

Thus, we argue that implementation of encrypt-and-release in the permissionless setting is best motivated in Nakamoto Proof-of-Stake, where only an honest majority of online parties must be assumed to perform the expensive key extraction.

Receipt-order-fairness in permissionless consensus. We note that receipt-order-fairness introduced in [KZGJ20] was later extended to the permissionless setting in [KDK22]. We emphasize that the notion of receipt-order-fairness implies a non-adversarial network between clients submitting inputs and elected committee members performing consensus, thereby excluding any adversarially induced receipt delays on inputs sent to nodes participating in consensus. In the spirit of the permissionless setting, we argue that this is an overly optimistic assumption.

4 L2: Decentralized Finance (DeFi)

This chapter provides background and motivation for the following research questions[1];

RQ 1: What are formal semantics & desired safety properties of DeFi?

RQ 2: Can lending protocols be securely parameterized with automated formal verification?

RQ 3: What is the optimal front-running attack on AMM's?

RQ 4: How can front-running in DeFi be mitigated?

Whilst Bitcoin [Nak08] introduced a limited scripting language to implement simple "spending conditions" for individual coins, it was Ethereum [But13] that first introduced support for general-purpose programs on the Ethereum Virtual Machine (EVM). The EVM is an expressive stack-machine and several high-level object-oriented programming languages have implemented which compile to native EVM bytecode.

Such user-friendly **smart contract programming languages** opened the floodgates for an innovative ecosystem of interoperable financial applications. In the EVM, each smart contract exposes user-defined interfaces that support message-passing, allowing **composition of smart contracts**; to facilitate contract composition across different applications, community-proposed smart contract interface standards have emerged from the Ethereum Request for Comments (ERC) forums; notable examples include standardized fungible token (ERC20) and non-fungible token (ERC721) interfaces.

The ecosystem built on standardized token interfaces that emerged in recent years is called **Decentralized Finance**, or **DeFi**. We dedicate this chapter to providing background for the DeFi domain, relevant to RQ 1-4 addressed by this thesis; DeFi applications feature similar functionalities as their traditional finance counterparts, but generally achieve such goals with very different mechanisms - we argue how these arise from the inherent constraints of building applications on permissionless consensus in §4.1.

We outline key design features of automatic market makers (§4.2.2) and lending protocols (§4.2.3) and detail how these are specifically designed to realize token exchanges and lending functionalities in the permissionless setting (RQ 1,2). Here, we contribute the earliest **formal modals** of automatic market makers (P1a/P1b) and lending protocols (P2); such executable specifications of DeFi permit the formal investigation of structural and incentive properties (e.g. arbitrage and liquidation safety) across different implementations. These models are amenable to automated formal verification; in P3, we explore optimal parameterizations for lending protocols, to ensure liquidation safety. Here, our formal lending protocol model from P2 is implemented in a **statistical model-checking** environment, allowing the modelling of stochastic price behaviour and its affect on liquidations in lending protocols.

Another constraint on DeFi applications is the general lack of input fairness (§4.3.1), leading to rampant front-running in DeFi (RQ 3,4); here, we present how such attacks represent a general tax on the utility on consensus platforms for all users. We contribute the first optimal front-running attack on constant-product automatic market makers in P4, a **multi-layer sandwich** attack on *all* AMM action types; surprisingly, our results indicate that the *redeem* action type submitted by honest liquidity providers are omitted from the optimal, adversarial strategy, as it does not contribute

[1]See Chapter 2 for an overview of research questions addressed by this thesis

to the adversarial profit. This illustrates the challenge of incentive composition; the block leader participating in layer 1 consensus is incentivized by layer 2 applications to order transactions in an manner unintended by protocol designers.

A key feature of DeFi is atomic composition, illustrated in §4.2.4; this enables risk-free strategies and flashloans, which are not possible in traditional finance, where settlement across market venues and financial institutions is highly asynchronous. Finally, we also discuss open questions on long-running notions of fairness due to a general lack of pre- and post-trade privacy in DeFi (§4.3.2); this challenge is addressed in our contribution P9 in this thesis and motivated in §5.3.

4.1 Constraints on L2 applications

We highlight selected constraints specific to the setting of permissionless consensus;

Limited scalability. Permissionless consensus is necessarily synchronous (§3.3) and parameterized by the worst-case network latency. This implies a bound on block sizes and frequency of block arrival, limiting the throughput of such systems. Such constraints necessarily effect the design of the smart contract virtual machine. In the EVM, smart contract execution is bounded by EVM Gas semantics, which limits computation and state access in each transaction and block. Thus, financial applications adapted to the DeFi environment have emerged with efficient designs to satisfy compute and storage constraints. In §4.2.2, we highlight how automatic market makers efficiently realize token exchanges with constant compute and storage overhead.

A secondary consequence of limited throughput is the **limited composability** of DeFi applications. An update in a decentralized exchange updating the asset price may, ideally, trigger liquidation of collateral present in other DeFi protocols; however, the cost of updating the state of both decentralized exchange and all collateral positions in dependent contracts is potentially unbounded. In practice, price updates and liquidations in lending protocols are de-coupled (see §4.2.3). Agents must be *incentivized* to explicitly perform *individual* liquidations with dedicated transactions submitted to the platform.

Incentivization. Public applications in Decentralized Finance are updated by signed transactions by pseudonymous participants. Ordering of transactions is determined by anonymous servers running leader-based consensus. Thus, both clients and block-generating servers must be incentivized to act according to protocol: Automatic market makers (§4.2.2) rely on *arbitrageurs* to align the exchange rate with the fair market rate. Lending protocols (§4.2.3) require *liquidators* to exchange lender collateral to recover outstanding loans. Maldesigned or malcomposed incentives may cause arbitrage and liquidations to fail or block-producers to censor, re-order and inject transactions to their benefit (Section 4.3.1).

Lack of fairness. Interaction between clients and smart contracts is public. The lack of input fairness (§3.4) incentivizes the block leader to inject its own transactions and generate an ordering to front-run or sandwich honest transactions (RQ 3,4 and §4.3). We sketch such attacks in §4.3.1 and how it reduces the utility of platforms for all users.

4.2 Application archetypes

4.2.1 TradFi: Limit order books

We illustrate a traditional limit order book to motivate the design of automated market makers intended for efficient execution in the setting of permissionless consensus protocols.

The task of trading financial assets is traditionally accomplished with limit order books (LOB). Let such an application support the buying or selling of specific asset at a price denominated in a standard numeraire currency; traders will submit limit orders, containing a buy or sell bit $d \in \{\mathsf{bid}, \mathsf{sell}\}$, a price limit p, below ($d = \mathsf{bid}$) or above ($d = \mathsf{sell}$) which they are willing to execute the trade, and the volume v or amount of the asset they wish to trade. A trade_i authorized by P_i is the tuple (d, p, v). We sketch the complexity overhead of performing a single trade at a venue with a limit order book trade execution engine.

The limit order book maintains lists bids and sells containing unmatched trade orders sorted by price limit; in the bids list, the bid order with the highest price limit is listed first, whereas in the sells list, the sell order with the lowest price limit is located at the top. The difference in price limit between the top of bids and sells lists is called the *spread*. Let the number of orders in bids and sells be m and n respectively.

A new trade order entering the limit order book must be matched against the sorted list of trades in the opposing direction. A (bid, p, v) order, for example, is matched against elements in sells, until none of the volume v remains; in the worst case, the order volume may match against all orders in sells, requiring $\mathcal{O}(n)$ matching operations, and still have remaining volume v'; in this case, (bid, p, v') must be inserted into bids, requiring $\mathcal{O}(m)$ complexity for insertion into a sorted list. Thus, the worst-case complexity of performing a trade in a limit order book is $\mathcal{O}(m + n)$.

In the next section, we will see how trades can be performed with $\mathcal{O}(1)$ complexity in DeFi applications called automated market makers .

4.2.2 DeFi: Automatic market makers

Automatic market makers (AMM) processes a trade with $\mathcal{O}(1)$ complexity by matching each trade order with all the token reserves available in the AMM application.

Consider an AMM offering trades between asset pair (τ_0, τ_1), and holding token reserves $(r_0 : \tau_0, r_1 : \tau_1)$. Here, let $\mathsf{swap}(v_0 : \tau_0, v_1 : \tau_1)$ denote a user order authorizing to trade v_0 units of τ_0 for a minimum of v_1 units of τ_1 in return. The AMM with reserve state $(r_0 : \tau_0, r_1 : \tau_1)$ will always process trade $\mathsf{swap}(v_0 : \tau_0, v_1 : \tau_1)$ if it satisfies the implemented AMM "strategy", which is to maintain an invariant over its reserves $(r_0 : \tau_0, r_1 : \tau_1)$. Ignoring trading fees for brevity, any trade updating AMM reserves from $(r_0 : \tau_0, r_1 : \tau_1)$ to $(r_0 + v_0 : \tau_0, r_1 - v_0 : \tau_1)$, must satisfy

$$I(r_0, r_1) = I(r_0 + v_0, r_1 - v_1) \tag{4.1}$$

for a constant function or swap invariant $I : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$. Thus, AMM's are also referred to as **constant function market makers** (CFMM) in the literature. A commonly deployed swap invariant is the constant product function $I_\times(r_0, r_1) = r_0 \cdot r_1$. Concretely, executing a trade $\mathsf{swap}(v_0 : \tau_0, v_1 : \tau_1)$ means computing v_1' in

$$r_0 \cdot r_1 = (r_0 + v_0) \cdot (r_1 - v_1') \tag{4.2}$$

and asserting that $v_1' \geq v_1$, where v_1 is the minimum amount of τ_1 the user is willing to accept for the trade; the swap order is invalid if this assertion fails. Thus, any trade order is evaluated in constant time and storage with only a few arithmetic operations. This represents a significant improvement in execution cost when compared to limit order books with large lists of buy and sell orders, making AMM's suitable for deployment as smart contracts.

Constant functions. The swap invariant or constant function implemented in AMM's is generally *convex* and *differentiable* (smooth); this is satisfied by commonly implemented constant production function $I_{\times}(r_0, r_1) = r_0 \cdot r_1$. Given a submitted trade $\mathsf{swap}(v_0 : \tau_0, v_1 : \tau_1)$ executed on AMM state $(r_0 : \tau_0, r_1 : \tau_1)$, the amount of $v_1' : \tau_1$ returned to the trader is determined by Eq. (4.2);

$$v_1' = r_1 - \frac{r_0 \cdot r_1}{(r_0 + v_0)} = \frac{r_1 \cdot v_0}{r_0 + v_0} \iff \frac{v_1'}{v_0} = \frac{r_1}{r_0 + v_0} \tag{4.3}$$

We highlight two observations. (1) For $v_0 \to 0$, the exchange rate approaches $v_1'/v_0 = r_1/r_0$; this is called the **marginal exchange rate**. (2) For increasing v_0, the amount v_1' returned to the trader decreases. Thus, the automated **market making strategy** of an AMM at state $(r_0 : \tau_0, r_1 : \tau_1)$ implied by (1) and (2) is to offer an initial, marginal exchange rate of r_1/r_0, and to *decrease* the exchange rate with *increasing* trade volume in favor of the AMM. Ideally, the marginal exchange rate of the AMM should be fair, reflecting the wider market prices for traded assets; otherwise, the AMM would be offering an exchange at an unprofitable rate for itself, or conversely, it would represent an unattractive market venue for traders.

Arbitrage. The marginal exchange rate of an AMM aligns with external, fair asset prices by means of interaction with rational arbitrageurs. As we formalize in P1a/P1b, the arbitrageur performs a trade such that the marginal exchange rate is consistent with that of the fair market exchange rate; informally, the optimal arbitrage strategy is found by increasing the trade volume, which causes the exchange rate to deteriorate (Eq. (4.3)). The optimal trade volume is found when additional increase in trade volume has a negative impact on arbitrage profit.

Note that financial trades are inherently **zero-sum**. A profit by a rational arbitrageur implies a loss for the AMM; specifically, it implies a loss for liquidity providers who have provided token reserves that are matched against trades. This is called **impermanent loss** and is incurred by liquidity providers who deposit funds in an AMM, which then experiences hanges in fair asset exchange rates, resulting in arbitrageurs re-aligning the composition of tokens.

Liquidity providers. A party which deposits funds to an AMM is called a liquidity provider and agrees to the automated market making strategy implied by the constant function implemented by the AMM. Thus far, we have omitted trading fees for simplicity, but in our formal AMM model (P1a/P1b), the rational liquidity provider anticipates that the fees from trading will exceed any effects from impermanent loss; note that impermanent loss can be avoided by liquidity providers by *actively* repositioning their funds, avoiding impermanent loss induced by rebalancing via arbitrage. The AMM implementation Uniswap V3 [AZS+21] permits liquidity providers to dedicate liquidity to chosen *price ranges*, thereby mitigating impermanent loss but not eliminating it; it represents a more expressive interface for liquidity providers and has been shown to be equivalent to limit order books in this regard [MMR23]. Still, it can be observed empirically that most liquidity providers in Uniswap V3 remain *unsuccessful* in rebalancing AMM funds and suffer impermanent loss nonetheless [HSW22].

We refer to P1a/P1b for a full formalization and analysis of AMM's and its interfaces exposed to traders and liquidity providers. We highlight a concurrent line of work that formalizes AMM states as sets of permissible trades [AECB22, AAE+22, DRCA23]; the proposed "valid trade set" abstraction conveniently allows for compositional modelling of AMM's, since it avoids the need for an analytical treatment of each specific AMM constant function implementation.

4.2.3 DeFi: Lending protocols

Lending protocols (or “lending pools” in P2) intermediate between borrowers and lenders in the DeFi environment, athough the dilineation between the two roles is not strict. Since participants are pseudonymous and carry no identity or reputation, any loan must be *collateralized* in order to protect against failure of repayment. A party depositing collateral is both borrower and lender; the collateral itself is made available for borrowing in other loans, avoiding locked, “unproductive capital”. This may appear to undermine the effectiveness of collateralization, but sufficient **overcollateralization** in real-world lending protocols have shown impressive real-world resilience in recent years [GWPK20a]. We provide an intuition of the formal semantics of the lending protocol model introduced in P2 with the following example execution.

Consider an initial state with two parties **A** and **B**, holding balances of 125 units of τ_0 and 100 units of τ_1 respectively. For simplicity, we assume the prices of both assets to be equal $P(\tau_0) = P(\tau_1)$.

$$\mathsf{A}\,[\,125 : \tau_0\,] \mid \mathsf{B}\,[\,100 : \tau_1\,]$$

Deposit. Both parties deposit funds to initialize a lending protocol and authorize $\mathsf{A} : \mathsf{dep}(100 : \tau_0)$ and $\mathsf{B} : \mathsf{dep}(100 : \tau_1)$ respectively; each deposit of τ_0 and τ_1 also results in the *minting* of **liquidity tokens** (“minted tokens” in P2), namely $\{\tau_0\}$ and $\{\tau_1\}$, which are returned to the depositors; they can be redeemed to withdraw the deposited tokens with additional interest, if the underlying funds are loaned out to borrowers. The deposits initialize a lending protocol **LP** holding $100 : \tau_0$ and $100 : \tau_1$.

$$\mathsf{A}\,[\,25 : \tau_0, \mathbf{100} : \{\tau_0\}\,] \mid \mathsf{B}\,[\,\mathbf{100} : \{\tau_1\}\,]$$
$$\mathsf{LP} : (\mathbf{100} : \tau_0) \mid (\mathbf{100} : \tau_1)$$

We note that liquidity tokens $\{\tau_0\}$ and $\{\tau_1\}$ are transferable if the transfer does not result in the sender becoming undercollateralized (See liquidation below).

Borrow. Having initialized the lending protocol, party **B** authorizes a borrow with $\mathsf{B} : \mathsf{bor}(50 : \tau_0)$, receiving 50 units of τ_1, previously deposited to the lending protocol by party **A**; the loan extended to **B** is recorded in the state of the lending protocol. Importantly, borrower **B** is able to receive the loan of 50 units of τ_0 because it holds liquidity shares consisting of 100 units of $\{\tau_1\}$ that serve as **collateral**; we assume a minimal collateralization factor of 2, which is the ratio of the collateral value $(100 : \{\tau_1\})$ over the loan value $(50 : \tau_0)$. Here, the collateral value of liquidity tokens is determined by the value of the tokens it can be redeemed for (see redeem action below).

$$\mathsf{A}\,[\,25 : \tau_0, 100 : \{\tau_0\}\,] \mid \mathsf{B}\,[\,\mathbf{50} : \tau_0, 100 : \{\tau_1\}\,]$$
$$\mathsf{LP} : (\mathbf{50} : \tau_0, \{\mathsf{B} : \mathbf{50}\}) \mid (100 : \tau_1)$$

Since the collateralization factor of **B** is exactly 2, **B**’s entire balance of liquidity shares is now non-transferable; otherwise, this would result in undercollateralization.

We now show that roles of borrowers and lenders are not clearly delineated; party **A** can also authorize $\mathsf{A} : \mathsf{bor}(25 : \tau_1)$, thereby utilizing its liquidity token balance of $100 : \{\tau_1\}$ as collateral. Thus, both **A** and **B** are both lender and borrower in the resulting state.

$$\mathsf{A}\,[\,25 : \tau_0, \mathbf{25} : \tau_1, 100 : \{\tau_0\}\,] \mid \mathsf{B}\,[\,50 : \tau_0, 100 : \{\tau_1\}\,]$$
$$\mathsf{LP} : (50 : \tau_0, \{\mathsf{B} : 50\}) \mid (\mathbf{75} : \tau_1, \{\mathsf{A} : \mathbf{25}\})$$

Interest accrual. Over time or with increasing block height, interest accrues in all loans recorded in the lending protocol. In our example, we apply a 20% interest to all outstanding loans taken by both **A** and **B**. Recall, however, that the minimal collateralization of 2 was already previously reached by **B**; given the increase in loan amount, this interest accrual will result in **B** becoming **undercollateralized**.

$$\mathsf{A}\,[\,25:\tau_0, 25:\tau_1, 100:\{\tau_0\}\,] \mid \mathsf{B}\,[\,50:\tau_0, 100:\{\tau_1\}\,]$$
$$\mathsf{LP}:(50:\tau_0,\{\mathsf{B}:\mathbf{60}\}) \mid (75:\tau_1,\{\mathsf{A}:\mathbf{30}\})$$

Note that smart contracts deployed to the EVM cannot automatically activate with each block height increase. Thus, any interest accrual is computed lazily in implementations; the update of the loan balances with interest is deferred to the next interaction between any party and the lending pool.

Liquidate. A borrower can become undercollateralized if interest continues to accrue without repayment or if token prices fluctuate: in such a state, the undercollateralized borrower can be liquidated by any party. In our example execution, **A** observes that **B** is undercollateralized and authorizes $\mathsf{A}:\mathsf{liq}(\mathsf{B}, 25:\tau_0, 30:\{\tau_1\})$, which *repays* $25:\tau_0$ of **B**'s outstanding loan. In return, **A** obtains collateral from **B**, namely $30:\{\tau_1\}$ which is directly deducted from **B**'s balance.

$$\mathsf{A}\,[\,25:\tau_1, 100:\{\tau_0\}, \mathbf{30}:\{\tau_1\}\,] \mid \mathsf{B}\,[\,50:\tau_0, \mathbf{70}:\{\tau_1\}\,]$$
$$\mathsf{LP}:(\mathbf{75}:\tau_0,\{\mathsf{B}:\mathbf{35}\}) \mid (75:\tau_1,\{\mathsf{A}:30\})$$

Note that the liquidating party obtains a discount on the collateral. This is to ensure that liquidations are performed in a timely manner by incentivizing rational parties.

Redeem. Finally, we illustrate how liquidity tokens can be redeemed for the underlying tokens, thereby extracting a profit accrued over time from interest. Here, party **A** authorizes $\mathsf{A}:\mathsf{rdm}(30:\{\tau_1\})$, thereby returning $30:\{\tau_1\}$ for $33:\tau_1$ in return. Since **A** originally deposited $1:\tau_1$ for each liquidity share $1:\{\tau_1\}$, the redeem action realizes a profit.

$$\mathsf{A}\,[\,\mathbf{58}:\tau_1, 100:\{\tau_0\}\,] \mid \mathsf{B}\,[\,50:\tau_0, 70:\{\tau_1\}\,]$$
$$\mathsf{LP}:(75:\tau_0,\{\mathsf{B}:35\}) \mid (\mathbf{42}:\tau_1,\{\mathsf{A}:30\})$$

We note that the actual interest accrual in lending protocol implementations is a function of **liquidity utilization**; namely, the ratio of deposited funds which have been loaned away. If the utilization ratio is too high, there are no funds that can be redeemed, potentially weakening the incentivization of liquidating agents, as the collateral may then become unredeemable. If the utilization ratio is too low, no interest can accrue and parties have no incentivize to deposit funds into the lending protocol. Thus, the interest function is designed to equilibriate between these two extremes by adjusting interest rates to incentivize more deposits (higher interest rate) or more borrows (lower interest rate). We refer to an overview of interest rate functions deployed by implementations in [GWPK20b] and to our formal model in P2 for full lending protocol semantics and their security properties.

Liquidation safety. The security of loans extended to borrowers in lending protocols is contingent on the possibility of recovering the loan in case the borrower's collateralization falls below the minimum threshold. Whilst the accrual of interest, applied in lock-step with chain growth, is predictable and gradual, the prices of loaned and collateralized token assets can be volatile, resulting in sudden undercollateralization of borrowing parties; we contribute a formal analysis to determine secure parameterizations of lending protocols by means of a stochastic model checking framework in P3, which allows the exploration of liquidation security for different price volatility regimes and liquidator behaviour. Further, we refer to [PWXL21] for an empircal study on the security of liquidation in real-world lending protocols.

On-chain prices & incentive composition. Lending protocols rely on price oracles to securely compute collateralization ratios of borrowing parties. In practice, third parties (price oracles) are trusted to accurately post token prices to smart contracts which are then read by lending protocol implementations during execution. In the spirit of the permissionless setting, it would seem preferable to rely on prices implied by the marginal exchange rate of automatic market makers deployed to the (on-chain) DeFi environment, but assessing fair token prices directly on AMM's in decentralized finance remains an open challenge. In particular, the **liquidation incentive** in lending protocols motivates manipulations of AMM oracles in DeFi by means of large trades which perturb the price; this price impact can then unlock liquidations exploited by the adversary for profit. Here, the **arbitrage incentive** alone does not suffice to keep the AMM price aligned with global, fair token prices when composed with the liquidation incentive of lending protocols.

This highlights an open challenge in DeFi to achieve secure composition of incentives, which in isolation motivate intended behaviour, but in composition with external protocol incentives lead to undesirable behaviour. In addition to composing poorly with AMM price oracles, the **interest incentive** from lending protocols on deposited tokens also competes directly with the underlying proof-of-stake consensus protocol, which rewards parties to stake coins and secure consensus with **staking incentives**; here, the rational player will express a preference for depositing in the lending protocol if the interest exceeds the staking reward, potentially weakening the security of the underlying proof-of-stake consensus protocol. This phenomena was investigated in [Chi21] with agent based simulations; a formal understanding of incentive composition remains an open problem of great importance.

4.2.4 Atomic composition & Flash loans

In contrast to traditional finance, interactions with multiple venues in a DeFi ecosystem can be performed within a single blockchain transaction, implying immediate or atomic settlement. More generally, a signed transaction txId can authorize messages to multiple smart contract interfaces; consider a txId that authorizes a sequence of calls to multiple contracts $\mathsf{c}_1, ..., \mathsf{c}_n$. Then, the txId either succeeds to update the state from Γ to Γ' by executing $\mathsf{action}_1, ..., \mathsf{action}_n$ or reverts entirely if even a single of the actions does not execute successfully.

$$\Gamma \xrightarrow{\mathsf{txId}} \Gamma' \qquad \Gamma \xrightarrow{\mathsf{txId}:\mathsf{c}_1:\mathsf{action}_1} \cdots \xrightarrow{\mathsf{txId}:\mathsf{c}_n:\mathsf{action}_n} \Gamma'$$

Risk free strategies In practice, atomic composition of actions in the EVM environment is performed by implementing a strategy in a user-deployed smart contract; let $\mathsf{c}_{\mathsf{usr}}$ denote such a contract instance. When the user issues a transaction authorizing a call to $\mathsf{c}_{\mathsf{usr}} : \mathsf{exec}$, the evaluation thereof can issue calls to external, DeFi contracts $\mathsf{c}_1, \mathsf{c}_2, ...$ thereby executing the intended financial strategy:

$$\Gamma \xrightarrow{\mathsf{txId\ calls\ c_{usr}:exec}(\mathsf{c_{usr}}, v:\tau)} \xrightarrow{\mathsf{c_{usr}\ calls\ c_1:action_1}} \xrightarrow{\mathsf{c_{usr}\ calls\ c_2:action_2}} \xrightarrow{\cdots} \xrightarrow{\mathsf{c_{usr}\ asserts\ (c_{usr}.bal > c_{usr}.bal}(\Gamma))} \Gamma'$$

Importantly, such a strategy can be implemented in risk-free fashion for the user, by ensuring that the implementation of $\mathsf{c}_{\mathsf{usr}} : \mathsf{exec}$ concludes with an assertion that the balance of $\mathsf{c}_{\mathsf{usr}}$ *increases* following execution. If this assertion fails, the entire transaction txId is reverted and the state is reset to Γ.

Flashloans Since atomic composition permits risk free strategies in the DeFi environment, lending funds to a party performing such strategy can also be realized in a risk free manner.

In the following execution sketch, a transaction txId authorizes a call to a flash loan contract $\mathsf{c}_{\mathsf{flashLn}}$, indicating the requested loan amount and a pointer to the user-deployed contract $\mathsf{c}_{\mathsf{usr}}$ implementing the user's financial strategy to deploy the borrowed tokens. The implementation of $\mathsf{c}_{\mathsf{flashLn}} : \mathsf{borr}$ then transfers $v : \tau$ to $\mathsf{c_{usr}}$ in a forwarded message to $\mathsf{c_{usr}} : \mathsf{exec}$, permitting the user strategy to proceed with the borrowed funds.

$$\Gamma \xrightarrow{\mathsf{txId\ calls\ c_{flashLn}{:}borr}(v{:}\tau,\mathsf{c_{usr}})} \xrightarrow{\mathsf{c_{flashLn}\ calls\ c_{user}{:}exec}} \xrightarrow{\mathsf{c_{flashLn}\ asserts}\ (\mathsf{c_{flashLn}.bal} > \mathsf{c_{flashLn}.bal}(\Gamma))} \Gamma'$$

Importantly, once the function body implementing $\mathsf{c_{usr}} : \mathsf{exec}$ completes, the program evaluation then returns to the remainder of $\mathsf{c_{flashLn}} : \mathsf{borr}$, which finally asserts that the final contract balance exceeds the initial balance at state Γ, thereby insuring repayment of the flashloan.

4.3 Notions of fairness in DeFi

4.3.1 Transaction ordering

As highlighted in §3.4, different notions of input fairness have been proposed in the permissionless consensus setting; in practice, popular blockchain platforms such as Ethereum guarantee no input fairness out-of-the-box; as a leader-based consensus protocol, the elected round leader can choose to **order**, **censor** and **inject** transactions in each block. Given a set of pending transactions received over the P2P gossip network, each block leader is incentivized to extract the maximum value from this set of transactions with each block; this has been named **Miner Extractable Value (MEV)** and formal definitions have been proposed in Clockwork finance [BDKJ21] and by Bartoletti et al. [BZ23].

Optimal MEV from AMM's. We sketch how miner extractable value can be extracted from AMM transactions by means of **sandwich attacks**. To illustrate such an attack, we first consider an honest trace of trades performed by honest user **A** on an AMM in state $(100 : \tau_0, 100 : \tau_1)$. Here, we assume $P(\tau_0) = P(\tau_1)$. In our initial, honest example, **A** authorizes the same swap twice, namely two $\mathsf{swap}(15 : \tau_0, 10 : \tau_1)$ actions, which each trade $15 : \tau_0$ for a minimum of $10 : \tau_1$. Subsequently, **A** restores the state of the AMM to its initial state by authorizing a swap in the opposing direction $\mathsf{swap}(23 : \tau_1, 30 : \tau_0)$; since trading fees are omitted, **A** also restores its own initial balance after this third and final trade. The execution is shown below and satisfies the constant product function in Eq. (4.2).

$$\begin{aligned}
& && (100 : \tau_0, 100 : \tau_1) \mid \mathbf{A}[30 : \tau_0] \\
& \xrightarrow{\mathbf{A}{:}\mathsf{swap}(15{:}\tau_0,10{:}\tau_1)} && (115 : \tau_0, 87 : \tau_1) \mid \mathbf{A}[15 : \tau_0, 13 : \tau_1] \\
& \xrightarrow{\mathbf{A}{:}\mathsf{swap}(15{:}\tau_0,10{:}\tau_1)} && (130 : \tau_0, 77 : \tau_1) \mid \mathbf{A}[0 : \tau_0, 23 : \tau_1] \\
& \xrightarrow{\mathbf{A}{:}\mathsf{swap}(23{:}\tau_1,30{:}\tau_0)} && (100 : \tau_0, 100 : \tau_1) \mid \mathbf{A}[30 : \tau_0]
\end{aligned}$$

We highlight two key observations: (1) firstly, **A** receives a lesser exchange rate with the second trade, even though the swap parameters of the first two trades are identical and (2) secondly, given that prices of both token types are equal, the final trade by **A** is necessarily profitable.

We modify our initial, honest execution to obtain an optimal front-running **sandwich attack** performed by malicious **M** on **A**'s authorized trade $\mathsf{swap}(23 : \tau_0, 30 : \tau_1)$; this sandwich attack is obtained by simply allowing **M** to perform the first and final action from the previous honest, execution. By observing the balance of **M** resulting from the attack and given $P(\tau_0) = P(\tau_1)$, it is apparent that **M** achieves a profit of 5.

$$\begin{aligned}
& (100 : \tau_0, 100 : \tau_1) \mid \mathbf{A}[30 : \tau_0] \mid \mathbf{M}[15 : \tau_0, 10 : \tau_1] \\
\xrightarrow{\mathbf{M}:\mathsf{swap}(15:\tau_0,10:\tau_1)} \; & (115 : \tau_0, 87 : \tau_1) \mid \mathbf{A}[30 : \tau_0] \mid \mathbf{M}[0 : \tau_0, 23 : \tau_1] \\
\xrightarrow{\mathbf{A}:\mathsf{swap}(15:\tau_0,10:\tau_1)} \; & (130 : \tau_0, 77 : \tau_1) \mid \mathbf{A}[15 : \tau_0, 10 : \tau_1] \mid \mathbf{M}[0 : \tau_0, 23 : \tau_1] \\
\xrightarrow{\mathbf{M}:\mathsf{swap}(23:\tau_1,30:\tau_0)} \; & (100 : \tau_0, 100 : \tau_1) \mid \mathbf{A}[15 : \tau_0, 10 : \tau_1] \mid \mathbf{M}[30 : \tau_0, 0 : \tau_1]
\end{aligned}$$

We argue the optimality of the attack; (1) honest **A** receives the minimum amount of $10 : \tau_1$ permitted by its authorized trade limit. (2) the marginal exchange rate of the AMM in the final state is aligned with the asset prices of τ_0 and τ_1, since $P(\tau_1) = P(\tau_0)$ and the marginal exchange rate is 1; thus no additional arbitrage can be extracted by **M**. Thus, the attack is optimal as there is no additional value for the attacker to extract.

We contribute the first formalization of the optimal MEV attack for constant-product AMM's in P4; here, we intrduce an optimal MEV strategy named the **Dagwood sandwich** which considers all AMM action types, namely **swap**, **deposit** and **redeem**. Surprisingly, the optimal attack never includes honest redeem actions, as it does not contribute to the attackers profit. This may imply difficulty to redeem liquidity from AMM's in times of high blockchain fees or congestion.

MEV penalizes utility. Consider the sandwich attack illustrated previously; in absence of the front-running adversary, let there be the honest user **A** that is intending to submit a single swap transaction to be finalized in the blockchain; here, the demand for block-space is for one transaction only. With the introduction of a front-running adversary **M**, the demand for block-space now increases to three transactions; namely, the adversarial front-run and back-run transaction in addition to the original honest transaction. The increased demand induced by the promise of front-running profits naturally increases the market fees for transactions finalized on the blockchain, thereby reducing the utility for all users, not just **A** who experiences a worsened exchange rate on its trade due to the sandwich attack.

Input fairness mitigates MEV. We survey proposals to prevent front-running in P5 and refer to our discussion of input fairness notions realizable in the permissionless (§3.4) and authenticated (§3.2) settings. We contribute FairPoS P6 which formally guarantees the *encrypt-and-reveal* notion of input fairness in permissionless consensus.

4.3.2 Pre- and Post-trade privacy

While input fairness may prevent front-running attacks formalized in P4, it does not guarantee post-trade privacy; the trade order is merely blinded temporarily prior to execution. However, there are trading strategies which require pre- and post-trade privacy; namely, both submitted trade orders and their execution remain private. Consider the time-weighted-average-price strategy [WRA21]; here, the trader breaks a larger trade volume into smaller trades, which are periodically executed to minimize price impact. In traditional finance, such strategies are executed at dark pool venues, where the venue operator is trusted to keep order flow and execution private. Realizing this in the DeFi setting requires a notion of privacy-preserving smart contracts currently available on protocols such as Ethereum.

This motivates a key line of investigation of this thesis; pre- and post-trade privacy requires **privacy-preserving smart contracts** where users can privately input parameters to the contract execution and preserve privacy of their individual private state (and balances). We introduce this model of privacy-preserving smart contracts in §5.2 and propose the first practical realization thereof in **Eagle** (P7), where private smart contract execution is performed in an outsourced secure multi-party computation (MPC) setting; to achieve practical efficiency, all cryptographic overhead is moved outside the costly MPC evaluation, leaving only the smart contract logic to be evaluated inside the MPC protocol. Such a framework permits the implementation of dark pools in the decentralized finance setting; **dark pools** are private trading venues in traditional finance which ensure that trades can be executed privately, relying on a trusted venue operator to maintain the privacy of client trade orders and execution.

Furthermore, we observe that even with privacy-preserving smart contracts, the classical design of **dark pools leak privacy**, even when the dark pool venue operator is distributed with MPC. This motivates the design of the first differentially private market mechanism in our work P8 that also extends the classical notion of differential privacy to the trusted curator or MPC setting, where individual, private outputs may be correlated. We refer to §5.3 for a discussion of differential privacy in smart contract applications.

5 L2: Privacy in Cryptoeconomic Systems

This chapter provides background and motivation for the following research questions[1];

RQ 6: Can privacy-preserving contracts be realized in a practical manner?

RQ 7: Can differentially-private markets be realized in the setting of Eagle?

RQ 8: What problems can privacy-enhancing techniques solve in finance?

Standard smart contracts, as those supported by the EVM platform, feature no privacy by default. We have shown how this can lead to economic attacks in the §4.3.1 and P4; however, more generally many internet application requires some form of privacy to be meaningful, thereby motivating the investigation of meaningful **computation over private states**.

Informally, let the owner of a private state fragment in a smart contract be a user who has exclusive knowledge or ability to access its current private state in "cleartext". An owner cannot arbitrarily modify its own private state; any update must adhere to the smart contract logic agreed upon apriori. We argue that meaningful applications require the simultaneous update on private states of multiple users, whilst maintaining the privacy of all private states *following* their updates. Achieving this is goal non-trivial, motivating our work Eagle in P7 and our proposed differentially-private market mechanisms in P8.

We provide an overview of current proposals to realize privacy-preserving smart contracts in Fig. 5.1 and organize the proposals to realize privacy-preserving smart contracts in the following two categories.

In §5.1, a **single private states** can only be updated by its owner in each smart contract state update; this makes it challenging to apply contract logic over private states owned by separate users, limiting its expressiveness. This is partially overcome in the privacy-preserving UTXO model proposed by [BCG+20, XCZ+22, SCG+14].

In §5.2, the update of **multiple private states** is enabled with the introduction of the the contract manager model, achieving higher expressiveness than in §5.1. Here, we highlight our contribution Eagle (P7), the first practical and efficient privacy-preserving smart contract framework in this setting, which extends the contract manager model by enabling long-running private contract execution, where clients can conveniently submit private inputs and "receive" outputs whenever they come online in a lazy fashion. In contrast to prior proposals, Eagle achieves its efficiency by moving expensive cryptographic overhead outside the MPC evaluation, whilst achieving the security from private input and output sub-protocols.

Note that even with privacy-preserving smart contracts permitting private inputs and privacy-preserving state updates, private data from other users can trivially leak from the contract logic itself; we propose the very first notions of **differential privacy for smart contracts** to address this in §5.3 and refer to P8 for detailed definitions (§5.3.1), as well as our proposed "fuzzy order matching" market mechanisms (§5.3.2) that satisfy these.

[1]See Chapter 2 for an overview of research questions addressed by this thesis

	Zkay	**Zexe**	**Kachina**	**Hawk**	**zkHawk**	**Eagle** (P7)
State update	single private state (§5.1)			multiple private states (§5.2)		
Contract logic	public	private	public/(private)			
User input	private/public	single private input		multiple private inputs		
User interaction	concurrent	non-concurrent	concurrent	round-based		multi-round (& optional)
User funds	private balance	private balance and transfers				
Security	FHE	ZK	UC ZK	UC MPC & ZK		
Complexity	FHE evaluation	single prover ZK		zkSNARK in MPC	NIZK in MPC	only contract logic in MPC

Figure 5.1: Overview of privacy-perserving smart contract proposals.

5.1 Update of single private states

Homomorphic encryption. Proposals **Zkay** [SBG+19], ZeeStar [SBBV22] and SmartFHE [SA21] deploy homomorphic or fully homomorphic encryption (FHE) to permit smart contract computation on encrypted data. In FHE, the holder of the public key can encrypt information, while only the holder of the private key can decrypt; Given encrypted of data as well as the public key, anyone can perform computations on the encrypted data and evaluate algorithms on encrypted inputs. For example, given encryptions $[x]$, $[y]$, $[z]$ of the values x, y, z, FHE allows to compute the encryption $[x \cdot y + z]$ of $x \cdot y + z$. In this setting, we consider the owner of the FHE key pair as the owner of the encrypted private state stored on the blockchain.

Note that FHE computation can only be performed on data encrypted to the same public key. Thus, each smart contract computation only updates **individual private state** fragments in isolation; encrypted data of owner A cannot trivially be input to a computation on encrypted data of another owner B that is encrypted under a different FHE key, making smart contract computation over multiple, single-owner private states challenging.

Alternatively, proposals such as Penumbra [Pen23] require the FHE key pair to be generated by a distributed key generation (DKG) committee; here, anybody can provide private inputs by encrypting to the public key output by the DKG; computation is then performed over all encrypted inputs resulting in a **secret state fragment**. Here, the notion of a “single owner private state” can no longer be maintained; opening the private key material by the DKG committee following FHE computation reveals the entire secret smart contract state to all parties.

Shared private states. Towards meaningful smart contract computation over multiple, single-owner private states, the Unspent Transaction Output Model (UTXO) implemented by **Zexe** [BCG+20, XCZ+22] extends the privacy cryptocurrency protocol Zcash [SCG+14] with smart contract functionality. In the UTXO model, every state transition consumes previously “unspent” private state fragments in cryptographic commitment form, and generates new private state fragments. In the following example, we highlight that the privacy-preserving UTXO model leverages **shared private state** fragments to achieve meaningful interaction between private states owned by separate users.

Let $\langle\sigma_i\rangle$ denote a private state fragment owned by party $P_i \in \mathcal{P}$. Further, we assume an initial smart contract state $\langle\sigma_a\rangle \mid \langle\sigma_b\rangle$ where parties P_a and P_b are exclusive owners of their private states. Let the smart contract state transition function be denoted $\boldsymbol{F}$; in this setting, each state transition is performed over private states authorized by the owner of these private states.

In a first step, let party P_a authorize a state transition over the initial smart contract state. Thus, P_a authorizes the “consumption” of single-owner private state σ_a, and provides input x_a to the state transition $\boldsymbol{F}$, such that $\sigma_{ab} \leftarrow \boldsymbol{F}(x_a; \sigma_a)$. As a result, σ_a is removed from the smart contract state and a new private state fragment σ_{ab} is added to the updated contract state, shown below.

$$\langle\sigma_a\rangle \mid \langle\sigma_b\rangle \xrightarrow{P_a \text{ authorizes } \boldsymbol{F}(x_a;\sigma_a)} \langle\sigma_b\rangle \mid \langle\sigma_{ab}\rangle$$

Here, private state σ_{ab} is **shared** between parties P_a and P_b, meaning that this state is known and can be spent by both P_a and P_b. Next, let party P_b authorize a state transition on both a private state σ_b and the private state it shares with P_a, namely σ_{ab}. The evaluation of the state transition function over consumed state fragment σ_b and P_b's input x_b results in the creation of private state σ_{bc}, now shared between P_b and P_c: namely, $\sigma_{bc} \leftarrow \boldsymbol{F}(x_b; \sigma_b, \sigma_{ab})$.

$$\langle\sigma_b\rangle \mid \langle\sigma_{ab}\rangle \xrightarrow{P_b \text{ authorizes } \boldsymbol{F}(x_b;\sigma_b,\sigma_{ab})} \langle\sigma_{bc}\rangle$$

Finally, party P_c authorizes a state transition, consuming σ_{bc} and outputting the final single-owner state fragment σ_c.

$$\langle\sigma_{bc}\rangle \xrightarrow{P_c \text{ authorizes } \boldsymbol{F}(x_c;\sigma_{bc})} \langle\sigma_c\rangle$$

Thus, over the sequence of state transitions above, we have illustrated a smart contract update which is performed over single-owner private state fragments σ_a and σ_b and resulted in the creation of single-owner private state σ_c, thus enabling smart contracts over multiple, single-owner private state fragments; here, each state transition is authorized by a party on private state fragments it (jointly) owns. Inherent limitation in expressiveness remain, however; performing such a state update over multiple, single-owner private states requires multiple authorization steps and cannot be achieved in a single contract update (Section 5.2).

We briefly sketch how shared private state model is implemented and refer to [BCG+20, XCZ+22] for details. Private states are realized as *cryptographic commitments*, generated by authorizing clients in each state transition; these can be added to a *cryptographic accumulator* to break any associations between creation and destruction of commitments. Consuming commitments previously added to the accumulator requires a proof of membership and a public posting of a unique *serial number*, preventing the *re-use* or *double-spending* of commitments. A state transition generating a shared private state fragment requires that the authorizing client share the *commitment opening* with the joint-owner of the new state fragment; thus, this assumes secure and private communication between the two. Finally, an **authorization** is a valid non-interactive zero-knowledge proof, or **NIZK**, that the authorizing party has knowledge of the private state(s) being consumed, and that the newly created state commitment adheres to the transition logic $\boldsymbol{F}$ applied to consumed states and a client-chosen input.

Input concurrency. The authors of Kachina [KKK21] highlight a lack of input concurrency in the preceding shared private state model. Consider extending our shared private state model with public state fragment σ. Let an initial contract state with private and public state fragments be updated by the following two transitions governed by smart contract function $\boldsymbol{F}$;

$$\langle\sigma_a\rangle \mid \langle\sigma_b\rangle \mid \sigma \xrightarrow{P_a \text{ authorizes } \boldsymbol{F}(x_a;\sigma_a,\sigma)} \langle\sigma_b\rangle \mid \langle\sigma_{ab}\rangle \mid \sigma' \xrightarrow{P_b \text{ authorizes } \boldsymbol{F}(x_b;\sigma_b,\sigma')} \langle\sigma_{ab}\rangle \mid \langle\sigma_{bc}\rangle \mid \sigma''$$

The validity of the execution above implies the following;

(a) $(\sigma_{ab}, \sigma') \leftarrow \boldsymbol{F}(x_a; \sigma_a, \sigma)$

(b) $(\sigma_{bc}, \sigma'') \leftarrow \boldsymbol{F}(x_b; \sigma_b, \sigma')$

Informally, let us assume that the two state updates are *logically* concurrent; namely,

(c) $(\sigma_{bc}, \sigma') \leftarrow \boldsymbol{F}(x_b; \sigma_b, \sigma)$

(d) $(\sigma_{ab}, \sigma'') \leftarrow \boldsymbol{F}(x_a; \sigma_a, \sigma')$

Thus, the reversed ordering of authorized transitions results in the same final contract state;

$$\langle\sigma_a\rangle \mid \langle\sigma_b\rangle \mid \sigma \xrightarrow{P_b \text{ authorizes } \boldsymbol{F}(x_b;\sigma_b,\sigma)} \langle\sigma_a\rangle \mid \langle\sigma_{bc}\rangle \mid \sigma' \xrightarrow{P_a \text{ authorizes } \boldsymbol{F}(x_a;\sigma_a,\sigma')} \langle\sigma_{ab}\rangle \mid \langle\sigma_{bc}\rangle \mid \sigma''$$

However, even though our example shows logically concurrent transitions over private and public state fragments, in practice, the NIZK proofs generated by P_a and P_b cannot be used in arbitrary order; namely, a ZK proof of statement (a) above, for example, does not imply a valid proof of (d), as the statement is proven over different public states (σ vs. σ'). Similarly, this holds for statement (b) and (c). Since orderings of transactions finalized to the blockchain are unknown at the time of generating transactions, authorized transactions may fail if they are interacting with public state fragments commonly updated by users. This makes the implementation of applications in decentralized finance challenging, as market applications are frequently updated by interactions with different clients.

In **Kachina** [KKK21], an alternative to the UTXO model is proposed for privacy-preserving smart contracts. Here, each update is still authorized by a single user, thereby updating private state(s) as well as a public one; input concurrency is achieved by introducing an **oracle transcript model**. A key idea here is the following; instead of proving a relation over the public state fragment σ explicitly, such as in statements (a-d) above, a statement over a valid interaction transcript $\mathcal{T}$ with a *public state oracle* $\mathcal{O}(\sigma)$ is proven in zero-knowledge. If the same authorized state update is performed on a different public state σ', the zero-knowledge proof remains valid as long as the interaction with $\mathcal{O}(\sigma')$ returns the same transcript $\mathcal{T}$, to which the zero-knowledge proof is bound. Thus, this enables concurrency for input authorizations on private smart contract updates that are logically concurrent. Kachina [KKK21] formalizes the oracle transcript model in the universal composability [Can01] framework. We refer to their work for further details on the oracle transcript model.

5.2 Update of multiple private states

Contract manager model. Towards a model of updates over multiple, single-owner private states, Hawk [KMS+16] proposes a "minimally trusted" **contract manager**; a corrupted contract manager breaks privacy, but the integrity of smart contract updates is preserved. In this model, state transitions occur in **round-based** interactions between users and contract manager, where clients are assumed to have established secure and private communication channels with the contract manager. Thus, such a model of privacy-preserving smart contracts is **not permissionless** (§3.3.1). Still, if this can restriction can be accepted in practice, this model realizes a level of expressiveness unmatched by other proposals.

In the contract manager model, each round consists of a input, evaluation and output phase, where updates to all individual private state fragments is finalized on the blockchain authorized by the contract manager. We illustrate the case for a non-reactive, single-shot contract execution consisting of a single update over single-owner private states. In the **input phase**, let users P_a, P_b and P_c provide private inputs and the cleartext of their private state to the contract manager, who computes contract transition function $\boldsymbol{F}$ over private inputs and state during the **evaluation phase**;

$$(y_a, y_b, y_c; \sigma'_a, \sigma'_b, \sigma'_c) \leftarrow \boldsymbol{F}(x_a, x_b, x_c; \sigma_a, \sigma_b, \sigma_c) \tag{5.1}$$

The contract manager computes both state updates and explicit output values which are privately returned to each participant during the **output phase**; here, the contract manager M also authorizes the update to private state fragments on the blockchain;

$$\langle\sigma_a\rangle \mid \langle\sigma_b\rangle \mid \langle\sigma_c\rangle \xrightarrow{M \text{ authorizes } \boldsymbol{F}(x_a,x_b,x_c;\sigma_a,\sigma_b,\sigma_c)} \langle\sigma'_a\rangle \mid \langle\sigma'_b\rangle \mid \langle\sigma'_c\rangle$$

In Hawk [KMS+16], the contract manager authorization of such a privacy-preserving update to individual private states requires it to generate a verifying zero-knowledge Succinct Argument of Knowledge (zkSNARK) proof, attesting that the update of private states is faithful to a valid evaluation of $\boldsymbol{F}$ over client inputs and their private states. Note that clients must also prove the validity of their "private state" provided to the contract manager during the input phase, as only the owner can access the the current state of its private state fragment. Private state fragments are implemented as in the UTXO model of [SCG+14, BCG+20, XCZ+22]; each round of interaction with the contract manager consumes cryptographic commitments of private states and generates new commitments, each binding an updated private state.

Distributed contract manager. A natural idea is to realize the contract manager by means of secure multi-party computation (MPC), thereby preventing partial committee corruption from breaking privacy; indeed, this is proposed in [KMS+16], but remains impractical due to the requirement for the contract manager to generate a zkSNARK proof inside the MPC protocol itself; computation performed in a MPC incurs a significant overhead, which is compounded by the need to compute cryptographic primitives in addition to the smart contract logic itself. Recent work zkHawk [BCT21] realizes the contract manager model with a **relaxed proof obligation** for the MPC committee acting as the distributed contract manager; here, the contract manager is only required to prove that no tokens were minted during the output phase, but the zero-knowledge proof is still computed inside an MPC circuit, incurring the overhead of MPC computation.

Our contribution **Eagle** in P7 represents the first *efficient* realization of the distributed contract manager with **secure multi-party computation** (MPC), which only evaluates the contract logic inside the MPC protocol over private inputs and state. This is achieved by moving all other cryptographic overhead outside the MPC computation; our proposed input protocol permits clients to efficiently prove consistency between their inputs and private state during the input phase. The output phase of Eagle guarantees that a fully corrupted MPC committee cannot arbitrarily mint tokens on the blockchain, thereby preserving integrity of the underlying assets.

Eagle also extends the contract manager model with long-running, privacy-preserving smart contracts, where a privacy-preserving smart contract is executed over multiple rounds in a reactive manner. Here, clients can optionally choose to participate in a given round and then go offline - the outputs and openings to the updated private state commitments are posted to the blockchain in “masked” form, and thus can be collected and “unmasked” by clients whenever they come online. We establish UC security [Can01] for the Eagle protocol, carefully composing our protocol from UC primitives and demonstrating simulatability of our proposed ideal functionality. We argue that this opens up a novel design space for cryptoeconomic applications, further extended in §5.3 and P8 with notions of differential privacy adopted to this setting that also enable our proposed differentially private market mechanisms.

Permissionless contract manager. We note that permissionless consensus has inspired a recent line of work to investigate multi-party computation in the permissionless setting. Inspired by electing parties to roles via “cryptographic sortion” introduced by Algorand [GHM+17], each party, upon election to a protocol role, computes a single message to send and can then go offline. This has been formalized as the “you-only-speak-once” (YOSO) setting in [GHK+21]; realizing general MPC in this setting requires encrypting and securely forwarding secret-shared state to committee members *elected in the future* [BGG+20, CDGK23]; achieving this in a practical manner remains an open research question. MPC in the YOSO model would enable **permissionless instantiation of the contract manager model** for privacy-preserving smart contracts.

5.3 Differential privacy in smart contracts.

Having established the contract manager model of privacy-preserving smart contracts, which securely updates private states owned by individual participants, we observe that privacy leakage can still occur trivially due to contract logic itself; consider the contract update function $\boldsymbol{F}$ in eq. 5.1, which is evaluated over private inputs and state and returns private outputs $y_1, ..., y_n$. Here, any single output may trivially reveal information that the contract manager privately exchanged with parties. Even though communication between a client and the trusted curator is private, the **smart contract logic** itself may leak private data to computed outputs $y_1, ..., y_n$. In fact, in most meaningful smart contract applications enabling economic coordination between participants, this is necessary. In this thesis, we propose to mitigate such leakage by contributing a novel extension of differential privacy [DR+14] to the setting of the trusted curator model. This represents a generalization of the contract manager model of private smart contracts (P7). Towards achieving **function privacy**, we propose the first market mechanisms which are differentially private in P8. To the best of our knowledge, we are the first to consider differential privacy in such a setting, as classical differential privacy is defined to protect a private data base from queries performed by an analyst; in the trusted curator model, we must protect the entire client transcript from each round of client-curator interaction.

Our definitional framework for differential privacy in the trusted curator model extends directly to **secure multi-party computation** (MPC), where the trusted curator is realized in a distributed manner by interacting servers. The following definitions are relevant whenever *function privacy* is required in MPC computation, and is likely of independent interest.

5.3.1 Definitions: differential privacy in the trusted curator model

We restate a selection of definitions from P8 to introduce our proposed notion of round-differential privacy (Def. 5.3) in the trusted curator model.

The trusted curator model. We restate our model of computation in the trusted curator setting from P8, which is consistent with the contract manager setting model except that any interaction with a blockchain is not explicitly modelled; still, this is sufficient to capture all leakage caused by the evaluated (contract) function. Interaction between clients and trusted curator occur in rounds;

1. **Input phase** All parties send their individual inputs to the trusted curator C, which obtains the input set $x_1, ..., x_n$ from clients $P_1, ..., P_n$ respectively.
2. **Evaluation phase** Upon receiving all inputs, the trusted curator locally computes a known algorithm M over inputs received in the input phase: namely, $\mathbf{y} \leftarrow \mathbf{M}(\mathbf{x})$ where $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{y} = (y_1, ..., y_n)$. Further, curator C is assumed to have access to to randomness to evaluate randomized algorithms.
3. **Output phase** The trusted curator privately sends each output element y_i in $\mathbf{y}$ to client P_i, and enters the input phase again.

The adversary $\mathcal{A}$ is defined such that it can corrupt up to $n-1$ individual clients, but not the trusted curator; the adversary decides the corrupted client input and observes the output received from the trusted curator. We refer to P8 for a detailed discussion of of the trusted curator model.

Round-differential privacy. In the spirit of standard differential privacy (DP) [DR+14], we capture a privacy notion for inputs and outputs each round, such that any "perturbation" in honest input or output values are not "observable" in the input and output of the adversary. In other words, we wish to "bound" the sensitivity of the adversarial transcript to any changes in the honest transcript; here, let transcript simply denote the input and output from a given round.

Towards formal definitions, we first define neighbouring input vectors $\mathbf{x} \sim \mathbf{x}'$ received by the trusted curator to differ only in a single input x_i received from the honest party P_i. Any change to a single honest input element in $\mathbf{x}$ results in neighboring input vector $\mathbf{x}'$. We limit the affect of such a change on adversarial view corrupting up to $n-1$ clients in a given round. Let $\mathbf{M}$ denote the function or mechanism evaluated by the trusted curator and $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$ the output distribution(s) received by the corrupted clients from an evaluation of $\mathbf{M}$ on inputs $\mathbf{x}$.

Definition 5.1 (**(ε, δ)-input-DP**)**.** For an evaluation of (ε, δ)-input differentially private algorithm $\mathbf{M}$ in the trusted curator model over neighboring private input vectors $\mathbf{x} \sim \mathbf{x}'$, the following must hold for any adversarially observable output event $\mathcal{S}^{\mathcal{A}}$.

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}') \in \mathcal{S}^{\mathcal{A}} \,] + \delta$$

In other words, the probability of any set of corrupted outputs observed by $\mathcal{A}$ should differ no more than factor $\exp(\varepsilon)$ when an honest client applies any change to its input. With probability δ, the mechanism is permitted to violate this guarantee; in practical terms, this can be interpreted as a "slack" budget granted to the mechanism designer.

We emphasize that today's prevalent market venues designed to preserve trader privacy do not satisfy input-differential privacy. Here, we consider **dark pools** [CSTA19, dGCP$^+$22] in traditional finance; trade orders are privately submitted to the trusted venue operator, which then outputs the resulting trade executions in private to clients. Note that this setting is consistent with that of the trusted curator. We show how input-differential privacy is violated even by dark pools; assume a corrupted client submitting a sell order observes that its trade order is executed. Any change in the honest counter-party's (privately submitted) buy order may cancel the matching of this order pair, observable to adversary with probability 1, thereby violating Definition 5.1. We overcome this with a technique named *fuzzy matching* in P8.

Next, we introduce correlated-output differential privacy to protect the honest output against *correlation* with other corrupted client outputs.

Definition 5.2 (**(ε, δ)-correlated-output-DP**)**.** For an evaluation of (ε, δ)-correlated output differentially private algorithm $\mathbf{M}$ in the trusted curator model over fixed input vector $\mathbf{x}$, the following must hold for any adversarial output event $\mathcal{S}^{\mathcal{A}}$ and any honest output event $\mathcal{S}^{h}$.

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^{h}(\mathbf{x}) \in \mathcal{S}^{h} \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^{h}(\mathbf{x}) \notin \mathcal{S}^{h} \,] + \delta$$

In words, given a fixed set of inputs submitted to the trusted curator, with probability $1 - \delta$, any change to the *honest output* will, at most, affect the probability of an adversarial output event by factor $\exp(\varepsilon)$. Note that outputs can vary despite fixing inputs, since input-differentially private mechanisms are necessarily randomized.

We argue that achieving this notion of privacy is non-trivial in economic applications or smart contracts, where outputs are necessarily correlated; here, the mechanism is often intended to "allocate" resources or assets over all participating clients. An adversary corrupting $n - 1$ clients can trivially infer funds privately output to the single honest client by just observing its own outputs, if the total supply of funds is known. In our differentially private market mechanisms introduced in P8, this is overcome by the temporary *freezing* of a bounded amount of funds provided by a liquidity provider.

Finally, we define round-differential privacy over both input- and correlated-output privacy.

Definition 5.3 (**Round-DP**)**.** The evaluation of a mechanism that satisfies $(\varepsilon^{\mathsf{in}}, \delta^{\mathsf{in}})$-input-DP and $(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-correlated-output-DP is $(\varepsilon^{\mathsf{in}}, \delta^{\mathsf{in}})$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-round differentially private.

We refer to P8 for a generalization of round-differential privacy to the multi-round setting, where clients perform multi-round interactions with the trusted curator (or contract manager).

5.3.2 Differentially private market mechanisms

In P8, we contribute round-differentially private (Def. 5.3) market mechanism designs, which improve on traditional dark pools by providing formal privacy guarantees for the full client transcript. Such privacy notions ensure pre-trade and post-trade privacy, enabling the fair execution of long-running trade strategies (see §4.3.2), which otherwise cannot be formally guaranteed in traditional dark pool venues. We restate a summary our market mechanism from P8, where we also contribute MPC-friendly algorithms for our round-differentially private mechanism designs.

Input phase: Traders privately submit limit orders to the venue operator. Our auction mechanism requires a market making *liquidity provider* to compensates for the noise added to trade matching.

Auction phase: A deterministic, optimal order matching is performed; such a matching will leak the inclusion or exclusion of a single trade request in the outputs and is not differentially private. The actual "trade", "no-trade" outcome for each order is determined by *sampling* a Bernoulli distribution biased towards the deterministically computed, optimal matching; however, since trades are filled or not filled based on independently sampling trade outcomes, there is no guarantee that each executed trade is matched with an equivalent volume in opposing direction; as such a liquidity deficit may occur. Here, market makers make up for liquidity deficits. To prevent market makers from learning about the traded volume of a single user (output privacy) from their updated liquidity balances, a random, yet bounded amount of market maker liquidity is *frozen* to obtain (ε, δ)-correlated-output differential privacy.

Output phase: Traders observe whether or not their order was fulfilled and liquidity providers observe a noisy update to the liquidity balance. The trade output *distribution* is ε-indistinguishable with respect to the inclusion or exclusion of a single submitted order. The liquidity balance distribution is (ε, δ)-indistinguishable with respect to a single trade outcome.

Part III

Publications & Manuscripts

A Theory of Automated Market Makers in DeFi

Publication Information

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. “A theory of Automated Market Makers in DeFi.” In *Coordination Models and Languages: 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14–18, 2021, Proceedings 23*, pp. 168-187. Springer International Publishing, 2021.

Contribution

- Co-author.

Remarks

Conference version.

A theory of Automated Market Makers in DeFi

Massimo Bartoletti[1][0000−0003−3796−9774],
James Hsin-yu Chiang[2][0000−0002−5126−9494], and
Alberto Lluch-Lafuente[2][0000−0001−7405−0818]

[1] Università degli Studi di Cagliari, Cagliari, Italy bart@unica.it
[2] Technical University of Denmark, DTU Compute, Copenhagen, Denmark
{jchi,albl}@dtu.dk

Abstract. Automated market makers (AMMs) are one of the most prominent decentralized finance (DeFi) applications. They allow users to exchange units of different types of crypto-assets, without the need to find a counter-party. There are several implementations and models for AMMs, featuring a variety of sophisticated economic mechanisms. We present a theory of AMMs. The core of our theory is an abstract operational model of the interactions between users and AMMs, which can be concretised by instantiating the economic mechanisms. We exploit our theory to formally prove a set of fundamental properties of AMMs, characterizing both structural and economic aspects. We do this by abstracting from the actual economic mechanisms used in implementations and identifying sufficient conditions which ensure the relevant properties. Notably, we devise a general solution to the *arbitrage problem*, the main game-theoretic foundation behind the economic mechanisms of AMMs.

1 Introduction

Decentralized finance (DeFi) is emerging as an alternative to the traditional finance, boosted by blockchain-based crypto-tokens and smart contracts. One of the main DeFi applications are *Automated Market Makers (AMMs)*, which allow users to exchange crypto-tokens of different types without the intermediation of third parties. As of April 2021, the two AMM platforms leading by user activity, Uniswap [14] and Curve Finance [5], alone hold \$8.1B and \$6.2B worth of tokens, and process \$1.5B and \$210M worth of transactions daily [4, 12].

AMMs are inherently hard to design, implement and understand, since they involve sophisticated economic incentive mechanisms. Although they generally only expose a handful of callable functions, interactions with AMMs are sensitive to transaction ordering [21,23,27,30]: thus, actors with the power to influence the order of transactions in the blockchain may be incentivized to do so for profit or to harm specific users. Thus, there exists a need for foundational work to devise formal models of AMMs which allow the study of their fundamental properties, transaction concurrency and the effect of economic incentives.

Current descriptions of AMMs are either economic models [17–19, 24], which focus on the efficacy of incentive design, or the actual implementations. While economic models are useful to understand the macroscopic financial aspects of AMMs, they do not precisely describe the interactions between AMMs and their users. Still, understanding these interactions is crucial to determine possible deviations from the expected behaviour. Implementations, instead, reflect the exact behaviour of AMMs, but at a level of detail that hampers high-level understanding and reasoning. Moreover, the rich variety of implementations, proposals and models for AMMs, each featuring different sophisticated economic mechanisms, makes it difficult to establish comparisons between AMM designs or to provide a clear contour for the space of possible "well behaving" designs.

Contributions In this paper we address these challenges by developing a theory of AMMs. The core of our theory is a formal model of AMMs (§2), based on a thorough inspection of leading AMM implementations like Uniswap [13], Curve [16], and Balancer [3], as well as existing models from the literature [1,30]. Our model precisely describes the interactions between users and AMMs, and their main economic features. An original aspect of our model is that it is parametric with respect to the key economic mechanism — the *swap invariant* — that algorithmically determines exchange rates between tokens. This makes our model general enough to encompass the mainstream implementations and models of AMMs. With respect to economic models, our theory considers implementation details that are crucial to guarantee (efficient) computability in practice. Our model features an *executable semantics*, which can support implementations and analysis tools. As a matter of fact, an open-source Ocaml implementation of our executable semantics is provided as a companion of this paper.[3]

Building upon our model, we prove a set of properties characterizing both structural (§3) and economic (§4) aspects of AMMs. With respect to previous works, which focus on specific economic mechanisms, all our results are parametric with respect to swap invariants. We identify indeed, for each property, a set of conditions on swap invariants that are sufficient for the property to hold. Our results include fundamental structural properties such as *net worth preservation* ("value cannot be created/destroyed"), *liquidity* ("assets cannot be frozen within an AMM"), and *transaction concurrency* ("two transactions can be executed in any order"), as well as fundamental economic properties such as *incentive-consistency*, which ensures an incentive feedback loop between deposits and swaps of tokens. Most notably, we generalize the formulation and the solution to the so-called *arbitrage problem*, the main game-theoretic foundation behind the economic aspects of AMMs. We show that users are incentivized to perform actions that keep the swap rates aligned with the exchange rates given by price oracles. Namely, if an AMM offers a better swap rate than the oracles' exchange rate, rational users will perform swaps to narrow the gap. Further, we show that, under certain conditions, deposits and swaps incentivize each other.

Overall, our theory encompasses and generalizes the main functional and economic aspects of the mainstream AMM implementations, providing solid grounds for the design of future AMMs.

[3] https://github.com/blockchain-unica/defi-workbench

2 A formal model of Automated Market Makers

We introduce a formal, operational model of AMMs, focussing on the common features implemented by the main AMM platforms. We discuss in §6 the differences between these platforms and our model.

2.1 AMM states

Basics We assume a set of ***users*** $\mathbb{A}$, ranged over by $\mathsf{A}, \mathsf{A}', \ldots$, and a set of ***token types*** $\mathbb{T}$, ranged over by $\tau, \tau', \ldots$. We denote with $\mathbb{T}_0 \subseteq \mathbb{T}$ a specific subset of token types that we call *initial* (they include, e.g., native blockchain tokens). The rest of the token types in $\mathbb{T}$ represent *minted* tokens, denoted as pairs (τ, τ') of distinct token types, and which represent shares in an AMM. We use v, v', r, r' to range over nonnegative real numbers ($\mathbb{R}_0^+$), and we write $r : \tau$ to denote r units of token type τ. We denote with $\operatorname{dom} f$ the domain of a partial map f. We model the ***wallet*** of a user A as a term $\mathsf{A}[\sigma]$, where the partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_0^+$ represents A's token holdings. We model an ***AMM*** as a pair of the form $(r_0 : \tau_0, r_1 : \tau_1)$, representing the fact that the AMM is holding, respectively, r_0 and r_1 units of token types τ_0 and τ_1.

States We formalise the interaction between users and AMMs as a labelled transition system (LTS). Its labels $\mathsf{T}, \mathsf{T}', \ldots$ represent blockchain ***transactions***, while the ***states*** $\Gamma, \Gamma', \ldots$ are compositions of wallets and AMMs:

$$\mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid (r_1 : \tau_1, r_1' : \tau_1') \mid \cdots \mid (r_k : \tau_k, r_k' : \tau_k')$$

where all A_i are distinct, and for all $i \neq j$: $\tau_i \neq \tau_i'$ (i.e., the token types in an AMM are *distinct*), and $(\tau_i = \tau_j \Rightarrow \tau_i' \neq \tau_j') \wedge (\tau_i = \tau_j' \Rightarrow \tau_i' \neq \tau_j)$ (i.e., distinct AMMs cannot hold exactly the same token types). Two AMMs can indeed have a common token type τ, as in $(r_1 : \tau_1, r : \tau)$, $(r' : \tau, r_2' : \tau_2')$, thus enabling indirect trades between token pairs not directly provided by any AMM. A state Γ is *initial* when it only contains wallets with initial tokens. We treat states as sets of terms (wallets/AMMs): hence, Γ and Γ' are equivalent when they contain the same terms; for a term Q, we write $Q \in \Gamma$ when $\Gamma = Q \mid \Gamma'$, for some Γ'.

Example 1. Figure 1 shows an execution trace in our model, that we will explain in detail later in Example 5. We write $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ for a state transition from Γ to Γ', triggered by a transaction T. The first two states are initial, while the others contain an AMM for a token pair (τ_0, τ_1). □

Token supply We define the ***supply*** of a token type τ in a state Γ as the sum of the balances of τ in all the wallets and the AMMs occurring in Γ. Formally:

$$\mathit{sply}_\tau(\mathsf{A}[\sigma]) = \begin{cases} \sigma(\tau) & \text{if } \tau \in \operatorname{dom}\sigma \\ 0 & \text{otherwise} \end{cases} \qquad \mathit{sply}_\tau(r_0 : \tau_0, r_1 : \tau_1) = \begin{cases} r_i & \text{if } \tau = \tau_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathit{sply}_\tau(\Gamma \mid \Gamma') = \mathit{sply}_\tau(\Gamma) + \mathit{sply}_\tau(\Gamma')$$

$$\mathsf{A}[70:\tau_0, 80:\tau_1] \mid \mathsf{B}[30:\tau_0]$$

$$\xrightarrow{\mathsf{A}:\mathsf{xfer}(\mathsf{B},10:\tau_1)} \mathsf{A}[70:\tau_0, 70:\tau_1] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \tag{1}$$

$$\xrightarrow{\mathsf{A}:\mathsf{dep}(70:\tau_0,70:\tau_1)} \mathsf{A}[70:(\tau_0,\tau_1)] \mid \mathsf{B}[\cdots] \mid (70:\tau_0, 70:\tau_1) \tag{2}$$

$$\xrightarrow{\mathsf{B}:\mathsf{swapL}(30:\tau_0,20:\tau_1)} \mathsf{A}[\cdots] \mid \mathsf{B}[0:\tau_0, 31:\tau_1] \mid (100:\tau_0, 49:\tau_1) \tag{3}$$

$$\xrightarrow{\mathsf{B}:\mathsf{swapR}(29:\tau_0,21:\tau_1)} \mathsf{A}[\cdots] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \mid (70:\tau_0, 70:\tau_1) \tag{4}$$

$$\xrightarrow{\mathsf{B}:\mathsf{rdm}(30:(\tau_0,\tau_1))} \mathsf{A}[30:\tau_0, 30:\tau_1, 40:(\tau_0,\tau_1)] \mid \mathsf{B}[\cdots] \mid (40:\tau_0, 40:\tau_1) \tag{5}$$

$$\xrightarrow{\mathsf{B}:\mathsf{swapL}(30:\tau_0,16:\tau_1)} \mathsf{A}[\cdots] \mid \mathsf{B}[0:\tau_0, 27:\tau_1] \mid (70:\tau_0, 23:\tau_1) \tag{6}$$

$$\xrightarrow{\mathsf{A}:\mathsf{rdm}(30:(\tau_0,\tau_1))} \mathsf{A}[82:\tau_0, 47:\tau_1, 10:(\tau_0,\tau_1)] \mid \mathsf{B}[\cdots] \mid (18:\tau_0, 6:\tau_1) \tag{7}$$

Fig. 1: Interactions between two users and an AMM.

Example 2. Consider the first state in Figure 1, $\Gamma_1 = \mathsf{A}[70:\tau_0, 80:\tau_1] \mid \mathsf{B}[30:\tau_0]$. We have that $\mathit{sply}_{\tau_0}(\Gamma_1) = 70 + 30 = 100$, while $\mathit{sply}_{\tau_1}(\Gamma_1) = 80$. Observe that the supply of both token types remains constant in Figure 1; we will show in Lemma 2 that the supply of initial token types is always preserved. □

Token prices and net worth Assume that initial tokens are priced by a global oracle $P^0 \in \mathbb{T}_0 \to \mathbb{R}_0^+$. We then define the ***price*** $P_\tau(\Gamma)$ of a token $\tau \in \mathbb{T}$ (either initial or minted) in a state Γ inductively as follows:

$$\begin{aligned} P_\tau(\Gamma) &= P^0(\tau) && \text{if } \tau \in \mathbb{T}_0 \\ P_{(\tau_0,\tau_1)}(\Gamma) &= \frac{r_0 \cdot P_{\tau_0}(\Gamma) + r_1 \cdot P_{\tau_1}(\Gamma)}{\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)} && \text{if } (r_0:\tau_0, r_1:\tau_1) \in \Gamma \end{aligned} \tag{8}$$

The main idea is that initial tokens are priced using directly the global oracle while minted tokens are priced under the assumption that they can be redeemed. Their price, hence, is obtained by (recursively) calculating the price of the tokens that can be obtained by redeeming them (i.e. the proportions of the reserves r_0 and r_1 given the current supply). This intuition will be further formalized later in Lemma 6.

Example 3. Let $\Gamma_7 = \mathsf{A}[82:\tau_0, 47:\tau_1, 10:(\tau_0,\tau_1)] \mid \cdots$ be the final state in Figure 1. We have that $\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma_7) = 10$. Assume that the prices of initial tokens are $P^0(\tau_0) = 5$ and $P^0(\tau_1) = 9$. The price of the minted token (τ_0,τ_1) is hence:

$$P_{(\tau_0,\tau_1)}(\Gamma_7) = \frac{1}{10}\Big(18 \cdot P_{\tau_0}(\Gamma_7) + 6 \cdot P_{\tau_1}(\Gamma_7)\Big) = \frac{18}{10} \cdot 5 + \frac{6}{10} \cdot 9 = 14.4$$ □

We now define a key concept to understand the incentives for users to participate in AMMs, namely the ***net worth*** of a user A in a state Γ:

$$W_{\mathsf{A}}(\Gamma) = \begin{cases} \sum_{\tau \in \operatorname{dom} \sigma} \sigma(\tau) \cdot P_\tau(\Gamma) & \text{if } \mathsf{A}[\sigma] \in \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The ***global net worth*** $W(\Gamma)$ of a state Γ is the sum of the net worth in users' wallets. The token units held in AMMs are not accounted for by $W(\Gamma)$, because their value is already recorded by minted tokens held in users' wallets. Indeed, the equality $sply_{(\tau_0,\tau_1)}(\Gamma) \cdot P_{(\tau_0,\tau_1)}(\Gamma) = r_0 \cdot P_{\tau_0}(\Gamma) + r_1 \cdot P_{\tau_1}(\Gamma)$ between the net value of a minted token and the value of the AMM is a direct consequence of the definition of price in (8).

As we shall see later, one of the main goals of users is to maximize their net worth. This can be achieved through different interactions with the AMM (e.g., by investing tokens or trading units of differently priced token types).

Example 4. Recall from Figure 1 the state $\Gamma_1 = \mathsf{A}[70 : \tau_0, 80 : \tau_1] \mid \mathsf{B}[30 : \tau_0]$, where τ_0 and τ_1 are initial tokens. Assume again that the prices are $P^0(\tau_0) = 5$ and $P^0(\tau_1) = 9$. The users' net worth in Γ_1 are then:

$$W_{\mathsf{A}}(\Gamma_1) = 70 \cdot P^0(\tau_0) + 80 \cdot P^0(\tau_1) = 1070 \qquad W_{\mathsf{B}}(\Gamma_1) = 30 \cdot P^0(\tau_0) = 150$$

In $\Gamma_7 = \mathsf{A}[82 : \tau_0, 47 : \tau_1, 10 : (\tau_0, \tau_1)] \mid \mathsf{B}[0 : \tau_0, 27 : \tau_1] \mid (18 : \tau_0, 6 : \tau_1)$ we have:

$$\begin{aligned} W_{\mathsf{A}}(\Gamma_7) &= 82 \cdot P_{\tau_0}(\Gamma_7) + 47 \cdot P_{\tau_1}(\Gamma_7) + 10 \cdot P_{(\tau_0,\tau_1)}(\Gamma_7) = 977 \\ W_{\mathsf{B}}(\Gamma_7) &= 27 \cdot P_{\tau_1}(\Gamma_7) = 243 \end{aligned}$$

Note that the net worth of A has decreased w.r.t. the initial state, while the net worth of B has increased. One may think that B has been more successful than A, but this depends on the users' goals. Note, e.g., that A holds 10 units of the minted token (τ_0, τ_1), whose price may increase in the future. □

2.2 AMM semantics

We now formally describe the interactions of the AMM that give rise to state transitions. State transitions are triggered by the ***transactions*** in Table 1. We formalise below their behaviour, but we give before an overview of our running example from Figure 1.

Example 5. Figure 1 actually displays a sequence of transitions in the LTS of our model. To keep the example simple, we have used there the *constant product* swap invariant, which requires swap transactions to preserve the product between the amounts of the two tokens in the AMM; further, we have assumed no fees. In step (1), A transfers $10 : \tau_1$ from her wallet to B's. In step (2), A creates a new AMM, depositing $70 : \tau_0$ and $70 : \tau_1$; in return, she receives 70 units of the minted token (τ_0, τ_1). In step (3), B swaps 30 of his units of τ_0 for *at least* 20 units of τ_1. The actual amount of units of τ_1 received by B is 21: indeed, $(70 + 30) \cdot (70 - 21) = 70 \cdot 70$, hence 21 satisfies the constant product swap invariant. In step (4), B reverses his prior action by swapping 21 of his units of τ_1 for *at least* 29 units of τ_0. Here, the actual amount of units of τ_1 received by B is 30, which also satisfies the constant product swap invariant. In step (5), B redeems 30 units of the minted token (τ_0, τ_1), accordingly reducing the funds in the AMM. Note that the received tokens exhibit the same 1-to-1 ratio as in the

$\mathsf{A} : \mathsf{xfer}(\mathsf{B}, v : \tau)$	A transfers $v : \tau$ to B
$\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$	A deposits $v_0 : \tau_0$ and $v_1 : \tau_1$ to an AMM $(r_0 : \tau_0, r_1 : \tau_1)$, receiving in return some units of the minted token (τ_0, τ_1)
$\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$	A tranfers $v_0 : \tau_0$ to an AMM $(r_0 : \tau_0, r_1 : \tau_1)$, receiving in return *at least* v_1 units of τ_1
$\mathsf{A} : \mathsf{swapR}(v_0 : \tau_0, v_1 : \tau_1)$	A tranfers $v_1 : \tau_1$ to an AMM $(r_0 : \tau_0, r_1 : \tau_1)$, receiving in return *at least* v_0 units of τ_0
$\mathsf{A} : \mathsf{rdm}(v : \tau)$	A redeems v units of minted token $\tau = (\tau_0, \tau_1)$ from an AMM $(r_0 : \tau_0, r_1 : \tau_1)$, receiving in return some units of τ_0 and τ_1

Table 1: AMM transactions.

initial deposit at step (2). In step (6), B swaps 30 of his units of τ_0 for *at least* 16 units of τ_1. Unlike in the previous swap at step (3), now the actual amount of τ_1 received by B is 17. Note that the implied *swap rate* between received τ_1 units and sent τ_0 units has deteriorated w.r.t. step (3), even if the pair (τ_0, τ_1) had the same 1-to-1 ratio of funds. This is caused by the reduction in funds resulting from A's redeem action: thus, the swap rate is sensitive to both the ratio of funds in the pair as well as their absolute balances, a key property of the incentive mechanisms, as we shall see later in §4. Finally, in step (7) A performs another redeem of 30 units of the minted token (τ_0, τ_1), thereby extracting 52 units of τ_0 and 17 units of τ_1 from the AMM. Note that the ratio of redeemed tokens is no longer 1-to-1 as in the previous redeem action (5), as the prior left swap has changed the ratio between the funds of τ_0 and τ_1 in the AMM. □

We now formalise the transition rules. We use the standard notation $\sigma\{^v/_x\}$ to update a partial map σ at point x: namely, $\sigma\{^v/_x\}(x) = v$, while $\sigma\{^v/_x\}(y) = \sigma(y)$ for $y \neq x$. Given a partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_0^+$, a token type $\tau \in \mathbb{T}$ and a partial operation $\circ \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \rightharpoonup \mathbb{R}_0^+$, we define the partial map $\sigma \circ v : \tau$ as follows:

$$\sigma \circ v : \tau = \begin{cases} \sigma\{^{\sigma(\tau) \circ v}/_\tau\} & \text{if } \tau \in \operatorname{dom}\sigma \text{ and } \sigma(\tau) \circ v \in \mathbb{R}_0^+ \\ \sigma\{^v/_\tau\} & \text{if } \tau \notin \operatorname{dom}\sigma \end{cases}$$

Token transfer A user A can transfer some of her tokens to another user B, provided that there are enough units of the token in A's wallet. Formally:

$$\frac{\sigma_{\mathsf{A}}(\tau) \geq v}{\mathsf{A}[\sigma_{\mathsf{A}}] \mid \mathsf{B}[\sigma_{\mathsf{B}}] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{xfer}(\mathsf{B},v:\tau)} \mathsf{A}[\sigma_{\mathsf{A}} - v : \tau] \mid \mathsf{B}[\sigma_{\mathsf{B}} + v : \tau] \mid \Gamma} \ [\textsc{Xfer}]$$

A consequence of this rule is that tokens (both initial and minted) are *fungible*, i.e. individual units of the same token type are interchangeable. In particular, amounts of tokens of the same type can be split into smaller parts, and two amounts of tokens of the same type can be joined.

Deposit Any user can create an AMM for a token pair (τ_0, τ_1) provided that such an AMM is not already present in the state. This is achieved by the transaction $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, through which A transfers $v_0 : \tau_0$ and $v_1 : \tau_1$ to the new AMM. In return for the deposit, A receives a certain positive amount of units of a new token type (τ_0, τ_1), which is minted by the AMM. The exact amount of units received is irrelevant. In our model we choose v_0 but any other choice would be valid. We formalise this behaviour by the rule:

$$\frac{\sigma(\tau_i) \geq v_i > 0 \ \ (i \in \{0,1\}) \qquad \tau_0 \neq \tau_1 \qquad (\_ : \tau_0, \_ : \tau_1), (\_ : \tau_1, \_ : \tau_0) \not\in \Gamma}{\begin{array}{l} \mathsf{A}[\sigma] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : (\tau_0, \tau_1)] \mid (v_0 : \tau_0, v_1 : \tau_1) \mid \Gamma \end{array}} \text{[Dep0]}$$

Once an AMM is created, any user can deposit tokens into it, as long as doing so preserves the ratio of the token holdings in the AMM. When a user deposits $v_0 : \tau_0$ and $v_1 : \tau_1$ to an existing AMM, it receives in return an amount of minted tokens of type (τ_0, τ_1). This amount is the ratio between the deposited amount v_0 and the ***redeem rate*** of (τ_0, τ_1) in the current state Γ, i.e. the ratio between the amount r_0 of τ_0 stored in the AMM, and the total supply $\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)$ of the minted token in the state.

$$\frac{\sigma(\tau_i) \geq v_i > 0 \ \ (i \in \{0,1\}) \qquad r_1 v_0 = r_0 v_1 \qquad v = \frac{v_0}{r_0} \cdot \mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)}{\begin{array}{l} \Gamma \ = \mathsf{A}[\sigma] \ \mid \ (r_0 : \tau_0, r_1 : \tau_1) \ \mid \ \Gamma' \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : (\tau_0, \tau_1)] \ \mid \ (r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1) \ \mid \ \Gamma' \end{array}} \text{[Dep]}$$

Note that the premise $r_1 v_0 = r_0 v_1$ ensures that the ratio between the holdings of τ_0 and τ_1 in the AMM is preserved by the dep transaction, i.e.:

$$\frac{r_1 + v_1}{r_0 + v_0} = \frac{r_1}{r_0}$$

As we shall see in §4, users are incentivized to invest tokens into AMMs by the fact that trading operations (i.e., swaps) are subject to a fee mechanism that makes the redeem rate increase over time.

Swap As shown in step (3) and on of Example 5, users can increase their net worth by swapping tokens. Any user A can swap units of τ_0 in her wallet for units of τ_1 in an AMM $(r_0 : \tau_0, r_1 : \tau_1)$ by firing a transaction $\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$. Here, v_0 is the amount of τ_0 transferred from A's wallet to the AMM, while v_1 is a *lower bound* on the amount of τ_1 that A will receive in return. The actual amount v is determined by a ***swap invariant*** $I \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, that must hold between the amounts of τ_0 and τ_1 held in the AMM before and after the swap. To determine v, the AMM requires a fraction $0 < \phi \leq 1$ of v_0; the rest is considered as a fee (the parameter ϕ is the ***fee rate***). Formally:

$$\frac{\sigma(\tau_0) \geq v_0 > 0 \qquad I(r_0 + \phi v_0, r_1 - v) = I(r_0, r_1) \qquad 0 < v_1 \leq v \leq r_1}{\begin{array}{l} \mathsf{A}[\sigma] \ \mid \ (r_0 : \tau_0, r_1 : \tau_1) \ \mid \ \Gamma \xrightarrow{\mathsf{A}:\mathsf{swapL}(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0 : \tau_0 + v : \tau_1] \ \mid \ (r_0 + v_0 : \tau_0, r_1 - v : \tau_1) \ \mid \ \Gamma \end{array}} \text{[SwapL]}$$

The effect of the fee is that the redeem rate of minted tokens increases; intuitively, the AMM retains a portion of the swapped amounts, but the overall reserve is still distributed among all minted tokens, thereby ensuring liquidity (as we shall formally establish liquidity later on in Lemma 4).

Although actual AMM implementations use a variety of different swap invariants, with the common aim to incentivize users to perform swaps, all these invariants share a few common design choices. A crucial one is that there exists *exactly* one v which satisfies the equation in the premise of [SwapL]; further, swapping 0 units of τ_0 results in 0 units of τ_1. Formally, for all $r_0, r_1 > 0$:

$$\forall v \in \mathbb{R}_0^+ : \exists! v' \in \mathbb{R}_0^+ : I(r_0 + v, r_1 - v') = I(r_0, r_1) \tag{10}$$

Hereafter, we assume that I always respects this condition. A common swap invariant, implemented e.g. by Uniswap [13] and Mooniswap [7] (and also used in Example 5), is the *constant product invariant*, which requires that the product of the amounts of τ_0 and τ_1 in the AMM remains constant, i.e. $I(r_0, r_1) = r_0 \cdot r_1$.

The rule [SwapR] allows for swaps in the other direction:

$$\frac{\sigma(\tau_1) \geq v_1 > 0 \qquad I(r_0 - v, r_1 + \phi\, v_1) = I(r_0, r_1) \qquad 0 < v_0 \leq v \leq r_0}{\begin{array}{l}\mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{swapR}(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma + v : \tau_0 - v_1 : \tau_1] \mid (r_0 - v : \tau_0, r_1 + v_1 : \tau_1) \mid \Gamma\end{array}} \text{[SwapR]}$$

where we assume that I enjoys the "right" version of the condition (10).

It is worth explaining why the swap transactions specify lower bounds for the amount of return tokens, instead of an exact amount. In practice, when a user emits a transaction, she cannot predict the exact state in which the transaction will be actually committed. This makes it unfeasible to guess the exact amount that will preserve the swap invariant: hence, users can only specify a lower bound that they are willing to accept.

Redeem Any user can redeem units of a minted token (τ_0, τ_1), obtaining in return units of the underlying tokens τ_0 and τ_1. The redeemable amounts are determined by the redeem rate: each unit of (τ_0, τ_1) can be redeemed for equal fractions of τ_0 and τ_1 remaining in the AMM:

$$\frac{\sigma(\tau_0, \tau_1) \geq v > 0 \qquad v_0 = v \frac{r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)} \qquad v_1 = v \frac{r_1}{sply_{(\tau_0,\tau_1)}(\Gamma)}}{\begin{array}{l}\Gamma = \mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma' \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:(\tau_0,\tau_1))} \\ \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : (\tau_0, \tau_1)] \mid (r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1) \mid \Gamma'\end{array}} \text{[Rdm]}$$

Example 6. Figure 2 shows the evolution of the AMM token holdings resulting from the trace in Figure 1, presented with Example 5. Recall that we have assumed a constant product swap invariant $x \cdot y = k$, and no swap fees ($\phi = 1$). We refer to a state in Figure 1 by the action number preceding it: the AMM $(70 : \tau_0, 70 : \tau_1)$ in state (2) is shown in Figure 2. Subsequent left (3) and right (4) swaps result in a traversal along $k = 70 \cdot 70$ from $(70 : \tau_0, 70 : \tau_1)$ to $(100 : \tau_0, 49 : \tau_1)$, and back as the swap invariant must hold for swap actions. The redeem action (5) reduces the holdings of both tokens by the same factor to reach $(40 : \tau_0, 40 : \tau_1)$. A left swap (6) traverses $k' = 40 \cdot 40$ to reach $(70 : \tau_0, 23 : \tau_1)$ in state (6), which is then followed by another redeem (7) action, reducing both token holdings proportionally to $(18 : \tau_0, 6 : \tau_1)$.

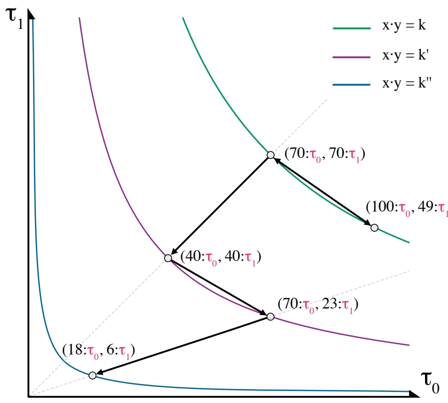


Fig. 2: Evolution of balances of AMM (τ_0, τ_1) along the trace in Figure 1.

3 Structural properties of AMMs

We now establish some structural properties of AMMs, which do not depend on the design of the economic mechanisms, i.e. on the choice of the swap invariant. We denote with $\to^*$ the reflexive and transitive closure of $\to$. Given a finite sequence of transactions $\lambda = \mathsf{T}_1 \cdots \mathsf{T}_k$, we write $\Gamma \xrightarrow{\lambda} \Gamma'$ when $\Gamma \xrightarrow{\mathsf{T}_1} \cdots \xrightarrow{\mathsf{T}_k} \Gamma'$. We say that a state Γ is *reachable* if $\Gamma_0 \to^* \Gamma$ for some initial Γ_0. We denote with $\mathit{type}(\mathsf{T})$ the type of T (i.e., $\mathsf{xfer}, \mathsf{dep}, \ldots$), with $\mathit{wal}(\mathsf{T})$ the set of wallets affected by T (e.g., $\mathit{wal}(\mathsf{A} : \mathsf{xfer}(\mathsf{B}, v : \tau)) = \{\mathsf{A}, \mathsf{B}\}$), and with $\mathit{tok}(\mathsf{T})$ the set of token types affected by T (e.g., $\mathit{tok}(\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)) = \{\tau_0, \tau_1\}$).

First, we establish that the AMMs' LTS is deterministic. Note that, in swap rules, an unconstrained swap invariant I could admit different solutions to the equation in the premise: determinism is ensured by condition (10), which we assume to be true for all swap invariants.

Lemma 1 (Determinism). *If $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ and $\Gamma \xrightarrow{\mathsf{T}} \Gamma''$, then $\Gamma' = \Gamma''$.*

We can lift the statement to sequences of transactions by using a simple inductive argument. The same applies to other single-step results in this section.

Lemma 2 ensures that the supply of each *initial* token type τ is preserved by transitions (of any type). Note that preservation does not hold for *minted* tokens, as they can be created (by rule [Dep]) and destroyed (by rule [Rdm]).

Lemma 2. *For all $\tau \in \mathbb{T}_0$, if $\Gamma \to \Gamma'$ then $\mathit{sply}_\tau(\Gamma) = \mathit{sply}_\tau(\Gamma')$.*

Lemma 3 ensures that the *global* net worth is preserved by transactions, whereas the user's net worth is preserved only by redeems/deposits.

Lemma 3 (Preservation of net worth). *Let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. Then, $W(\Gamma) = W(\Gamma')$. Further, if $\mathit{type}(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$ or $\mathsf{A} \notin \mathit{wal}(\mathsf{T})$, then $W_{\mathsf{A}}(\Gamma) = W_{\mathsf{A}}(\Gamma')$.*

Lemma 4 ensures that funds cannot be *frozen* in an AMM, i.e. that users can always redeem arbitrary amounts of the tokens deposited in an AMM.

Lemma 4 (Liquidity). *Let Γ be a reachable state such that $(r_0 : \tau_0, r_1 : \tau_1) \in \Gamma$ with $r_0 + r_1 > 0$. Then: (a) $sply_{(\tau_0,\tau_1)}(\Gamma) > 0$; (b) for all $r'_0 \leq r_0$, there exists $r'_1 \leq r_1$ such that $\Gamma \to^* (r'_0 : \tau_0, r'_1 : \tau_1) \mid \cdots$; (c) for all $r'_1 \leq r_1$, there exists $r'_0 \leq r_0$ such that $\Gamma \to^* (r'_0 : \tau_0, r'_1 : \tau_1) \mid \cdots$.*

We now study the concurrency of transactions. Two finite sequences of transactions λ_0 and λ_1 are *observationally equivalent*, in denoted $\lambda_0 \sim \lambda_1$, when, for all states Γ, if $\Gamma \xrightarrow{\lambda_0} \Gamma_0$ and $\Gamma \xrightarrow{\lambda_1} \Gamma_1$ then $\Gamma_0 = \Gamma_1$. We say that two distinct transactions T, T' are *concurrent* (denoted, $\mathsf{T} \# \mathsf{T}'$) if $\mathsf{T}\mathsf{T}' \sim \mathsf{T}'\mathsf{T}$. Note that this does not mean that T and T' cannot disable each other as demanded by stricter notions of concurrency. Lemma 5 provides sufficient conditions for two transactions to be concurrent: intuitively, two non-swap transactions are always concurrent, while swap transactions are concurrent with xfer transactions, and with any transactions which do not affect the same token types.

Lemma 5. *Two distinct transactions T_0, T_1 are concurrent if, for $i \in \{0, 1\}$, $type(\mathsf{T}_i) \in \{\mathsf{swapL}, \mathsf{swapR}\}$ implies $tok(\mathsf{T}_i) \cap tok(\mathsf{T}_{1-i}) = \emptyset$ or $type(\mathsf{T}_{1-i}) = \mathsf{xfer}$.*

As we shall see later in §4, it is actually desirable, and crucial for the economic mechanism of AMMs, that swap transactions interfere with other transactions that trade the same token type.

The theory of Mazurkiewicz's trace languages [26] allows us to lift Lemma 5 to sequences of transactions. Let R be a symmetric and irreflexive relation on the set $\mathbb{X}$ of all transactions. The *Mazurkiewicz equivalence* $\sim_R$ is the least congruence in the free monoid $\mathbb{X}^*$ such that: $\forall \mathsf{T}, \mathsf{T}' \in \mathbb{X}$: $\mathsf{T} \, R \, \mathsf{T}' \implies \mathsf{T}\mathsf{T}' \sim_R \mathsf{T}'\mathsf{T}$. Theorem 1 states that the Mazurkiewicz equivalence constructed on the concurrency relation $\#$ is an observational equivalence.

Theorem 1 (Concurrent transactions can be reordered). $\sim_\# \subseteq \sim$.

A direct consequence of Theorem 1 is that we can transform a finite sequence of transactions into an observationally equivalent one by repeatedly exchanging adjacent concurrent transactions — provided that both sequences are executable in the LTS. For example, sequences of $\mathsf{A} : \mathsf{rdm}(\_)$ transactions can be freely reordered, resulting in the same, unique state. This is exploited in the following lemma, which supports the inductive definition of the price of minted tokens in (8): indeed, computing the net worth of a user A under that price definition corresponds to making A first redeem all her minted tokens, and then summing the price of the resulting initial tokens.

Lemma 6. *For all states Γ and users A, let $rdm_{\mathsf{A}}(\Gamma)$ be the unique state reached from Γ by performing only $\mathsf{A} : \mathsf{rdm}(\_)$ actions, such that A's wallet in $rdm_{\mathsf{A}}(\Gamma)$, only contains initial tokens. Then:*

$$W_{\mathsf{A}}(\Gamma) \; = \; \textstyle\sum_{\tau \in \operatorname{dom} \sigma} \sigma(\tau) \cdot P^0(\tau) \qquad \text{if } \mathsf{A}[\sigma] \in rdm_{\mathsf{A}}(\Gamma)$$

Example 7. Recall from Example 4 that $W_{\mathsf{A}}(\Gamma_7) = 977$. Assume that A performs a further transaction to redeem all 10 units of (τ_0, τ_1) from her wallet. The resulting state is $\Gamma_8 = \mathsf{A}[100 : \tau_0, 53 : \tau_1] \mid \cdots$. We compute A's net worth in that state, using the oracle token prices: $W_{\mathsf{A}}(\Gamma_8) = 100 \cdot P_{(\tau_0)}(\Gamma_7) + 53 \cdot P_{(\tau_1)}(\Gamma_7) = 100 \cdot 5 + 53 \cdot 9 = 977$, as correctly predicted by Lemma 6. □

4 Properties of AMM incentives

We now study the incentive mechanisms of AMMs. We start in §4.1 by introducing a few notions of *exchange rate*, which are pivotal to understanding these mechanisms. In §4.2 we devise general conditions on swap invariants, overall named *incentive-consistency*, which guarantee that AMMs enjoy relevant economic properties. In §4.3 we study solutions to the *arbitrage problem*, which is the key to incentivize users to perform swap operations towards an ideal state where the AMM's exchange rates align with the exchange rates set by price oracles. Finally, in §4.4 we study the incentives to swap and deposit larger amounts.

4.1 Exchange rates

The ***exchange rate*** between two token types is the number of units of one token needed to buy one unit of the other token at the current price. We define *L*eft and *R*ight versions of this notion, that reflect the direction of the exchange:

$$XL_\Gamma(\tau_0, \tau_1) = P_{\tau_0}(\Gamma)/P_{\tau_1}(\Gamma) \qquad XR_\Gamma(\tau_0, \tau_1) = P_{\tau_1}(\Gamma)/P_{\tau_0}(\Gamma) \tag{11}$$

The ***swap rate*** between τ_0 and τ_1 upon a payment of $v_i : \tau_i$ (for $i \in \{0, 1\}$) is the ratio between v and v_i, where v is the received amount of τ_{1-i} resulting from a swap action on an AMM $(r_0 : \tau_0, r_1 : \tau_1)$. We first introduce an auxiliary notion, parameterized over the balances r_0 and r_1, instead of the token types:

$$\begin{aligned} XL^{\mathsf{swap}}_\phi(v_0, r_0, r_1) &= v/v_0 \quad \text{if } I(r_0, r_1) = I(r_0 + \phi v_0, r_1 - v) \\ XR^{\mathsf{swap}}_\phi(v_1, r_0, r_1) &= v/v_1 \quad \text{if } I(r_0, r_1) = I(r_0 - v, r_1 + \phi v_1) \end{aligned} \tag{12}$$

The swap rate is parameterized over the fee rate ϕ: the case where $\phi = 1$ represents an ideal scenario with no fees: in this case, we write just $XL^{\mathsf{swap}}(v_0, r_0, r_1)$. We define the swap rate in a state Γ such that $(r_0 : \tau_0, r_1 : \tau_1) \in \Gamma$ as follows:

$$XL^{\mathsf{swap}}_{\Gamma,\phi}(v_0, \tau_0, \tau_1) = XL^{\mathsf{swap}}_\phi(v_0, r_0, r_1) \qquad XR^{\mathsf{swap}}_{\Gamma,\phi}(v_1, \tau_0, \tau_1) = XR^{\mathsf{swap}}_\phi(v_1, r_0, r_1)$$

We also define the ***redeem rate***. The left version is:

$$XL^{\mathsf{rdm}}_\Gamma(\tau_0, \tau_1) = r_0/\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma) \qquad \text{if } (r_0 : \tau_0, r_1 : \tau_1) \in \Gamma \tag{13}$$

4.2 General properties of swap invariants

We now introduce a set of properties of swap invariants, called cumulatively ***incentive-consistency***, which overall incentivize users to interact with AMMs by performing swap and deposit actions.

Swap-rate continuity This property requires that, for all $r_0, r_1 > 0$:

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) \;=\; 1/\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}(\varepsilon, r_0, r_1) \in \mathbb{R}^+ \tag{14}$$

Figure 3 (left) illustrates this property, displaying the points (x, y) which satisfy the constant product invariant $x \cdot y = k$. The left swap rate limit for the constant product invariant and $\phi = 1$ is $\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) = r_1/r_0$, while for the right swap we have $\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}(\varepsilon, r_0, r_1) = r_0/r_1$. Coinciding left swap limit and right swap limit inverse are illustrated as the slope of the product constant curve at a selected point in Figure 3 (left). The constant product invariant satisfies (14), i.e. it is swap-rate continuous.

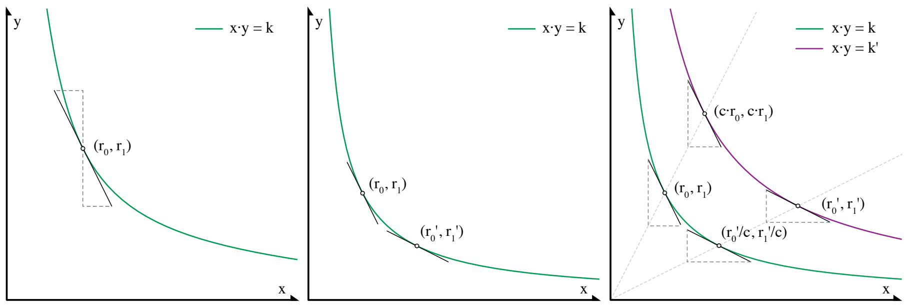


Fig. 3: The constant product invariant $I(x, y) = x \cdot y$ is swap-rate-consistent (left), demand-sensitive (center), non-depletable, funds-consistent (right) and swap-rate consistent (right).

Demand-sensitivity A swap invariant is demand-sensitive if the swap rate strictly decreases with demand. Formally, for all $r_0, r_1, r_0', r_1' > 0$:

$$I(r_0, r_1) = I(r_0', r_1') \land r_0' > r_0 \implies \lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0, r_1) \;>\; \lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0', r_1') \tag{15}$$

We implicitly require that (15) and the subsequent properties stated for the left version of an exchange rate also hold for the right version.

Figure 3 (center) depicts two points (r_0, r_1), (r_0', r_1') on the constant product curve, which satisfy $x \cdot y = k$ for identical k. For the constant product invariant, the left swap limit can be expressed as $\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) = \phi \cdot r_1/r_0$. For the given k and points in Figure 3 (center):

$$\lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0, r_1) = \phi \cdot k/r_0^2 \qquad \lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0', r_1') = \phi \cdot k/{r_0'}^2$$

Thus for $r_0' > r_0$, $\lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0, r_1) > \lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0', r_1')$: the constant product invariant is demand-sensitive.

Non-depletion This property ensures that the balance of tokens within an AMM cannot be zeroed via swaps. Formally, I is non-depletable when, for all $r_0, r_1 > 0$ and $r'_0, r'_1 \geq 0$:

$$I(r_0, r_1) = I(r'_0, r'_1) \implies r'_0, r'_1 \neq 0 \tag{16}$$

Note that the constant product invariant trivially satisfies this property.

Funds-consistency Deposits to an AMM ensure higher swap rates for a given input amount v, whereas redeems will reduce the swap rates for v. This behaviour is formalized later on in Theorem 4, but is a consequence of the funds-consistency property of the swap invariant. Formally, we require that for all $r_0, r_1, r'_0, r'_1 > 0$:

$$\begin{aligned} &I(r_0, r_1) \neq I(r'_0, r'_1) \iff \\ &\qquad \exists! c \in \mathbb{R}^+ \setminus \{1\} : I(c \cdot r'_0, c \cdot r'_1) = I(r_0, r_1) \ \wedge \ I(\tfrac{r_0}{c}, \tfrac{r_1}{c}) = I(r'_0, r'_1) \end{aligned} \tag{17}$$

Figure 3 (right) illustrates funds-consistency for the constant product invariant. Here, $r_0 \cdot r_1 = k \neq k' = r'_0 \cdot r'_1$. We observe that there exists a unique $c > 0$ in $(c \cdot r_0) \cdot (c \cdot r_1) = r'_0 \cdot r'_1 = k'$: namely, $c = \sqrt{(r'_0 \cdot r'_1)/(r_0 \cdot r_1)}$. Conversely, $(r'_0/c) \cdot (r'_1/c) = r_0 \cdot r_1$, which holds for the same value of c.

Swap-rate-consistency The design of AMMs aims to ensure that redeems and deposits do not interfere with the alignment of the swap rate towards the exchange rate. Since both deposits and redeems preserve the balance ratio of a token pair, we require swap rate limits for all balances of a given ratio to be constant. For all $r_0, r_1, c > 0$:

$$\lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, r_0, r_1) \ = \ \lim_{\varepsilon \to 0} XL_{\phi}^{\mathsf{swap}}(\varepsilon, c \cdot r_0, c \cdot r_1) \tag{18}$$

Figure 3 (right) illustrates equal swap rate limits for given r_0, r_1 and $c \cdot r_0, c \cdot r_1$. Here, $\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) = \phi \cdot r_1 / r_0 = \phi \cdot (c \cdot r_1)/(c \cdot r_0)$ for $c > 0$.

Finally, the following lemma establishes that the constant product swap invariant (the one used e.g. by Uniswap and Mooniswap) is indeed incentive-consistent. We conjecture that the same is true for the swap invariants implemented by the other mainstream AMM platforms.

Lemma 7. *The constant product swap invariant is incentive-consistent.*

4.3 The arbitrage game

We now study the incentive mechanisms of AMMs from a game-theoretic perspective. Indeed, AMMs can be seen as multi-player games where users collaborate or compete to achieve possibly conflicting goals. In such games the allowed moves of users are the interactions with other users and with AMMs, while their goal is typically to increase their net worth.

The ***arbitrage problem*** is an interesting example of an AMM game since it is directly linked to the incentive of swaps in a way that makes AMMs track exchange rates. The arbitrage problem has been formalized for specific swap invariants, namely the *weighted* and *constant product* swap invariant [17,19]. We generalize here the arbitrage problem to *arbitrary* swap invariants. We provide sufficient conditions for the existence of solutions, and we link the solutions to the expected relation between AMMs and exchange rates.

We model the arbitrage problem as a single-player, single-round game. The *initial game state* is $\Gamma_0 = \mathsf{A}[\sigma] \mid (r_1 : \tau_0, r_1 : \tau_1)$, where A is the only *player*. The *moves* of A are all the possible transactions fired by A; we also consider doing nothing as a possible move. The *goal* of A is to maximize her net worth, i.e. to maximize $W_{\mathsf{A}}(\Gamma) - W_{\mathsf{A}}(\Gamma_0)$, where Γ is the state resulting from executing the selected move. A *solution* to the game is a move that satisfies the goal, i.e. one of the optimal moves. We further assume that A holds no minted tokens containing (τ_0, τ_1) as a subterm (i.e., (τ_0, τ_1) itself, $((\tau_0, \tau_1), \tau_2)$, etc.). In this way, any change in A's net worth only depends on the exchange rate between τ_0 and τ_1, and on the transfer of value resulting from A's move.

Before presenting the solution to the game we examine the potential candidates for the solution. First, note that transfers are not valid solutions, as they can only decrease A's net worth. A second observation is that doing nothing, depositing or redeeming do not alter A's net worth (cf. Lemma 3). Hence, if one of such moves is a solution, so are the other two. The only moves that may affect A's net worth are swaps. For a swap to be a solution to the game, it must, first of all, result in a positive change of A's net worth. This happens when the swap rate is greater than the exchange rate. Theorem 2 presents the solution to the game. Note that if $\mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$ is a solution, then for all $v_1' \leq v_1$, also $\mathsf{swapL}(v_0 : \tau_0, v_1' : \tau_1)$ is a solution. Without loss of generality, our statement singles our the solution with the greatest v_1 (similarly for the right swap).

Theorem 2. *Let I be demand-sensitive and non-depletable, and let the initial state of the game be $\Gamma_0 = \mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1)$, with $r_0, r_1 > 0$. Let $\sigma(\tau_0)$, $\sigma(\tau_1)$ be large enough to enable any needed swap. Then, the solution to the game is:*

– $\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$ *if* $\Gamma_0 \xrightarrow{\mathsf{A}:\mathsf{swapL}(v_0:\tau_0, v_1:\tau_1)} \Gamma$*, and:*

$$(1)\ \lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma_0,\phi}(\varepsilon, \tau_0, \tau_1) > XL_{\Gamma_0}(\tau_0, \tau_1)$$

$$(2)\ \lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0 + \phi \cdot v_0, r_1 - v_1) = XL_{\Gamma}(\tau_0, \tau_1) \quad \textit{where } \exists!\, \delta :$$
$$I(r_0, r_1) = I(r_0 + \phi \cdot v_0, r_1 - v_1) = I(r_0 + \phi \cdot (v_0 + \varepsilon), r_1 - (v_1 + \delta))$$

– $\mathsf{A} : \mathsf{swapR}(v_0 : \tau_0, v_1 : \tau_1)$ *if* $\Gamma_0 \xrightarrow{\mathsf{A}:\mathsf{swapR}(v_0:\tau_0, v_1:\tau_1)} \Gamma$*, and:*

$$(1)\ \lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\Gamma_0,\phi}(\varepsilon, \tau_0, \tau_1) > XR_{\Gamma_0}(\tau_0, \tau_1)$$

$$(2)\ \lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\phi}(\varepsilon, r_0 - v_0, r_1 + \phi \cdot v_1) = XR_{\Gamma}(\tau_0, \tau_1) \quad \textit{where } \exists!\, \delta :$$
$$I(r_0, r_1) = I(r_0 - v_0, r_1 + \phi \cdot v_1) = I(r_0 - (v_0 + \delta), r_1 + \phi \cdot (v_1 + \varepsilon))$$

– *do nothing (or do any deposit or redeem), otherwise.*

Intuitively, condition (1) requires that the swap rate for infinitesimal amounts is greater than the exchange rate in the initial state; (2) requires that in the state Γ reached by performing the move of the solution, the swap rate for infinitesimal amounts tends to the exchange rate — thus achieving one of the main desiderata on AMMs. Note that Γ is an *equilibrium*: no move from there can improve A's net worth, i.e. doing nothing is a solution for the arbitrage problem in Γ.

Note that for the swapL/swapR solutions, the swapped amounts are unique: this is a consequence on the assumption (10). An implicit desideratum on these solutions is that, given a specific instance of the swap invariant, they are efficiently computable: this is the case, e.g., for the constant product invariant [17].

For $\phi = 1$ we can observe by inspection of (14) that the do-nothing solution for Theorem 2 only holds for:

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma_0}(\varepsilon, \tau_0, \tau_1) \;=\; 1/\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\Gamma_0}(\varepsilon, \tau_0, \tau_1) \;=\; XL_{\Gamma_0}(\tau_0, \tau_1) \tag{19}$$

Thus, the solution to the game results in the AMM tracking global exchange rates precisely: any infinitesimal deviation of the global exchange rate implies a swap action in the arbitrage game.

The assumption that the players' wallets are sufficiently large is common in formulations of the arbitrage problem. We note that any rational agent is incentivized to perform such a swap: the optimal solution to the arbitrage game can thus be approximated by multiple users exchanging smaller swap amounts. Furthermore, the availability of flash-loans [28, 29] can provide up-front funds, and thus significantly reduce the balance requirements for arbitrage swaps.

Finally, we prove that a AMM deposits and redeems do not affect the solution type of the arbitrage game. If the arbitrage solution prior to a deposit or redeem is swapL, swapR or nothing, the arbitrage solution in the subsequent state should remain of the same type.

Theorem 3. *Let I be incentive-consistent. Let $(r_0 : \tau_0, r_1 : \tau_1) \in \Gamma$ with $r_0, r_1 > 0$. If $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$, type$(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$, then the arbitrage solutions in Γ and Γ' will have the same type or both be nothing.*

In other words, the design of AMMs aims to ensure that deposits and redeems do not interfere with the alignment of the swap rate towards the exchange rate.

Example 8. Consider the arbitrage game with player B and initial state $\Gamma_7 = \mathsf{B}[0 : \tau_0, 27 : \tau_1] \mid (18 : \tau_0, 6 : \tau_1) \mid \cdots$ resulting after the last step in Figure 1. Assuming the constant product invariant and no fees (i.e., $\phi = 1$), we have that:

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma_7}(\varepsilon, \tau_0, \tau_1) \;=\; r_1/r_0 \;=\; 6/18 \;<\; 5/9 \;=\; XL_{\Gamma_7}(\tau_0, \tau_1)$$

$$\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\Gamma_7}(\varepsilon, \tau_0, \tau_1) \;=\; r_0/r_1 \;=\; 18/6 \;>\; 9/5 \;=\; XR_{\Gamma_7}(\tau_0, \tau_1)$$

Hence, by Theorem 2 it follows that the optimal move is $\mathsf{swapR}(v_0 : \tau_0, v_1 : \tau_1)$, for suitable v_0 and v_1. To find these values, we must solve for v_0 and v_1 the equations in item (2) of Theorem 2, i.e.:

$$\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\phi}(\varepsilon, r_0 - v, r_1 + v_1) = XR_{\Gamma}(\tau_0, \tau_1) \qquad I(r_0, r_1) = I(r_0 - v_0, r_1 + v_1)$$

Solving these equations gives:

$$v_1 = \sqrt{\frac{5}{9} \cdot r_0 r_1} - r_1 \approx 1.74 \qquad v_0 = \frac{r_0 v_1}{r_1 + v_1} \approx 4$$

By performing $\mathsf{swapR}(v_0 : \tau_0, v_1 : \tau_1)$ with these values from Γ_7, we obtain:

$$\Gamma \;=\; \mathsf{B}[4 : \tau_0, 25.26 : \tau_1] \mid (14 : \tau_0, 7.74 : \tau_1) \mid \cdots$$

This action maximizes B's net worth: indeed, we have $W_{\mathsf{B}}(\Gamma_7) = 243$ and $W_{\mathsf{B}}(\Gamma) = 247.6$; any other action will result in a lower net worth for B. □

4.4 Incentivizing deposits and swaps

Theorem 2 ensures that incentive-consistent AMMs incentivize swaps to align to exchange rates. We now show that, under certain conditions, deposits and swaps incentivize each other. The intuition is that larger amounts of tokens in an AMM provide better swap rates, therefore attracting users interested in swaps. These swaps, in turn, result in increased redeem rates, making the AMM attractive for further deposits. Note that this behaviour relies on an underlying assumption of our model, i.e. that exchange rates are stable: oracle prices are fixed. In the wild, exchange rates can vary over time, possibly making the net worth of users holding minted AMM tokens decrease: this phenomenon is commonly referred to as *impermanent loss* [9].

The following theorem shows that deposits increase swap rates, hence incentivizing swaps, whilst redeems have the opposite effect.

Theorem 4. *Let I be incentive-consistent. Let $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$, with $r_0, r_1 > 0$, and let $\Gamma \xrightarrow{\mathsf{A}:\ell} \Gamma'$. Then, for all $v \in \mathbb{R}^+$:*

$$\begin{array}{l} XL^{\mathsf{swap}}_{\Gamma,\phi}(v, \tau_0, \tau_1) \circ XL^{\mathsf{swap}}_{\Gamma',\phi}(v, \tau_0, \tau_1) \\ XR^{\mathsf{swap}}_{\Gamma,\phi}(v, \tau_0, \tau_1) \circ XR^{\mathsf{swap}}_{\Gamma',\phi}(v, \tau_0, \tau_1) \end{array} \quad \textit{where } \circ = \begin{cases} < & \textit{if } \ell = \mathsf{dep}(\_ : \tau_0, \_ : \tau_1) \\ > & \textit{if } \ell = \mathsf{rdm}(\_ : (\tau_0, \tau_1)) \end{cases}$$

We now show that, under certain conditions, swaps incentivize deposits. Intuitively, swaps contribute to higher redeem rates, which increase the net wealth of the holders of minted AMM tokens:

Theorem 5. *Let I be incentive-consistent. Let $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$ and $\Gamma \to^* \Gamma'$, where $\Gamma' = (r_0' : \tau_0, r_1' : \tau_1) \mid \cdots$. If $r_1/r_0 = r_1'/r_0'$ then:*

$$XL^{\mathsf{rdm}}_{\Gamma}(\tau_0, \tau_1) \leq XL^{\mathsf{rdm}}_{\Gamma'}(\tau_0, \tau_1) \qquad XR^{\mathsf{rdm}}_{\Gamma}(\tau_0, \tau_1) \leq XR^{\mathsf{rdm}}_{\Gamma'}(\tau_0, \tau_1)$$

Recall that a user who deposits into an AMM $(r_0 : \tau_0, r_1 : \tau_1)$ in state Γ receives in return an amount of minted tokens. A consequence of Theorem 5 is that these minted tokens can be redeemed with a higher redeem rate in any subsequent state Γ' which preserves the funds ratio r_1/r_0. Note that swaps are the only actions that may affect the redeem rate along the run $\Gamma \to^* \Gamma'$.

Therefore, performing swaps that eventually re-align the funds ratio to r_1/r_0 incentivizes deposits.

The condition of constant funds ratio in Theorem 5 is practically relevant. For instance, for stable exchange rates, such as in the case of exchanges between stable coins [6], the arbitrage game ensures stable fund ratios: users are hence incentivized to provide funds, as the redeem rate is likely to increase over time.

5 Related Work

To the best of our knowledge, our work is the first to study AMMs abstracting from the swap invariant. All works in literature consider concrete swap invariants; most of them focus on the constant product, popularized by Uniswap [13]. The arbitrage problem for constant-product swap invariants has been formalized in [17,19], which show that the solution can be efficiently computed, and suggest that constant product AMMs accurately tend towards exchange rates. Our work generalizes such results. Furthermore, as we have shown in §4.3 (19), the fee-rate ϕ determines how much AMMs deviate from global exchange rates: higher fees, however, also result in reduced swap amounts in arbitrage actions, negatively affecting fee accrual. In [24], the optimal fee-rate that maximizes the fee accrual for the depositing user is analytically derived.

A executable model of Uniswap [13] has been specified in [1] to analyze integer rounding errors in the Uniswap implementation.

A few alternatives to the constant product invariant have been proposed. Curve features a peculiar invariant [22] optimized for large swap volumes between *stable coins*, where the swap rate can support large amounts with small sensitivity. To efficiently compute swap invariant, implementations perform numerical approximations [15]. Should these approximations fail to converge, these implementations still guarantee that the AMM remains liquid. We conjecture that the invariants in [3, 22] are incentive-consistent. The work [25] proposes a constant product invariant that is adjusted dynamically based on the oracle price feed, thus reducing the need for arbitrage transactions, but at the cost of lower fee accrual. AMMs with *virtual* balances have been proposed [2] and implemented [7,8]. In these AMMs, the swap rate depends on past actions, besides the current funds balances in the AMM. This, similarly to [25], aims to minimize the need for arbitrage transactions to ensure the local AMM swap rate tends towards the exchange rates.

Some implementations [3] generalise AMM pairs to n-tokens, allowing users to swap any non-intersecting sets of token types. For example, the constant-product invariant becomes $I(r_0, \ldots, r_n) = r_0^{w_0} \cdot \ldots \cdot r_n^{w_n}$ where $\sum_{i=0}^{n} w^i = 1$.

6 Conclusions

We have proposed a theory of AMMs, featuring a model of their behaviour and a formally proven set of fundamental properties, characterizing both structural and economic aspects. Our theory is parametric w.r.t. platform-specific features (e.g., swap invariants), and it abstracts from implementation-specific features, and from the features that are orthogonal to the core functionality of AMMs (e.g., governance).

There are some differences between our model and the existing AMM platforms. Uniswap implements flash-loans as part of the swap actions: namely, the user can optionally borrow available pair funds [10] whilst returning these within the same *atomic group* of actions. Further, Uniswap implements an exchange rate oracle, allowing smart contracts to interpret (averages of) recent swap rates as exchange rates [11]. Balancer [3] extends token pairs to token *tuples*: a user can swap any two non-coinciding sets of supported tokens, such that the swap invariant is maintained. In all AMM implementations, token balances are represented as integers: consequently, they are subject to rounding errors [1]. AMM platforms frequently implement a governance logic, which allow "governance token" holders to coordinate changes to AMM fee-rates or swap invariant parameters.

AMM platforms like Uniswap [13] and Curve [22] have overtaken centralized cryptocurrency markets in size and usage. On the one hand, a better understanding of AMM design in cases where AMMs host the majority of the token's global swap volume is critical [18]. It would be interesting to investigate how our theory can be used to formally explain such behaviours. On the other hand, the growth of AMMs is making them more attractive for malicious users. Current research efforts [21, 23, 27, 30] are devoted to understanding vulnerabilities and attacks, which we plan to investigate formally, exploiting our theory.

This paper, together with our work on formalizing another DeFi archetype called *lending pool* [20], is the first step towards a general theory of DeFi. We believe that a general theory encompassing interactions between different DeFi archetypes is crucial to be able to reason about their structural, economic and security aspects, as typical DeFi applications operate within a wider ecosystem, composed by a set of collaborating or competing agents, which interact through possibly separate execution environments.

Acknowledgements Massimo Bartoletti is partially supported by Conv. Fondazione di Sardegna & Atenei Sardi project F74I19000900007 *ADAM*. James Hsin-yu Chiang is supported by the PhD School of DTU Compute. Alberto Lluch Lafuente is partially supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu).

References

1. Formal specification of constant product market maker model & implementation (2018), https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf
2. Improving frontrunning resistance of x*y=k market makers (2018), https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281
3. Balancer whitepaper (2019), https://balancer.finance/whitepaper/
4. Curve statistics (2020), https://www.curve.fi/dailystats

5. Curve website (2020), https://www.curve.fi
6. Makerdao website (2020), https://https://makerdao.com
7. Mooniswap implementation (2020), https://github.com/1inch-exchange/mooniswap/blob/02dccfab2ddbb8a409400288cb13441763370350/contracts/Mooniswap.sol
8. Mooniswap whitepaper (2020), https://mooniswap.exchange/docs/MooniswapWhitePaper-v1.0.pdf
9. Uniswap Documentation: Understanding Returns (2020), https://uniswap.org/docs/v2/advanced-topics/understanding-returns/
10. Uniswap flash loan implementation (2020), https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol#L172
11. Uniswap oracle template (2020), https://github.com/Uniswap/uniswap-v2-periphery/blob/dda62473e2da448bc9cb8f4514dadda4aeede5f4/contracts/examples/ExampleOracleSimple.sol
12. Uniswap statistics (2020), https://info.uniswap.org
13. Uniswap token pair implementation (2020), https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol
14. Uniswap website (2020), https://www.uniswap.org
15. Curve computation of invariant constant (2021), https://github.com/curvefi/curve-contract/blob/a1b5a797790d3f5ef12b0e358892a0ce47c12f85/contracts/pool-templates/base/SwapTemplateBase.vy#L206
16. Curve token pair implementation (2021), https://github.com/curvefi/curve-contract/blob/a1b5a797790d3f5ef12b0e358892a0ce47c12f85/contracts/pool-templates/base/SwapTemplateBase.vy
17. Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. In: ACM Conference on Advances in Financial Technologies (AFT). pp. 80–91. ACM (2020). https://doi.org/10.1145/3419614.3423251, https://arxiv.org/abs/2003.10001
18. Angeris, G., Evans, A., Chitra, T.: When does the tail wag the dog? curvature and market making. arXiv preprint arXiv:2012.08040 (2020), https://arxiv.org/abs/2012.08040
19. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of Uniswap markets. Cryptoeconomic Systems Journal (2019), https://ssrn.com/abstract=3602203
20. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: SoK: Lending pools in decentralized finance. In: Workshop on Trusted Smart Contracts. LNCS, Springer (2021), (To appear)
21. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symposium on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
22. Egorov, M.: Stableswap - efficient mechanism for stablecoin (2019), https://www.curve.fi/stableswap-paper.pdf
23. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13
24. Evans, A., Angeris, G., Chitra, T.: Optimal fees for geometric mean market makers (2021), https://web.stanford.edu/~guillean/papers/g3m-optimal-fee.pdf

25. Krishnamachari, B., Feng, Q., Grippo, E.: Dynamic curves for decentralized autonomous cryptocurrency exchanges. arXiv preprint arXiv:2101.02778 (2021), https://arxiv.org/abs/2101.02778
26. Mazurkiewicz, A.W.: Basic notions of trace theory. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. LNCS, vol. 354, pp. 285–363. Springer (1988). https://doi.org/10.1007/BFb0013025
27. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? (2021), https://arxiv.org/abs/2101.05511
28. Qin, K., Zhou, L., Livshits, B., Gervais: Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2003.03810
29. Wang, D., Wu, S., Lin, Z., Wu, L., Yuan, X., Zhou, Y., Wang, H., Ren, K.: Towards understanding flash loan and its applications in defi ecosystem. arXiv preprint arXiv:2010.12252 (2020), https://arxiv.org/abs/2010.12252
30. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. arXiv preprint arXiv:2009.14021 (2020), https://arxiv.org/abs/2009.14021

A Proofs

Proof of Lemma 1 Straightforward inspection of the rules in §2. □

Proof of Lemma 2 We proceed by cases on the rule used in the last transition: it is straightforward to check that, in all the rules, changes applied to initial tokens cancel out. □

Proof of Lemma 3 We first prove that the net worth of a user A, i.e. $W_{\mathsf{A}}(\Gamma)$, is preserved by transactions of type dep and rdm, or by transactions which do not affect A's wallet. More precisely, we prove that:

$$\Gamma \xrightarrow{\mathsf{T}} \Gamma' \ \wedge \ (\mathit{type}(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\} \vee \mathsf{A} \not\in \mathit{wal}(\mathsf{T})) \implies W_{\mathsf{A}}(\Gamma) = W_{\mathsf{A}}(\Gamma') \quad (20)$$

We have the following cases:

– [Xfer]. Let $\mathsf{C} : \mathsf{xfer}(\mathsf{B}, v : \tau)$ be the fired transaction. By hypothesis, it must be $\mathsf{A} \not\in \mathit{wal}(\mathsf{T}) = \{\mathsf{B}, \mathsf{C}\}$, hence the thesis follows by inspection of the rule.

– [Dep0]. Let $\mathsf{B} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ be the fired transaction. We have that:

$$\begin{aligned} \Gamma &= \mathsf{B}[\sigma] \mid \Gamma_0 \\ \Gamma' &= \mathsf{B}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : (\tau_0, \tau_1)] \mid (v_0 : \tau_0, v_1 : \tau_1) \mid \Gamma_0 \end{aligned}$$

If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected. Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned} W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) - v_0 P_{\tau_0}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + v_0 P_{(\tau_0,\tau_1)}(\Gamma) \\ &= W_{\mathsf{A}}(\Gamma) - v_0 P_{\tau_0}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + v_0 \frac{v_0 P_{\tau_0}(\Gamma) + v_1 P_{\tau_1}(\Gamma)}{v_0} \\ &= W_{\mathsf{A}}(\Gamma) \end{aligned}$$

– [Dep]. Let $\mathsf{B} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ be the fired transaction. We have that:

$$\begin{aligned} \Gamma &= \mathsf{B}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma_0 \\ \Gamma' &= \mathsf{B}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : (\tau_0, \tau_1)] \mid (r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1) \mid \Gamma_0 \end{aligned}$$

where $v = (v_0 \cdot s)/r_0)$, with $s = \mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)$. If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected. Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned} W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) - v_0 P_{\tau_0}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + v P_{(\tau_0,\tau_1)}(\Gamma) \\ &= W_{\mathsf{A}}(\Gamma) - v_0 P_{\tau_0}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + v \frac{r_0 P_{\tau_0}(\Gamma) + r_1 P_{\tau_1}(\Gamma)}{s} \\ &= W_{\mathsf{A}}(\Gamma) - v_0 P_{\tau_0}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + \frac{v_0}{r_0}\Big(r_0 P_{\tau_0}(\Gamma) + r_1 P_{\tau_1}(\Gamma)\Big) \\ &= W_{\mathsf{A}}(\Gamma) - v_1 P_{\tau_1}(\Gamma) + \frac{v_0}{r_0} r_1 P_{\tau_1}(\Gamma) \\ &= W_{\mathsf{A}}(\Gamma) \end{aligned}$$

where the last equality follows from the premise $r_1 v_0 = r_0 v_1$ of rule [Dep].

– [SwapL]. Let $\mathsf{B} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$ be the fired transaction. By hypothesis, it must be $\mathsf{A} \not\in \mathit{wal}(\mathsf{T}) = \{\mathsf{B}\}$. The thesis follows by inspection of the rule.
– [SwapR]. Similar to the previous case.
– [Rdm]. Let $\mathsf{B} : \mathsf{rdm}(v : (\tau_0, \tau_1))$ be the fired transaction. We have that:

$$\begin{aligned}\Gamma &= \mathsf{B}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma_0\\ \Gamma' &= \mathsf{B}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : (\tau_0, \tau_1)] \mid (r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1) \mid \Gamma_0\end{aligned}$$

where $v_0 = (vr_0)/s$ and $v_1 = (vr_1)/s$, with $s = \mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)$. If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected. Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned}W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) + v_0 P_{\tau_0}(\Gamma) + v_1 P_{\tau_1}(\Gamma) - v P_{(\tau_0,\tau_1)}(\Gamma)\\ &= W_{\mathsf{A}}(\Gamma) + v_0 P_{\tau_0}(\Gamma) + v_1 P_{\tau_1}(\Gamma) - \frac{vr_0}{s} P_{\tau_0}(\Gamma) - \frac{vr_1}{s} P_{\tau_1}(\Gamma)\\ &= W_{\mathsf{A}}(\Gamma) + \frac{vr_0}{s} P_{\tau_0}(\Gamma) + \frac{vr_1}{s} P_{\tau_1}(\Gamma) - \frac{vr_0}{s} P_{\tau_0}(\Gamma) - \frac{vr_1}{s} P_{\tau_1}(\Gamma)\\ &= W_{\mathsf{A}}(\Gamma) \qquad \square\end{aligned}$$

We now prove that the global net worth is preserved by *any* transactions. First, we recall the definition of global net worth. Let:

$$\Gamma = \mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid (r_1 : \tau_1, r'_1 : \tau'_1) \mid \cdots \mid (r_k : \tau_k, r'_k : \tau'_k)$$

The global net worth of Γ is defined as:

$$W(\Gamma) \;=\; \textstyle\sum_{i \in 1..n} W_{\mathsf{A}_i}(\Gamma) \tag{21}$$

We prove that:

$$\Gamma \xrightarrow{\mathsf{T}} \Gamma' \implies W(\Gamma) = W(\Gamma') \tag{22}$$

We have the following cases:

– [Xfer]. A xfer transaction subtracts an amount of tokens from the wallet of a user, and adds an equal amount of the same token type to the wallet of another user. Therefore, it preserves the global net worth.
– [Dep0], [Dep], [Rdm]. These rules affect the number of tokens in AMMs, which does not contribute to the global net worth, and the balances of users, which are preserved (20). Therefore, the global net worth is preserved.
– [SwapL]. We have that:

$$\begin{aligned}\Gamma &= \mathsf{A}[\sigma] \;\mid\; (r_0 : \tau_0, r_1 : \tau_1) \;\mid\; \Gamma_0\\ \Gamma' &= \mathsf{A}[\sigma - v_0 : \tau_0 + v : \tau_1] \;\mid\; (r_0 + v_0 : \tau_0, r_1 - v : \tau_1) \;\mid\; \Gamma_0\end{aligned}$$

Note that $sply_{\tau_0}(\Gamma') = sply_{\tau_0}(\Gamma)$, and $sply_{\tau}(\Gamma') = sply_{\tau}(\Gamma)$. By (8), we have that $P_{\tau_0}(\Gamma') = P_{\tau_0}(\Gamma)$ Therefore:

$$\begin{aligned}
W(\Gamma') &= W(\Gamma_0) - v_0 P_{\tau_0}(\Gamma') + v P_{\tau_1}(\Gamma') \\
&\quad - sply_{(\tau_0,\tau_1)}(\Gamma) P_{(\tau_0,\tau_1)}(\Gamma) + sply_{(\tau_0,\tau_1)}(\Gamma') P_{(\tau_0,\tau_1)}(\Gamma') \\
&= W(\Gamma_0) - v_0 P_{\tau_0}(\Gamma) + v P_{\tau_1}(\Gamma) \\
&\quad - sply_{(\tau_0,\tau_1)}(\Gamma) \cdot \big(P_{(\tau_0,\tau_1)}(\Gamma) + P_{(\tau_0,\tau_1)}(\Gamma')\big) \\
&= W(\Gamma_0) - v_0 P_{\tau_0}(\Gamma) + v P_{\tau_1}(\Gamma) \\
&\quad - r_0 P_{\tau_0}(\Gamma) + r_1 P_{\tau_1}(\Gamma) + (r_0 + v_0) P_{\tau_0}(\Gamma) + (r_1 - v) P_{\tau_1}(\Gamma) \\
&= W(\Gamma)
\end{aligned}$$

– [SwapR]. Similar to the previous case. □

Proof of Lemma 4 For item (a) we proceed by induction on the length of the computation $\Gamma_0 \to^* \Gamma$, with Γ_0 initial. The base case is trivial, since Γ_0 contains no AMMs. For the inductive step, we must show that (a) is preserved by single transitions. Assume that Γ satisfies (a), and that $\Gamma \to \Gamma'$. Assume that Γ' contains an AMM $(r'_0 : \tau_0, r'_1 : \tau_1)$ with $r'_0 + r'_1 > 0$, i.e. $r'_0 > 0$ or $r'_1 > 0$. There are the following cases cases, depending on the rule used to infer $\Gamma \to \Gamma'$:

– [Xfer], [SwapL], [SwapR]. trivial, because these actions do not affect the supply of minted tokens.
– [Dep0], [Dep]. trivial, because deposits increase the supply of minted tokens.
– [Rdm]. Assume that the state of the AMM in Γ is $(r_0 : \tau_0, r_1 : \tau_1)$. By contradiction, suppose that the [Rdm] action burns all the $v = sply_{(\tau_0,\tau_1)}(\Gamma)$ units of the minted token. The rule premise requires $v > 0$, and it implies:

$$r'_0 = r_0 - v\frac{r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)} = 0 \qquad r'_1 = r_1 - v\frac{r_1}{sply_{(\tau_0,\tau_1)}(\Gamma)} = 0$$

Therefore, we would have $r'_0 = r'_1 = 0$ — contradiction.

For item (b), we proceed by induction on the number of AMMs in the state. The base case is trivial, since the premise of (b) is false. For the inductive step, let $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma''$ be a reachable state such that $r_0 + r_1 > 0$. Let $r'_0 \leq r_0$, and let $v = sply_{(\tau_0,\tau_1)}(\Gamma)$. By item (a) we have that $v > 0$. By the induction hypothesis, there exists a sequence of transactions $\Gamma'' \to^* \Gamma_1$ where users can redeem $v : (\tau_0, \tau_1)$ from other AMMs. From Γ_1, users can perform another sequence of transactions $\Gamma_1 \to^* \Gamma_2$ to transfer these units to the wallet of a user, say A. Summing up, we have that $\Gamma \to^* (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma_2$. Now, let:

$$\tilde{v} = \frac{(r_0 - r'_0)}{r_0} sply_{(\tau_0,\tau_1)}(\Gamma)$$

Since $\tilde{v} < v$, A can fire $\mathsf{A} : \mathsf{rdm}(\tilde{v} : (\tau_0, \tau_1))$, obtaining, for some $r_1' \leq r_1$:

$$
\begin{aligned}
(r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma_2 \rightarrow \Gamma' &= \left(r_0 - \tilde{v}\frac{r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)} : \tau_0, r_1' : \tau_1\right) \mid \cdots \\
&= (r_0' : \tau_0, r_1' : \tau_1) \mid \cdots
\end{aligned}
$$

The proof of item (c) is similar. □

Proof of Lemma 5 Let $\Gamma \xrightarrow{\mathsf{T}_0\mathsf{T}_1} \Gamma_{01}$ and $\Gamma \xrightarrow{\mathsf{T}_1\mathsf{T}_0} \Gamma_{10}$ with the following intermediate states:

$$
\Gamma \xrightarrow{\mathsf{T}_0} \Gamma_0 \xrightarrow{\mathsf{T}_1} \Gamma_{01} \qquad \Gamma \xrightarrow{\mathsf{T}_1} \Gamma_1 \xrightarrow{\mathsf{T}_0} \Gamma_{10}
$$

We have the following exhaustive cases on the type of the two transactions:

1. $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{xfer}(\mathsf{B}_0, v_0 : \tau_0)$.
   (a) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{xfer}(\mathsf{B}_1, v_1 : \tau_1)$. Straightforward from rule [Xfer].
   (b) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{dep}(v_1 : \tau_1, v_1' : \tau_1')$. We have that:

$$
\begin{aligned}
\Gamma &= \mathsf{A}_0[\sigma_{\mathsf{A}_0}] \mid \mathsf{B}_0[\sigma_{\mathsf{B}_0}] \mid \Gamma' \\
\Gamma_0 &= \mathsf{A}_0[\sigma_{\mathsf{A}_0} - v_0 : \tau_0] \mid \mathsf{B}_0[\sigma_{\mathsf{B}_0} + v_0 : \tau_0] \mid \Gamma'
\end{aligned}
$$

   If $\mathsf{A}_1 \neq \mathsf{A}_0, \mathsf{B}_0$, then the two transactions affect different part of the state, so the thesis follows trivially. Otherwise, we have two subcases, depending on whether [Dep0] or [Dep] was used.
   If [Dep0] was used, then:

$$
\Gamma_1 = \mathsf{A}_1[\sigma_{\mathsf{A}_1} - v_1 : \tau_1 - v_1' : \tau_1' + v_1 : (\tau_1, \tau_1')] \mid (v_1 : \tau_1, v_1' : \tau_1') \mid \Gamma'
$$

   If $\mathsf{A}_1 = \mathsf{A}_0$, then $\sigma_{\mathsf{A}_0} = \sigma_{\mathsf{A}_1}$, and the thesis $\Gamma_{01} = \Gamma_{10}$ follows by:

$$
\begin{aligned}
\Gamma_{01} = {} & \mathsf{A}_0[\sigma_{\mathsf{A}_0} - v_0 : \tau_0 - v_1 : \tau_1 - v_1' : \tau_1' + v_1 : (\tau_1, \tau_1')] \mid \\
& \mathsf{B}_0[\sigma_{\mathsf{B}_0} + v_0 : \tau_0] \mid (v_1 : \tau_1, v_1' : \tau_1') \mid \Gamma' \\
\Gamma_{10} = {} & \mathsf{A}_0[\sigma_{\mathsf{A}_0} - v_1 : \tau_1 - v_1' : \tau_1' + v_1 : (\tau_1, \tau_1') - v_0 : \tau_0] \mid \\
& \mathsf{B}_0[\sigma_{\mathsf{B}_0} + v_0 : \tau_0] \mid (v_1 : \tau_1, v_1' : \tau_1') \mid \Gamma'
\end{aligned}
$$

   If $\mathsf{A}_1 = \mathsf{B}_0$, then $\sigma_{\mathsf{B}_0} = \sigma_{\mathsf{A}_1}$, and the thesis $\Gamma_{01} = \Gamma_{10}$ follows by:

$$
\begin{aligned}
\Gamma_{01} = {} & \mathsf{A}_0[\sigma_{\mathsf{A}_0} - v_0 : \tau_0] \mid \\
& \mathsf{B}_0[\sigma_{\mathsf{B}_0} + v_0 : \tau_0 - v_1 : \tau_1 - v_1' : \tau_1' + v_1 : (\tau_1, \tau_1')] \mid (v_1 : \tau_1, v_1' : \tau_1') \mid \Gamma' \\
\Gamma_{10} = {} & \mathsf{A}_0[\sigma_{\mathsf{A}_0} - v_0 : \tau_0] \mid \\
& \mathsf{B}_0[\sigma_{\mathsf{B}_0} - v_1 : \tau_1 - v_1' : \tau_1' + v_1 : (\tau_1, \tau_1') + v_0 : \tau_0] \mid (v_1 : \tau_1, v_1' : \tau_1') \mid \Gamma'
\end{aligned}
$$

   The case where [Dep] was used is similar. In the following cases we omit the detailed computation of the states as done here, and we appeal the intuition to argue for the equivalence of the reached states.

   (c) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapL}(v_1 : \tau_1, v_1' : \tau_1')$. The equality of Γ_{01} and Γ_{10} follows from the fact that neither the updated users' balances nor the amounts which update the AMM depend on the current state. In particular, the swap invariant only depends on the values r_1 and r_1'.
   (d) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapR}(v_1 : \tau_1, v_1' : \tau_1')$. Similar to the previous case.
   (e) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{rdm}(v_1 : \tau_1)$. The thesis follows from the fact that the only part of the state which affects the update of the users' balances and the AMM is the supply of the minted token, which is not affected by T_0.
2. $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{dep}(v_0 : \tau_0, v_0' : \tau_0')$.
   (a) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{xfer}(\mathsf{B}_1, v_1 : \tau_1)$. Symmetric to case (1b).
   (b) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{dep}(v_1 : \tau_1, v_1' : \tau_1)$. If $(\tau_0, \tau_0') \neq (\tau_1, \tau_1')$, then the thesis is straightforward. Otherwise, note that the amount of tokens minted by the AMM only depends on the ratio between the amounts of τ_0 and τ_0' initially held by the AMM, and the values subsequently transferred to the AMM by T_0 and T_1.
   (c) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapL}(v_1 : \tau_1, v_1' : \tau_1')$. By hypothesis, $\{\tau_1, \tau_1'\}$ and $\{\tau_0, \tau_0'\}$ are disjoint. The thesis is straightforward, because AMMs operating on distinct token pairs do not interfere.
   (d) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapR}(v_1 : \tau_1, v_1' : \tau_1')$. Similar to the previous case.
   (e) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{rdm}(v_1 : \tau_1)$. If $(\tau_0, \tau_0') \neq (\tau_1, \tau_1')$, then the thesis is straightforward. Otherwise, note that the amount of tokens minted by the AMM in response of T_0 only depends on the ratio between the amounts of τ_0 and τ_0' initially held by the AMM, and the values transferred to the AMM by T_0. Similarly, the tokens paid by the AMM in response of T_1 only depend on the ratio between the amounts of τ_0 and τ_0' initially held by the AMM, which are constrained to preserve the ratio, and the amount deposited by T_0. Then, the two transactions are concurrent.
3. $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{rdm}(v_0 : \tau_0)$.
   (a) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{xfer}(\mathsf{B}_1, v_1 : \tau_1)$. Symmetric to case (1e).
   (b) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{dep}(v_1 : \tau_1, v_1' : \tau_1')$. Symmetric to case (2e).
   (c) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapL}(v_1 : \tau_1, v_1' : \tau_1')$. By hypothesis, $\{\tau_1, \tau_1'\}$ and $\{\tau_0, \tau_0'\}$ are disjoint. The thesis is straightforward, because AMMs operating on distinct token pairs do not interfere.
   (d) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapR}(v_1 : \tau_1, v_1' : \tau_1')$. Similar to the previous case.
   (e) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{rdm}(v_1 : \tau_1)$. The tokens paid by the AMM in response of T_0 and T_1 only depend on the ratio between the amounts of τ_0 and τ_0' initially held by the AMM, which are constrained to preserve the ratio.
4. $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{swapL}(v_0 : \tau_0, v_0' : \tau_0')$. The only case not covered by the previous one is when $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swapL}(v_1 : \tau_1, v_1' : \tau_1')$. By hypothesis, $\{\tau_1, \tau_1'\}$ and $\{\tau_0, \tau_0'\}$ are disjoint. The thesis is straightforward, because AMMs operating on distinct token pairs do not interfere.
5. $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{swapR}(v_0 : \tau_0, v_0' : \tau_0')$. Similar to the previous case. □

Lemma A.1 $\lambda \sim \lambda' \implies \forall \lambda_0, \lambda_1 : \lambda_0 \lambda \lambda_1 \sim \lambda_0 \lambda' \lambda_1$.

Proof. Direct from the fact that semantics of transactions is a function (Lemma 1), and it only depends on the states after the execution of λ and λ', which are equal starting from any state Γ, since $\lambda \sim \lambda'$.

Proof of Theorem 1 By definition, $\sim_{\#}$ is the least equivalence relation closed under the following rules (where ε denotes the empty sequence):

$$\frac{}{\varepsilon \sim_{\#} \varepsilon}[0] \qquad \frac{}{\mathsf{T} \sim_{\#} \mathsf{T}}[1] \qquad \frac{\mathsf{T}\#\mathsf{T}'}{\mathsf{T}\mathsf{T}' \sim_{\#} \mathsf{T}'\mathsf{T}}[2] \qquad \frac{\lambda_0 \sim_{\#} \lambda_0' \quad \lambda_1 \sim_{\#} \lambda_1'}{\lambda_0\lambda_1 \sim_{\#} \lambda_0'\lambda_1'}[3]$$

Let $\lambda \sim_{\#} \lambda'$. We have to show $\lambda \sim \lambda'$. We proceed by induction on the rules above. For rules [0] and [1], the thesis follows by reflexivity, since $\sim$ is an equivalence relation. For rule [2], the thesis follows immediately by definition of the concurrency relation $\#$. For rule [3], first note that $\lambda = \lambda_0\lambda_1$ and $\lambda' = \lambda_0'\lambda_1'$. By the induction hypothesis it follows that:

$$\lambda_0 \sim \lambda_0' \quad \text{and} \quad \lambda_1 \sim \lambda_1'$$

Therefore, by two applications of Lemma A.1:

$$\lambda = \lambda_0\lambda_1 \sim \lambda_0\lambda_1' \sim \lambda_0'\lambda_1' = \lambda'$$ □

Proof of Lemma 6 Define the *rank* $\|\tau\|$ of a token τ inductively as follows:

$$\|\tau\| \;=\; \begin{cases} 0 & \text{if } \tau \in \mathbb{T}_0 \\ \max\{\|\tau_0\|, \|\tau_1\|\} & \text{if } \tau = (\tau_0, \tau_1) \end{cases}$$

and accordingly define the rank of A in Γ as:

$$\|\Gamma\|_{\mathsf{A}} \;=\; \begin{cases} \max\{\|\tau\| \mid \tau \in \operatorname{dom}\sigma\} & \text{if } \mathsf{A}[\sigma] \in \Gamma \\ 0 & \text{otherwise} \end{cases}$$

We proceed by induction on $n = \|\Gamma\|_{\mathsf{A}}$. In the base case $n = 0$, A's wallet in Γ contains only initial tokens: here, $rdm_{\mathsf{A}}(\Gamma) = \Gamma$, so the thesis is immediate. For the inductive case, let $n > 0$, and assume that the statement holds for all $n' < n$. Let $\tau_1, \ldots, \tau_k$ be the tokens of rank n in A's wallet $\mathsf{A}[\sigma]$. Let:

$$\Gamma = \Gamma_0 \xrightarrow{\mathsf{A}:\mathsf{rdm}(\sigma_0(\tau_1):\tau_1)} \Gamma_1 \xrightarrow{\mathsf{A}:\mathsf{rdm}(\sigma_1(\tau_2):\tau_2)} \cdots \xrightarrow{\mathsf{A}:\mathsf{rdm}(\sigma_{k-1}(\tau_k):\tau_k)} \Gamma_k$$

where $\Gamma_i = \mathsf{A}[\sigma_i] \mid \Gamma_i'$ for all $i < k$. The rank of each token in A's wallet in Γ_k is strictly less then n. Therefore, by the induction hypothesis:

$$W_{\mathsf{A}}(\Gamma_k) = \textstyle\sum_{\tau \in \operatorname{dom}\sigma_k} \sigma_k(\tau) \cdot P^0(\tau)$$

By Lemma 3, we have that $W_{\mathsf{A}}(\Gamma_0) = W_{\mathsf{A}}(\Gamma_k)$, from which the thesis follows. □

Lemma A.2 *Let I be swap-rate continuous. Then, for all v_0, v_1 and $0 < \phi \leq 1$:*

$$XL^{\mathsf{swap}}(v_0, r_0, r_1) \geq XL^{\mathsf{swap}}_{\phi}(v_0, r_0, r_1)$$
$$XR^{\mathsf{swap}}(v_1, r_0, r_1) \geq XR^{\mathsf{swap}}_{\phi}(v_1, r_0, r_1)$$

Proof. For $\phi = 1$ the statement holds trivially, since $XL^{\mathsf{swap}}(\varepsilon, r_0, r_1)$ is a shorthand for $XL^{\mathsf{swap}}_{\phi=1}(\varepsilon, r_0, r_1)$. So, let $\phi < 1$. By (12), for any v_0 we have that:

$$XL^{\mathsf{swap}}_{\phi}(v_0, r_0, r_1) = \frac{v_\phi}{v_0} \qquad \text{where } \exists! v_\phi : I(r_0, r_1) = I(r_0 + \phi v_0, r_1 - v_\phi) \tag{23}$$

$$XL^{\mathsf{swap}}(v_0, r_0, r_1) = \frac{v}{v_0} \qquad \text{where } \exists! v : I(r_0, r_1) = I(r_0 + v_0, r_1 - v) \tag{24}$$

We show that $v > v_\phi$, from which the thesis follows. By (24), we have that the left swap rate for input $(1-\phi)v_0$ on the AMM $(r_0 + \phi v_0 : \tau_0, r_1 - v_\phi : \tau_1)$ is:

$$XL^{\mathsf{swap}}((1-\phi)v_0, r_0 + \phi v_0, r_1 - v_\phi) \;=\; \frac{x}{(1-\phi)v_0} \tag{25}$$

where x is the unique value satisfying:

$$\begin{aligned} I(r_0 + \phi v_0, r_1 - v_\phi) &= I(r_0 + \phi v_0 + (1-\phi)v_0, r_1 - v_\phi - x) \\ &= I(r_0 + v_0, r_1 - (v_\phi + x)) \end{aligned}$$

Since, by (23), $I(r_0 + \phi v_0, r_1 - v_\phi) = I(r_0, r_1)$, then x must satisfy:

$$I(r_0 + v_0, r_1 - (v_\phi + x)) \;=\; I(r_0, r_1)$$

By (24), the unique solution y to the equation $I(r_0, r_1) = I(r_0 + v_0, r_1 - y)$ is $y = v$. Therefore, is must be $v = v_\phi + x$, from which we obtain $x = v - v_\phi$. Since I is swap-rate continuous, by (14) it follows that the swap rate (25) must be positive, and so $v - v_\phi > 0$. The proof for the right swap rate is similar. $\square$

Proof of Theorem 2 Let $\Gamma_0 = \mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1)$. The statement of the theorem singles out three cases for the solutions of the arbitrage game:

(a) $\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0, r_1) > XL_{\Gamma_0}(\tau_0, \tau_1)$
(b) $\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}_{\phi}(\varepsilon, r_0, r_1) > XR_{\Gamma_0}(\tau_0, \tau_1)$
(c) otherwise

These cases are trivially exhaustive; we now show that they are also mutually exclusive. Assume that condition (a) holds. By Lemma A.2, we have that:

$$\forall \varepsilon > 0 \;:\; XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) \;\geq\; XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0, r_1)$$

Therefore, by the limit inequality theorem and by (a):

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}(\varepsilon, r_0, r_1) \;\geq\; \lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0, r_1) \;>\; XL_{\Gamma_0}(\tau_0, \tau_1)$$

Since I is swap-rate-continuous (14), this implies that:

$$1/\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}(\varepsilon, r_0, r_1) \;>\; XL_{\Gamma_0}(\tau_0, \tau_1)$$

from which by (11) it follows that:

$$\lim_{\varepsilon \to 0} XR^{\mathsf{swap}}(\varepsilon, r_0, r_1) \ < \ XR_{\Gamma_0}(\tau_0, \tau_1)$$

which contradicts condition (b). A similar argument can be used to show that if condition (b) is true, then condition (a) is false. Therefore, the two conditions are mutually exclusive.

Now, assume that case (a) applies. We prove that the solution prescribed by Theorem 2 for this case is optimal. In order for the net worth of user A to increase with a left swap, the following must hold:

$$\begin{aligned}&W_{\mathsf{A}}(\Gamma) - W_{\mathsf{A}}(\Gamma_0) = -v_0 \cdot P(\tau_0) + v_0 \cdot XL^{\mathsf{swap}}_{\Gamma_0,\phi}(v_0, \tau_0, \tau_1) \cdot P(\tau_1) > 0\\ &XL^{\mathsf{swap}}_{\Gamma_0,\phi}(v_0, \tau_0, \tau_1) > P(\tau_0)/P(\tau_1)\end{aligned} \tag{26}$$

Given case (a), there must exist a finite v_0 which satisfies (26). Demand-sensitivity (15) ensures that $XL^{\mathsf{swap}}_{\Gamma_0,\phi}(v_0, \tau_0, \tau_1)$ *decreases* to zero with increasing v_0. To find optimal v_0 which maximizes the user net worth, we consider the behaviour of $XL^{\mathsf{swap}}_{\Gamma_0,\phi}(v_0, \tau_0, \tau_1) = XL^{\mathsf{swap}}(v_0, r_0, r_1)$ as v_0 is increased by Δ for $(r_0 : \tau_0, r_1 : \tau_1) \in \Gamma_0$:

$$\begin{gathered} XL^{\mathsf{swap}}_{\phi}(v_0, r_0, r_1) = \tfrac{v}{\phi \cdot v_0} \qquad XL^{\mathsf{swap}}_{\phi}(v_0 + \Delta, r_0, r_1) = \tfrac{v+\Delta'}{\phi \cdot (v_0 + \Delta)}\\ \textit{where } \exists! v, \Delta' : \\ ① \ I(r_0 + \phi \cdot v_0, r_1 - v) = ② \ I(r_0 + \phi \cdot (v_0 + \Delta), r_1 - v - \Delta') = I(r_0, r_1)\end{gathered} \tag{27}$$

From ① and ② and swap rate definition (12) we can infer:

$$XL^{\mathsf{swap}}_{\phi}(\Delta, r_0 + \phi \cdot v_0, r_1 - v) = \Delta'/\Delta \tag{28}$$

For an incremental increase in swap amount v_0 by Δ, the net wealth is only further increased if $\Delta'/\Delta > P(\tau_0)/P(\tau_1)$ (26). Due to demand-sensitivity, this ratio is ever decreasing at higher values of v_0. Thus, for optimal v_0 in $XL^{\mathsf{swap}}_{\phi}(v_0, r_0, r_1)$:

$$\lim\nolimits_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0 + \phi \cdot v_0, r_1 - v) = XL_{\Gamma}(\tau_0, \tau_1) \tag{29}$$

Any infinitesimal increase in v_0 no longer increases net wealth. We omit the proof for case (b) as it follows that of (a).

To prove case (c), we observe that since neither (a) nor (b) hold, there exists no swap which results in a net positive increase in net wealth. Thus, the strategy of the user will be to perform no action at all. $\square$

Proof of Theorem 3 We demonstrate that the same cases in Theorem 2 apply to both Γ, Γ' where $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ and *type*(T) $\in \{\mathsf{dep}, \mathsf{rdm}\}$. For a $\Gamma = (r_0, \tau_0, r_1, \tau_1) \mid \cdots$

- [Dep] maintains the ratio of holdings (τ_0, τ_1), such that $\Gamma' = (c{\cdot}r_0, \tau_0, c{\cdot}r_1, \tau_1) \mid \cdots$ where $c > 1$

– [Rdm] maintains the ratio of holdings (τ_0, τ_1), such that $\Gamma' = (c' \cdot r_0, \tau_0, c' \cdot r_1, \tau_1) \mid \cdots$ where

$$c' = 1 - \frac{v}{sply_{(\tau_0,\tau_1)}(\Gamma)} \tag{30}$$

By SR-consistency (18), for $c'' > 0$

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, r_0, r_1) \;=\; \lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\phi}(\varepsilon, c'' \cdot r_0, c'' \cdot r_1) \tag{31}$$

and since we have shown that the ratio of funds is preserved for either deposit or redeem step,

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma,\phi}(\varepsilon, \tau_0, \tau_1) \;=\; \lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma',\phi}(\varepsilon, \tau_0, \tau_1) \tag{32}$$

And similarly for the right swap rate limit. Thus the arbitrage solution type will remain the same. □

Proof of Theorem 4 By inspection of swap rate definition (17) and funds-consistency (17).

For $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ where $type(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$ and $(r_0 : \tau_0, r_1 : \tau_1) \in \Gamma$, we demonstrate that $(c \cdot r_0 : \tau_0, c \cdot r_1 : \tau_1) \in \Gamma'$, where $c > 0$.

– [Dep]. The balance of τ_0, τ_1 are increased by $c > 0$ and their ratio preserved.
– [Rdm]. The balance of τ_0, τ_1 after redeeming v units of (τ_0, τ_1) in $(r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$ are

$$\begin{aligned} & r_0 - \tfrac{v \cdot r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)}, r_1 - \tfrac{v \cdot r_1}{sply_{(\tau_0,\tau_1)}(\Gamma)} \\ = {} & r_0 \cdot (1 - \tfrac{v}{sply_{(\tau_0,\tau_1)}(\Gamma)}), r_1 \cdot (1 - \tfrac{v}{sply_{(\tau_0,\tau_1)}(\Gamma)}) \end{aligned} \tag{33}$$

and thus the holdings are reduced by a factor $1 > c > 0$ and their ratio is preserved.

Next, consider a left swap performed in $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$

$$XL^{\mathsf{swap}}_{\Gamma,\phi}(v, \tau_0, \tau_1) = \tfrac{r'_1 - r_1}{v} \quad \text{where } \exists! r'_1 : I(r_0, r_1) = I(r_0 + \phi \cdot v, r'_1) \tag{34}$$

Then, from funds-consistency (17), it follows that the swap invariant equality above is preserved if we multiply lhs and rhs parameters of I with $c > 0$

$$I(c \cdot r_0, c \cdot r_1) = I(c \cdot (r_0 + \phi \cdot v), c \cdot r'_1) \tag{35}$$

From this and the swap rate definition (12) we can infer the left swap rate for input amount $c \cdot v$ performed in $\Gamma' = (c \cdot r_0 : \tau_0, c \cdot r_1 : \tau_1) \mid \cdots$

$$XL^{\mathsf{swap}}_{\Gamma',\phi}(c \cdot v, \tau_0, \tau_1) = \tfrac{c \cdot r'_1 - c \cdot r_1}{c \cdot v} \tag{36}$$

Thus, $XL^{\mathsf{swap}}_{\Gamma,\phi}(v,\tau_0,\tau_1) = XL^{\mathsf{swap}}_{\Gamma',\phi}(c \cdot v,\tau_0,\tau_1)$. Here, we consider a change of the swap amount in the rhs to v, so both input amounts are equal. Due to demand-sensitivity (15), for a $\mathsf{T} = \mathsf{dep}$ $(c > 1)$ in $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ this implies

$$XL^{\mathsf{swap}}_{\Gamma,\phi}(v,\tau_0,\tau_1) < XL^{\mathsf{swap}}_{\Gamma',\phi}(v,\tau_0,\tau_1)$$

as the input swap amount is reduced from $c \cdot v$ to v and conversely for $\mathsf{T} = \mathsf{rdm}$ and $(1 > c > 0)$, where it is increased.

$$XL^{\mathsf{swap}}_{\Gamma,\phi}(v,\tau_0,\tau_1) > XL^{\mathsf{swap}}_{\Gamma',\phi}(v,\tau_0,\tau_1) \qquad \square$$

Proof of Theorem 5 Let I be incentive-consistent, and let:

$$\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots \to^* (r'_0 : \tau_0, r'_1 : \tau_1) \mid \cdots = \Gamma' \qquad \text{where } \frac{r_1}{r_0} = \frac{r'_1}{r'_0} \quad (37)$$

We have to prove that:

$$XL^{\mathsf{rdm}}_{\Gamma}(\tau_0,\tau_1) \leq XL^{\mathsf{rdm}}_{\Gamma'}(\tau_0,\tau_1) \qquad XR^{\mathsf{rdm}}_{\Gamma}(\tau_0,\tau_1) \leq XR^{\mathsf{rdm}}_{\Gamma'}(\tau_0,\tau_1) \quad (38)$$

We begin with some auxiliary definitions. Firstly, we define *contours*, i.e. sets of pairs which fulfill a specific incentive-consistent swap invariant constraint:

$$c_k = \{(x,y) \mid I(x,y) = k\} \quad (39)$$

Next, for all $R > 0$ and for all states $\Gamma = (x : \tau_0, y : \tau_1) \mid \cdots$ we define the *projected* left redeem rate in Γ as follows (the right version is similar):

$$\begin{gathered} XL^{\mathsf{rdm}}_{\Gamma,R}(\tau_0,\tau_1) = \frac{x'}{sply_{(\tau_0,\tau_1)}(\Gamma)} \\ \text{s.t. } I(x', R \cdot x') = I(r_0, r_1) \text{ for } (r_0 : \tau_0, r_1 : \tau_1) \in \Gamma \end{gathered} \quad (40)$$

Note that for a state $\Gamma_i = (r_{i,0} : \tau_0, r_{i,1} : \tau_1) \mid \cdots$, the left redeem rate in Γ_i equals to the projected left redeem rate when choosing R as the funds ratio, i.e.:

$$XL^{\mathsf{rdm}}_{\Gamma_i,R_{\Gamma_i}}(\tau_0,\tau_1) = XL^{\mathsf{rdm}}_{\Gamma_i}(\tau_0,\tau_1) \qquad \text{if } R_{\Gamma_i} = r_{i,1}/r_{i,0} \quad (41)$$

Since for $\Gamma \to^* \Gamma'$ in (37), the funds ratio in Γ, Γ' are equal (i.e., $R_{\mathsf{init}} = r_1/r_0 = r'_1/r'_0 = R_{\mathsf{final}}$), the redeem and projected redeem rates in Γ and Γ' coincide. So, to prove (38) it is sufficient to show that $XL^{\mathsf{rdm}}_{\Gamma_i,R_{\mathsf{init}}}(\tau_0,\tau_1)$ increases at each step $\Gamma_i = (r_{i,0} : \tau_0, r_{i,1} : \tau_1) \mid \cdots \to \Gamma_{i+1} = (r_{i+1,0} : \tau_0, r_{i+1,1} : \tau_1) \mid \cdots$ along the trace $\Gamma \to^* \Gamma'$. We proceed to observe the evolution of the projected redeem rate for each possible action.

– [Dep0]. This rule is not used in $\Gamma \to^* \Gamma'$ since $(\_ : \tau_0, \_ : \tau_1) \in \Gamma$.

– [Rdm]/[Dep]. The evolution of the projected redeem rate is:

$$\begin{aligned} XL^{\mathsf{rdm}}_{\Gamma_i, R_{\mathsf{init}}}(\tau_0, \tau_1) &= \frac{x'_i}{sply_{(\tau_0,\tau_1)}(\Gamma_i)} && ① \; I(r_{i,0}, r_{i,1}) = I(x'_i, R_{\mathsf{init}} \cdot x'_i) \\ XL^{\mathsf{rdm}}_{\Gamma_{i+1}, R_{\mathsf{init}}}(\tau_0, \tau_1) &= \frac{c \cdot x'_i}{c \cdot sply_{(\tau_0,\tau_1)}(\Gamma_{i+1})} && ② \; I(c \cdot r_{i,0}, c \cdot r_{i,1}) = I(c \cdot x'_i, R_{\mathsf{init}} \cdot c \cdot x'_i) \end{aligned} \tag{42}$$

From [Dep]/[Rdm], $r_{i+1,0}, r_{i+1,1} = c \cdot r_{i,0}, c \cdot r_{i,1}$ where $c > 0$, resulting in the lhs of ②. This can be inferred as follows: in [Dep] the holdings of τ_0, τ_1 are increased and their ratio preserved. In [Rdm], the holdings of τ_0, τ_1 after redeeming v units of (τ_0, τ_1) in $(r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$ are

$$\begin{aligned} & r_0 - \frac{v \cdot r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)}, r_1 - \frac{v \cdot r_1}{sply_{(\tau_0,\tau_1)}(\Gamma)} \\ = \; & r_0 \cdot \Big(1 - \frac{v}{sply_{(\tau_0,\tau_1)}(\Gamma)}\Big), r_1 \cdot \Big(1 - \frac{v}{sply_{(\tau_0,\tau_1)}(\Gamma)}\Big) \end{aligned} \tag{43}$$

and thus their ratio is also preserved. The rhs of ② in (42) is infered from the funds-consistency property (17). If R_{init} is defined in ①, then ① $\implies$ ②. The projected redeem rate must therefore be maintained for a redeem or deposit, because both numerator and denominator are scaled by the same factor c.

– [SwapL]/[SwapR]. We interpret a swap action $\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$ in Γ_i as (1) a traversal from $(r_{i,0}, r_{i,1})$ to $(r'_{i,0}, r'_{i,1}) = (r_{i,0} + \phi \cdot v_0, r_{i,1} - v)$ such that

$$I(r_{i,0}, r_{i,1}) = I(r_{i,0} + \phi \cdot v_0, r_{i,1} - v) \tag{44}$$

and (2) a subsequent traversal to $(r_{i+1,0}, r_{i+1,1}) = (r'_{i,0} + (1 - \phi) \cdot v_0, r'_{i,1})$ in $\Gamma_{i+1} = (r_{i+1,0} : \tau_0, r_{i+1,1} : \tau_1) \mid \cdots$. Traversal (1) is within a contour c_k, where as traversal (2) occurs from c_k to c'_k where $k \neq k'$: we show this by contradiction. If $c_k = c'_k$, then for traversal (2), the following would hold

$$I(r'_{i,0}, r'_{i,1}) = I(r'_{i,0} + (1 - \phi) \cdot v_0, r'_{i,1}) \tag{45}$$

and imply $\lim_{\varepsilon \to \infty} XL^{\mathsf{swap}}(\varepsilon, r'_{i,0}, r'_{i,1}) = 0$, violating SR-continuity (14), which requires swap rate limits to be positive.
We now examine the pair $(r'_{i,0} - \Delta, r'_{i,1} + \Delta') \in c_k$, which has the same ratio as $R_{\Gamma_{i+1}} = r_{i+1,1}/r_{i+1,0}$ following the swap action, where $(r_{i+1,0}, r_{i+1,1}) \in c_{k'}$.

$$\begin{aligned} R_{\Gamma_{i+1}} = ① \; \frac{r_{i+1,1}}{r_{i+1,0}} = ② \; \frac{r'_{i,1} - \Delta'}{r'_{i,0} + \Delta} = ③ \; \frac{r_{i+1,1} - \Delta'}{r'_{i,0} + \Delta} \\ \text{s.t. } I(r'_{i,0} - \Delta) = I(r'_{i,1} + \Delta') \end{aligned} \tag{46}$$

For ② and ③, we can equate $r'_{i,1}$ and $r_{i+1,1}$ because the funds of τ_1 are maintained in traversal (b). Comparing numerators and denominators in ① and ③, we can infer

$$r_{i+1,1} - \Delta' < r_{i+1,1} \quad r'_{i,0} + \Delta < r_{i+1,0} \tag{47}$$

Thus, $c > 1$ in $(r_{i+1,0}, r_{i+1,1}) = c \cdot (r'_{i,0} + \Delta, r'_{i,1} - \Delta')$, where $(r_{i+1,0}, r_{i+1,1}) \in c'_k$ and $(r'_{i,0} + \Delta, r'_{i,1} - \Delta') \in c_k$. From funds-consistency (17), we can infer that for two pairs $x, y \in c_k$ and $x', y' \in c'_k$ where $y'/x' = y/x$, then $x', y' > x, y$.

Finally, we can compare projected redeem rates in Γ_i and Γ_{i+1} when performing a left swap step.

$$\begin{array}{ll} XL^{\mathsf{rdm}}_{\Gamma_i,R_{\mathsf{init}}}(\tau_0,\tau_1) = \frac{x'_i}{sply_{\tau_0,\tau_1}(\Gamma_i)} & ① \; I(r_{i,0},r_{i,1}) = I(x'_i, R_{\mathsf{init}} \cdot x'_i) \\ XL^{\mathsf{rdm}}_{\Gamma_{i+1},R_{\mathsf{init}}}(\tau_0,\tau_1) = \frac{x'_{i+1}}{sply_{\tau_0,\tau_1}(\Gamma_i)} & ② \; I(r_{i+1,0},r_{i+1,1}) = I(x'_{i+1}, R_{\mathsf{init}} \cdot x'_{i+1}) \end{array} \tag{48}$$

Since swap actions do not affect the supply of minted tokens, we only need to relate x'_i to x'_{i+1} to compare projected redeem rates in Γ_i and Γ_{i+1}. In ① $(x'_i, R_{\mathsf{init}} \cdot x'_i) \in c_k$ and in ② $(x'_{i+1}, R_{\mathsf{init}} \cdot x'_{i+1}) \in c'_k$, so the former must be strictly smaller than the latter. Thus

$$XL^{\mathsf{rdm}}_{\Gamma_i,R_{\mathsf{init}}}(\tau_0,\tau_1) < XL^{\mathsf{rdm}}_{\Gamma_{i+1},R_{\mathsf{init}}}(\tau_0,\tau_1) \tag{49}$$

Therefore, since the projected left redeem rate increases with each step and coincides with the left redeem rate in Γ and Γ' of Theorem 5

$$XL^{\mathsf{rdm}}_{\Gamma}(\tau_0,\tau_1) \leq XL^{\mathsf{rdm}}_{\Gamma'}(\tau_0,\tau_1) \qquad XR^{\mathsf{rdm}}_{\Gamma}(\tau_0,\tau_1) \leq XR^{\mathsf{rdm}}_{\Gamma'}(\tau_0,\tau_1)$$

The proof for the right redeem rate follows similarly and is omitted. □

B Constant product swap rate limit

For a left swap $\mathsf{A} : \mathsf{swapL}(v_0 : \tau_0, v_1 : \tau_1)$ in $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$, the following must hold for the constant product funds invariant

$$(r_0 + \phi \cdot v_0)(r_1 - v) = r_0 \cdot r_1 \tag{50}$$

Thus for a given v_0, the received amount $v : \tau_1$ must be

$$v = \frac{\phi \cdot r_1 \cdot v_0}{r_0 + \phi \cdot v_0} \tag{51}$$

By definition, the right swap rate is simply v/v_0, or

$$XL^{\mathsf{swap}}_{\Gamma}(v_0,\tau_0,\tau_1) = \frac{\phi \cdot r_1}{r_0 + \phi \cdot v_0} \tag{52}$$

The right swap rate limit is thus

$$\lim_{\varepsilon \to 0} XL^{\mathsf{swap}}_{\Gamma}(\varepsilon,\tau_0,\tau_1) = \frac{\phi \cdot r_1}{r_0} \tag{53}$$

$$\frac{\sigma_{\mathsf{A}}\tau \geq v}{\mathsf{A}[\sigma_{\mathsf{A}}] \mid \mathsf{B}[\sigma_{\mathsf{B}}] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{xfer}(\mathsf{B},v:\tau)} \mathsf{A}[\sigma_{\mathsf{A}} - v:\tau] \mid \mathsf{B}[\sigma_{\mathsf{B}} + v:\tau] \mid \Gamma}\ [\textsc{Xfer}]$$

$$\frac{\sigma\tau_i \geq v_i > 0 \ \ (i \in \{0,1\}) \qquad \tau_0 \neq \tau_1 \qquad (\_:\tau_0,\_:\tau_1),(\_:\tau_1,\_:\tau_0) \notin \Gamma}{\begin{array}{l}\mathsf{A}[\sigma] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0,v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0:\tau_0 - v_1:\tau_1 + v_0:(\tau_0,\tau_1)] \mid (v_0:\tau_0, v_1:\tau_1) \mid \Gamma\end{array}}\ [\textsc{Dep0}]$$

$$\frac{\sigma\tau_i \geq v_i > 0 \ \ (i \in \{0,1\}) \qquad r_1 v_0 = r_0 v_1 \qquad v = \frac{v_0}{r_0} \cdot \mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)}{\begin{array}{l}\Gamma \ = \mathsf{A}[\sigma] \ \mid \ (r_0:\tau_0, r_1:\tau_1) \ \mid \ \Gamma' \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0,v_1:\tau_1)} \\ \quad \mathsf{A}[\sigma - v_0:\tau_0 - v_1:\tau_1 + v:(\tau_0,\tau_1)] \ \mid \ (r_0 + v_0:\tau_0, r_1 + v_1:\tau_1) \ \mid \ \Gamma'\end{array}}\ [\textsc{Dep}]$$

$$\frac{\sigma\tau_0 \geq v_0 > 0 \qquad I(r_0 + \phi\, v_0, r_1 - v) = I(r_0, r_1) \qquad 0 < v_1 \leq v \leq r_1}{\begin{array}{l}\mathsf{A}[\sigma] \ \mid \ (r_0:\tau_0, r_1:\tau_1) \ \mid \ \Gamma \xrightarrow{\mathsf{A}:\mathsf{swapL}(v_0:\tau_0,v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0:\tau_0 + v:\tau_1] \ \mid \ (r_0 + v_0:\tau_0, r_1 - v:\tau_1) \ \mid \ \Gamma\end{array}}\ [\textsc{SwapL}]$$

$$\frac{\sigma\tau_1 \geq v_1 > 0 \qquad I(r_0 - v, r_1 + \phi\, v_1) = I(r_0, r_1) \qquad 0 < v_0 \leq v \leq r_0}{\begin{array}{l}\mathsf{A}[\sigma] \ \mid \ (r_0:\tau_0, r_1:\tau_1) \ \mid \ \Gamma \xrightarrow{\mathsf{A}:\mathsf{swapR}(v_0:\tau_0,v_1:\tau_1)} \\ \mathsf{A}[\sigma + v:\tau_0 - v_1:\tau_1] \ \mid \ (r_0 - v:\tau_0, r_1 + v_1:\tau_1) \ \mid \ \Gamma\end{array}}\ [\textsc{SwapR}]$$

$$\frac{\sigma(\tau_0,\tau_1) \geq v > 0 \qquad v_0 = v\frac{r_0}{\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)} \qquad v_1 = v\frac{r_1}{\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)}}{\begin{array}{l}\Gamma \ = \mathsf{A}[\sigma] \ \mid \ (r_0:\tau_0, r_1:\tau_1) \ \mid \ \Gamma' \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:(\tau_0,\tau_1))} \\ \quad \mathsf{A}[\sigma + v_0:\tau_0 + v_1:\tau_1 - v:(\tau_0,\tau_1)] \ \mid \ (r_0 - v_0:\tau_0, r_1 - v_1:\tau_1) \ \mid \ \Gamma'\end{array}}\ [\textsc{Rdm}]$$

Fig. 4: Operational semantics of AMMs.

A Theory of Automated Market Makers in DeFi

Publication Information

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. “A Theory of Automated Market Makers in DeFi.” *Logical Methods in Computer Science* 18, no. 4 (2022): 12.

Contribution

- Co-author.

Remarks

Journal version.

A THEORY OF AUTOMATED MARKET MAKERS IN DEFI

MASSIMO BARTOLETTI [a], JAMES HSIN-YU CHIANG [b], AND ALBERTO LLUCH-LAFUENTE [b]

[a] University of Cagliari, Cagliari, Italy
e-mail address: bart@unica.it

[b] Technical University of Denmark, DTU Compute, Copenhagen, Denmark
e-mail address: jchi@dtu.dk, albl@dtu.dk

Abstract. Automated market makers (AMMs) are one of the most prominent decentralized finance (DeFi) applications. AMMs allow users to trade different types of crypto-tokens, without the need to find a counter-party. There are several implementations and models for AMMs, featuring a variety of sophisticated economic mechanisms. We present a theory of AMMs. The core of our theory is an abstract operational model of the interactions between users and AMMs, which can be concretised by instantiating the economic mechanisms. We exploit our theory to formally prove a set of fundamental properties of AMMs, characterizing both structural and economic aspects. We do this by abstracting from the actual economic mechanisms used in implementations, and identifying sufficient conditions which ensure the relevant properties. Notably, we devise a general solution to the *arbitrage problem*, the main game-theoretic foundation behind the economic mechanisms of AMMs.

1. Introduction

Decentralized finance (DeFi) is a software infrastructure, based on blockchains and smart contracts, which allows users to create and trade crypto-tokens without the intermediation of central authorities, unlike traditional finance [WPG$^+$21, QZA$^+$21]. *Automated Market Makers (AMMs)* are one of the main DeFi archetypes: roughly, AMMs are decentralized markets of crypto-tokens, providing users with three core operations: depositing crypto-tokens to obtain shares in an AMM; the dual operation of redeeming shares in the AMM for the underlying tokens; and swapping tokens of a given type for tokens of another type. The amount of tokens received by a user upon a swap is algorithmically determined by the AMM: roughly, this is the amount of tokens sent from the user to the AMM, times the *swap rate*, which is computed by the AMM based on its internal state and the input amount.

Despite the apparent simplicity of these operations, AMMs manifest an emerging behaviour, where users are incentivized to swap tokens to keep their swap rates aligned with the *exchange rate*, i.e. the ratio between the prices of the exchanged tokens given by external price oracles. Namely, if an AMM offers a better swap rate than the oracles' exchange rate, rational users will perform swaps to narrow the gap. Formally, the optimal strategy can be seen as the solution of a game, called the *arbitrage game*. Executing the optimal strategy closes the gap between AMM's and oracles' exchange rates, and in this sense AMMs offer users exhange rates that align towards the external, global exchange rates.

Preprint submitted to
Logical Methods in Computer Science

As of December 2022, the two AMM platforms leading by user activity, Uniswap and Curve Finance, alone hold \$3B and \$4B worth of tokens, and process \$1B and \$250M worth of transactions daily [uni22, cur22]. Although this massive adoption could suggest that AMMs are a consolidated, well-understood technology, in practice their economic mechanisms are inherently hard to design and implement. For instance, interactions with AMMs are sensitive to *transaction ordering attacks*, where actors with the power to influence the order of transactions in the blockchain can profit from an opportunistic behaviour, causing detriment to other users. The relevance of attacks to AMMs is witnessed by the proliferation of scientific literature on the topic [BCL22, ZQT+21, DGK+20, EMC20, QZG21]. Still, attacks to DeFi applications are not purely theoretical: indeed, there is a growing history of DeFi incidents, which have caused losses exceeding \$2.4B [def22] so far. These issues witness a need for foundational work to devise formal theories of AMMs which allow the understanding of their structural properties and of their economic incentive mechanisms.

Current descriptions of AMMs are either economic models [AKC+21, AC20, EAC21, AEC20], which focus on the incentive mechanism alone, or concrete AMM implementations. While economic models are useful to understand the macroscopic financial aspects of AMMs, they do not precisely describe the interactions between AMMs and their users. Still, a precise formalisation of these interactions is fundamental to understand the structural and economic properties of AMMs, and to determine possible deviations from safe behaviour. Implementations, instead, reflect the exact behaviour of AMMs, but at a level of detail that hampers high-level understanding and reasoning. Moreover, the rich variety of implementations, proposals and models for AMMs, each featuring different economic mechanisms, makes it difficult to compare AMM designs or to provide a clear contour for the space of possible "well behaving" designs.

1.1. **Contributions.** In this paper we exploit techniques from concurrency theory to provide a formal backbone for AMMs and to study their fundamental properties. More specifically, our main contributions can be summarised as follows:

(1) We introduce a formal model of AMMs (section 2), which distils the common features of leading AMM implementations like Uniswap [uni21], Curve [cur21b], and Balancer [bal19]. The core of our model is a transition system that describes the evolution of AMM states resulting from the interaction between users and AMMs. A peculiar aspect of our model is that it abstracts from the *swap rate function*, a key economic mechanism of AMMs, which is used to determine the exchange rates between tokens.

(2) Building upon our model, in section 3 we define basic economic notions like token prices, exchange rates, slippage, net worth, and gain. We compute the gain resulting from swap actions (Lemma 3.2), and we establish a key relation between the gain of swap actions, the swap rate and the exchange rate: a swap action has a strictly positive gain *if and only if* the swap rate is strictly greater than the exchange rate between the swapped tokens (Lemma 3.3). Both lemmata are instrumental to prove many subsequent results.

(3) In section 4 we establish a set of structural properties of AMMs. In particular, we establish preservation results for the supply of tokens (Lemma 4.3) and for the global net worth (Lemma 4.5). We show that assets cannot be frozen within AMMs, i.e. users can always extract any amount of the token reserves deposited in AMMs (Lemma 4.8). In Lemma 4.9 we investigate when transactions can be reordered without affecting the resulting state. In Theorems 4.10 and 4.11 we study compositionality of deposit

and redeem transactions: in particular, we establish that two deposit actions on the same AMM can be merged in a single action (and similarly for two redeems), and that the effect of deposits and redeem actions can be reverted by suitable transactions. Remarkably, all the structural properties in section 4 do not depend on the choice of the swap rate function.

(4) In section 5 we devise sufficient conditions on swap rate functions that induce good behavioural properties of AMMs. These conditions allow us to extend to swap actions the additivity and reversibility properties enjoyed by deposit and redeem actions (Theorems 5.6 and 5.9), as well as to compute the gain of composed and reversed swaps (Lemmata 5.7 and 5.10). We study the effect of deposits and redeems on the swap rate and on the internal exchange rate (Lemma 5.13). We then study the properties of three notable swap rate functions: the constant sum, the constant product, and the constant mean.

(5) In section 6 we investigate the incentive mechanism of AMMs. We start by considering the *arbitrage problem*, which requires to find the action which maximizes the gain of a user. Performing such optimal action has the side effect of aligning the internal exchange rate of the AMM to the external exchange rate given by token price oracles. This gives one of the landmark economic properties of AMMs: assuming rational users, AMMs can be seen as price oracles themselves [AC20]. Notably, while solutions to the arbitrage problem are already known for specific swap rate functions, in Theorem 6.3 we generalize the result to any swap rate function respecting the conditions given in section 5. We then show that depositing tokens into AMMs incentivizes subsequent swaps (Theorem 6.6), while redeeming tokens disincentivizes them (Theorem 6.9). Finally, in Theorems 6.8 and 6.10 we relate the solution of the arbitrage problem in the states before and after a deposit or redeem action, and we compare their gains.

(6) In section 7 we discuss Maximal Extractable Value (MEV), a class of attacks where miners exploit their power of dropping and reordering user transactions (and inserting their own) to increase their gain to the detriment of users. These attacks are one of the most carefully studied AMM phenomena, occuring widely in practice and frequently making up the bulk of interactions with AMMs [QZG21]. The fact that our AMM model can accurately express these attacks supports the coherence of our modelling choices with respect to behaviour exhibited by actual AMM implementations.

(7) In section 8 we discuss some extensions to our basic AMM model to make it closer to the implementation of Uniswap [uni21], and their impact on the results in the paper.

(8) As a byproduct, we provide an open-source Ocaml implementation of our executable semantics as a companion of this paper.[1]

(9) We provide full proofs of all our statements in the Appendices.

1.2. **Related Work.** The work [AKC+21] proposed one of the first analyses of the incentive mechanism of Uniswap. This analysis was then generalised in [AC20] to *constant function AMMs (CFMMs)*, where, for a pair of token types, the reserves r_0, r_1 before a swap and the reserves r'_0, r'_1 after the swap must preserve the invariant $f(r_0, r_1) = f(r'_0, r'_1)$, for a given trading function f. Constant product AMMs, like Uniswap, are an instance of CFMMs, where $f(x, y) = xy$. Both works study the arbitrage problem, for constant product AMMs and CFMMs, respectively. The two works show that the solution can be efficiently computed,

[1]`https://github.com/blockchain-unica/defi-workbench`

and suggest that constant product AMMs accurately report exchange rates. Our work and [AC20] share a common goal, i.e. a theory of AMMs generalizing that of constant product AMMs. However, the two approaches are quite different. The work [AC20] considers a class of AMMs, i.e. CFMMs with a convex trading set, and studies the properties enjoyed by AMMs under these assumptions. Instead, in this paper we devise a minimal set of properties of the swap rate function which induce good behavioural properties of AMMs. Notably, we find conditions on the swap rate function which ensure that a given swap action maximizes the gain of the player (Theorem 6.3). Another difference is that the AMM model in [AC20] describes the evolution of a single AMM, abstracting away the other components of the state (i.e. the users and the other AMMs); instead, we model AMMs as *reactive systems*, borrowing techniques from concurrency theory. While the approach followed by [AC20] is still adequate to study problems that concern AMMs in isolation (e.g., arbitrage), viewing AMMs as reactive systems allows us to study what happens when many agents (users and AMMs) can interact. E.g., we are able to reason about Maximal Extractable Value (section 7).

The work [DKP21] generalises the arbitrage problem to the setting where a swap between two token types τ_0 and τ_n can be obtained through a sequence of n intermediate swaps between τ_i and τ_{i+1}, for $0 \leq i < n$. In practice, this represents the situation where users can interact with different AMM platforms, each one providing its own set of token pairs. To model this scenario, [DKP21] introduces *exchange networks*, i.e. multi-graphs where nodes are tokens, and edges are AMMs which allow users to swap the two endpoint tokens. To encompass different AMM platforms, each edge has its own price function, which determines how many output tokens are paid for a given amount of input tokens. The authors show that, under some conditions on the price functions (i.e., monotonicity, continuity, boundedness and concavity), the arbitrage problem always admits a non-trivial solution. In the special case of constant product AMMs, a closed formula for the solution is provided. Besides arbitrage, [DKP21] also considers the *optimal routing problem*, i.e. finding a strategy to maximize the amount of tokens τ_1 received for at most a given amount of tokens τ_0. Under the same assumptions on the price function used for the arbitrage problem, the optimal routing problem admits a solution. There are several differences between our approach and that of [DKP21], besides the fact that we assume the same swap rate function for all AMMs, and a graph instead of a multi-graph (i.e., we admit at most one AMM for each token pair). A technical difference is that we assume that the amount y of output tokens received for an amount x of input tokens is given by $y = SX(x, r_0, r_1) \cdot x$, whereas [DKP21] defines this amount as $y = f_{r_0,r_1}(x)$. This results in different structural properties for $SX(x, r_0, r_1)$ and $f(x)$ in order to achieve the desired behavioural properties of AMMs. Having the AMM reserves r_0, r_1 as parameters of our swap rate functions SX has a benefit, in that we can express conditions which relate states before and after a transaction: this is what happens, e.g., in the additivity, reversibility and homogeneity properties (Definitions 5.5, 5.8 and 5.11). As a consequence of this choice, compared to [DKP21] our theory encompasses also deposit and redeem actions, providing results that clarify how these actions interfere with swaps (e.g., Theorems 6.6, 6.9, 6.8, and 6.10).

A few alternatives to constant product AMMs have been studied. Balancer [bal19] generalizes the constant product function used by Uniswap to a constant (weighted geometric) mean $f(r_1, \cdots, r_n) = \prod_{i=1}^{n} r_i^{w_i}$, where the weight w_i reflects the relevance of a token τ_i in a tuple of tokens $(\tau_1, \cdots, \tau_n)$. This still fits within the CFMM setting of [AC20], thus inheriting its results about solvability of the arbitrage problem [EAC21]. Curve [Ego19] features a hybrid of a constant sum and constant product function, optimized for large swap volumes between *stable coins*, where the swap rate can support large amounts with small sensitivity. To efficiently compute swap rates, implementations perform numerical approximations [cur21a]. Should these approximations fail to converge, these implementations still guarantee that the AMM remains liquid. The work [KFG21] proposes a constant product invariant that is adjusted dynamically based on the oracle price feed, thus reducing the need for arbitrage transactions, but at the cost of lower fee accrual. AMMs with *virtual* balances have been proposed [vir18] and implemented [moo20b, moo20a]. In these AMMs, the swap rate depends on past actions, besides the current funds balances in the AMM. This, similarly to [KFG21], aims to minimize the need for arbitrage transactions to ensure the local AMM swap rate tends towards the exchange rates. Establishing whether these sophisticated swap rate functions enjoy the properties in section 5 is an interesting open problem.

AMMs are well-known to suffer from transaction-ordering attacks, through which an adversary with the power of influencing the order of transactions (e.g., a miner) can extract value from user transactions. For instance, if the transaction pool contains a swap transaction sent by user A, then a miner M can extract value from A's swap through a transaction "sandwich" constructed as follows. First, M front-runs A's swap with its own swap, crafted so that A's swap decreases A's net worth as much as possible. Then, M closes the sandwich by appending another swap transaction which maximizes M's gain, and finalises the whole sandwich on the blockchain. In this way, A will always have a negative gain, which is counterbalanced by a positive gain of M. This and other kinds of attacks have fostered the research on adversarial and defensive strategies, and on empirical analyses of the impact of attacks [BCL22, CAE22, ZQT$^+$21, QZG21, DGK$^+$20, EMC20]. For instance, the work [BCL22] devises an optimal strategy through which an adversary can extract the maximal value from users' transactions (not only swaps, but also deposits and redeems), in the setting of Uniswap-like AMMs. The swap-rate-agnostic approach pursued by this paper could be exploited to generalise the attack of [BCL22] to AMMs beyond Uniswap.

A high-level survey on various AMM protocols is in [XVPC22].

Comparison with previous work. A preliminary version of this work was presented at COORDINATION 2021 [BCL21b]. The current version substantially extends it, streamlining the theory and providing additional results. A crucial difference between the two papers is that, while in [BCL21b] the semantics of swap actions was parameterized by an invariant between the old and the new token reserves, here we make the semantics parametric w.r.t. the swap rate function SX. This leads to a substantial simplification of the conditions that are put to obtain nice behavioural properties of swaps, and consequently of the corresponding proofs. Among the new results w.r.t. [BCL21b], we mention in particular the additivity and reversibility properties (Theorems 4.10, 4.11, 5.6, and 5.9), and the results that relate the gain of swaps before and after deposit/redeem actions (Theorems 6.6, 6.9, 6.8, and 6.10). Besides these extensions, the current paper includes a discussion of the constant sum and of the constant mean swap rate functions, a new section on MEV attacks (see section 7), and it provides detailed proofs for all its statements.

2. A formal model of Automated Market Makers

We introduce a formal, operational model of AMMs, which focusses on the common operations implemented by AMM platforms. In order to simplify the resulting theory, our model abstracts from a few features that are often found in AMM implementations, like e.g. fees, price updates, and guarded transactions. We discuss in §8 how to extend our model to make it closer to the Uniswap protocol [uni21].

We introduce here some general notation. We denote by $f x$ the application of a function f to a value x (we use parentheses, e.g. $f(x)$, to resolve ambiguities). We denote with $\operatorname{dom} f$ the domain of f. We use the standard notation $f\{{}^{v}/_{x}\}$ to update a partial map f at point x: namely, $f\{{}^{v}/_{x}\}(x) = v$, while $f\{{}^{v}/_{x}\}(y) = f y$ for $y \neq x$.

2.1. AMM basics.

Tokens. We assume a set $\mathbb{T}_0$ of ***atomic token types***, which represent native cryptocurrencies and application-specific tokens. For instance, $\mathbb{T}_0$ may include ETH, the native cryptocurrency of Ethereum, and WBTC, i.e. Bitcoins wrapped with the ERC20 interface for Ethereum tokens. A ***minted token type*** is an unordered pair of distinct atomic token types: if τ_0 and τ_1 are atomic token types and $\tau_0 \neq \tau_1$, then the minted token type $\{\tau_0, \tau_1\}$ represents shares in an AMM holding reserves of τ_0 and τ_1. We denote by $\mathbb{T}_1$ the set of minted token types. In our model, tokens are *fungible*, i.e. individual units of the same type are interchangeable. This means that amounts of tokens of the same type can be split into smaller parts, and two amounts of tokens of the same type can be joined. We use v, v', r, r', x, x' to range over nonnegative real numbers ($\mathbb{R}_{\geq 0}$). We write $\mathbb{T}$ for the universe of all token types, i.e. $\mathbb{T} = \mathbb{T}_0 \cup \mathbb{T}_1$, and we use $\tau, \tau', \ldots$ to range over $\mathbb{T}$. We write $r : \tau$ to denote r units of a token of type τ, either atomic or minted.

Wallets and AMMs. We assume a set of ***users*** $\mathbb{A}$, ranged over by $\mathsf{A}, \mathsf{A}', \ldots$ We model the ***wallet*** of a user A as a term $\mathsf{A}[\sigma]$, where the finite partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_{\geq 0}$ represents A's token balance. We model an ***AMM*** holding reserves of $r_0 : \tau_0$ and $r_1 : \tau_1$ (with $\tau_0 \neq \tau_1$) as an unordered pair $\{r_0 : \tau_0, r_1 : \tau_1\}$. Since the order of the token reserves in an AMM is immaterial, the terms $\{r_0 : \tau_0, r_1 : \tau_1\}$ and $\{r_1 : \tau_1, r_0 : \tau_0\}$ denote exactly the same AMM.

States. We model the interaction between users and AMMs as a labelled transition system (LTS). Its labels represent blockchain ***transactions***, while the ***states*** $\Gamma, \Gamma', \Delta, \ldots$ are finite non-empty compositions of wallets and AMMs. Formally, states are terms of the form:

$$\mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid \{r_1 : \tau_1, r'_1 : \tau'_1\} \mid \cdots \mid \{r_k : \tau_k, r'_k : \tau'_k\}$$

and subject to the following conditions. For all $i \neq j$:

(1) $\mathsf{A}_i \neq \mathsf{A}_j$ (each user has a single wallet);
(2) $\{\tau_i, \tau'_i\} \neq \{\tau_j, \tau'_j\}$ (distinct AMMs cannot hold exactly the same token types).

Note that these conditions allow AMMs to have a common token type τ, e.g. as in $\{r_1 : \tau_1, r : \tau\}$, $\{r' : \tau, r_2 : \tau_2\}$, thus enabling indirect trades between token pairs not directly provided by any AMM. A state is *initial* when it has no AMMs, and its wallets hold only atomic tokens. We stipulate that the ordering of terms in a state is immaterial. Hence, we consider two states Γ and Γ' to be equivalent when they contain the same terms (regardless of their order). For a term Q and a state Γ, we write $Q \in \Gamma$ when $\Gamma = Q \mid \Gamma'$, for some Γ'.

Transactions. State transitions are triggered by transactions $\mathsf{T}, \mathsf{T}', \ldots$, which can have the following forms (where τ_0 and τ_1 are atomic tokens):

- $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$. A deposits $v_0 : \tau_0$ and $v_1 : \tau_1$ to an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$, receiving in return some freshly-minted units of the token $\{\tau_0, \tau_1\}$;
- $\mathsf{A} : \mathsf{swap}(v, \tau_0, \tau_1)$. A tranfers $v : \tau_0$ to an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$, receiving in return some units of τ_1, which are removed from the AMM;
- $\mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$. A redeems v units of the minted token $\{\tau_0, \tau_1\}$: this means that some units of τ_0 and τ_1 are transferred from the AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$ to A's wallet, and that v units of $\{\tau_0, \tau_1\}$ are burned.

We denote with $\mathit{type}(\mathsf{T})$ the type of T (i.e., dep, swap, or rdm), with $\mathit{wal}(\mathsf{T})$ the user whose wallet is affected by T, and with $\mathit{tok}(\mathsf{T})$ the set of token types affected by T. For example, if $\mathsf{T} = \mathsf{A} : \mathsf{swap}(v, \tau_0, \tau_1)$, then $\mathit{type}(\mathsf{T}) = \mathsf{swap}$, $\mathit{wal}(\mathsf{T}) = \mathsf{A}$, and $\mathit{tok}(\mathsf{T}) = \{\tau_0, \tau_1\}$.

Token supply. We use $S_\Gamma \tau$ to denote the ***supply*** of a token type τ in a state Γ, defined as the sum of the reserves of τ in all the wallets and the AMMs in Γ. Formally, we define $S_\Gamma \tau$ by induction on the structure of states as follows:

$$S_{\mathsf{A}[\sigma]} \tau = \begin{cases} \sigma\tau & \text{if } \tau \in \operatorname{dom} \sigma \\ 0 & \text{otherwise} \end{cases} \qquad S_{\{r_0:\tau_0, r_1:\tau_1\}} \tau = \begin{cases} r_i & \text{if } \tau = \tau_i \\ 0 & \text{otherwise} \end{cases} \qquad S_{\Gamma \mid \Gamma'} \tau = S_\Gamma \tau + S_{\Gamma'} \tau$$

For example, let $\Gamma = \mathsf{A}[1 : \tau_0, 2 : \{\tau_0, \tau_1\}] \mid \{3 : \tau_0, 4 : \tau_1\} \mid \{5 : \tau_0, 6 : \tau_2\}$. We have that $S_\Gamma \tau_0 = 9$, $S_\Gamma \tau_1 = 4$, $S_\Gamma \tau_2 = 6$, while $S_\Gamma \tau = 0$ for $\tau \not\in \{\tau_0, \tau_1, \tau_2\}$. Note that $S_\Gamma \tau$ is always defined, since it is defined when Γ is an atomic term (wallet or AMM), and states Γ are *finite* compositions of atomic terms.

2.2. **AMM semantics.** We now formalise the transition rules between states. We write $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ for a state transition from Γ to Γ', triggered by a transaction T. When $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ for some Γ', we say that T is *enabled* in Γ. We denote with $\to^*$ the reflexive and transitive closure of $\to$. Given a finite sequence of transactions $\lambda = \mathsf{T}_1 \cdots \mathsf{T}_k$, we write $\Gamma \xrightarrow{\lambda} \Gamma'$ when $\Gamma \xrightarrow{\mathsf{T}_1} \cdots \xrightarrow{\mathsf{T}_k} \Gamma'$, and in this case we say that λ is enabled in Γ. We say that a state Γ is *reachable* if $\Gamma_0 \to^* \Gamma$ for some initial Γ_0. Hereafter, all the states mentioned in our results are implicitly assumed to be reachable. Given a partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_{\geq 0}$, a token type $\tau \in \mathbb{T}$ and a partial operation $\circ \in \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightharpoonup \mathbb{R}_{\geq 0}$ with $\circ \in \{+, -\}$, we define the partial map $\sigma \circ v : \tau$ as follows:

$$\sigma \circ v : \tau = \begin{cases} \sigma\{{}^{(\sigma\tau) \circ v}/_{\tau}\} & \text{if } \tau \in \operatorname{dom} \sigma \text{ and } (\sigma\tau) \circ v \in \mathbb{R}_{\geq 0} \\ \sigma\{{}^{v}/_{\tau}\} & \text{if } \tau \not\in \operatorname{dom} \sigma \text{ and } \circ = + \end{cases}$$

These partial operations allow to increase/decrease the amount of tokens in a balance. For instance, if $\sigma = 5 : \tau_0$, then $\sigma + 1 : \tau_0 = 6 : \tau_0$, and $\sigma + 1 : \tau_1 = 5 : \tau_0, 1 : \tau_1$.

Deposit. Any user can create an AMM for two tokens τ_0 and τ_1, if such an AMM is not already present in the state. This is achieved by the transaction $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, through which A transfers $v_0 : \tau_0$ and $v_1 : \tau_1$ to the new AMM. In return for the deposit, A receives a certain positive amount of units of a new token type $\{\tau_0, \tau_1\}$, which is minted by the AMM.[2] We formalise this behaviour by the rule:

$$\frac{\sigma\tau_i \geq v_i > 0 \;\; (i \in \{0,1\}) \qquad S_\Gamma\{\tau_0,\tau_1\} = 0}{\begin{array}{l}\mathsf{A}[\sigma] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : \{\tau_0,\tau_1\}] \mid \{v_0 : \tau_0, v_1 : \tau_1\} \mid \Gamma\end{array}} \; [\text{DEP0}]$$

Note that the premise $S_\Gamma\{\tau_0,\tau_1\} = 0$ implies that τ_0, τ_1 are distinct atomic tokens, since otherwise $\{\tau_0,\tau_1\}$ would not be a minted token. If Γ is reachable, then this premise also implies that Γ does *not* contain an AMM for the token pair τ_0, τ_1.

Once an AMM is created, any user can deposit tokens into it — *as long as* doing so preserves the ratio of the token reserves in the AMM. When a user deposits $v_0 : \tau_0$ and $v_1 : \tau_1$ to an existing AMM, it receives in return an amount of minted tokens of type $\{\tau_0,\tau_1\}$. This amount is the ratio between the deposited amount v_0 and the ***redeem rate*** of $\{\tau_0,\tau_1\}$ in the current state Γ, which is defined as follows for $i \in \{0,1\}$:

$$RX^i_\Gamma(\tau_0,\tau_1) = \frac{r_i}{S_\Gamma\{\tau_0,\tau_1\}} \qquad \text{if } \{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma \tag{2.1}$$

The effect of a deposit transaction on the state is then formalised by the following rule:

$$\frac{\sigma\tau_i \geq v_i > 0 \;\; (i \in \{0,1\}) \qquad v_i = v \cdot RX^i_\Gamma(\tau_0,\tau_1)}{\begin{array}{l}\Gamma \;=\; \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \\ \qquad \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : \{\tau_0,\tau_1\}] \mid \{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\} \mid \Delta\end{array}} \; [\text{DEP}]$$

We anticipate that the premises of the [DEP] rule ensure that deposits preserve some key quantities across state transitions, namely:

- the ratio between the reserves of τ_0 and τ_1 in the AMM (see Lemma 4.4(a)). This ratio is always defined, since the reserves of a token in an AMM cannot be zeroed (see Lemma 4.2);
- the *net worth* of the user performing the action (see Lemma 4.5). In particular, the value of the minted tokens $\{\tau_0,\tau_1\}$ received by the user upon a deposit is equal to the value of the tokens τ_0, τ_1 transferred to the AMM;
- the *internal exchange rate* of the AMM (see Lemma 5.12). This preservation property holds for a relevant class of swap rate functions, called *homogeneous* (see Definition 5.11).

Redeem. Any user can redeem units of a minted token $\{\tau_0,\tau_1\}$, obtaining in return units of the underlying atomic tokens τ_0 and τ_1. Their actual amounts are determined by the redeem rate: the idea is that each unit of the minted token $\{\tau_0,\tau_1\}$ can be redeemed for equal fractions of τ_0 and τ_1 remaining in the AMM:

$$\frac{\sigma\{\tau_0,\tau_1\} \geq v > 0 \qquad v < S_\Gamma\{\tau_0,\tau_1\} \qquad v_i = v \cdot RX^i_\Gamma(\tau_0,\tau_1) \;\; (i \in \{0,1\})}{\begin{array}{l}\Gamma \;=\; \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:\{\tau_0,\tau_1\})} \\ \qquad \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : \{\tau_0,\tau_1\}] \mid \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \mid \Delta\end{array}} \; [\text{RDM}]$$

[2]The actual amount of received units is irrelevant. Here we choose v_0, but any other choice would be valid.

Note that the premise $v < S_\Gamma\{\tau_0, \tau_1\}$ ensures that the reserves are not depleted, i.e. $v_i < r_i$. Similarly to the [Dep] rule, the premises of [Rdm] ensure that:

- the net worth of the user performing the action is preserved (i.e., the net worth of burnt minted tokens is equal to that of the tokens received by A);
- the internal exchange rate of the AMM is unaffected by the transition, if the swap rate function is homogeneous.

Swap. Any user A can swap v units of τ_0 in her wallet for some units of τ_1 in an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$ through the transaction $\mathsf{A} : \mathsf{swap}(v, \tau_0, \tau_1)$. Symmetrically, A can swap v of her units of τ_1 for units of τ_0 in the AMM through a transaction $\mathsf{A} : \mathsf{swap}(v, \tau_1, \tau_0)$. The ***swap rate*** $SX(x, r_0, r_1)$ determines the amount of *output tokens* τ_1 that a user receives upon an amount of x *input tokens* τ_0 in an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$.

$$\frac{\sigma\tau_0 \geq x \qquad y = x \cdot SX(x, r_0, r_1) < r_1}{\begin{array}{l}\mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}(x,\tau_0,\tau_1)} \\ \mathsf{A}[\sigma - x : \tau_0 + y : \tau_1] \mid \{r_0 + x : \tau_0, r_1 - y : \tau_1\} \mid \Gamma\end{array}}\,[\text{Swap}]$$

The swap rate function is a parameter of our model: we will discuss in §5 some desiderata for this function, and the behavioural properties they induce on the AMM semantics. As an instance, we consider below the ***constant product swap rate*** [rva18], which is used in mainstream AMM implementations, like e.g. in Uniswap v2 [uni21], Mooniswap [moo20a] and SushiSwap [sus21]. We will use this swap rate function in all the examples in this paper.

Definition 2.1 (Constant product swap rate)**.** The constant product swap rate function is:

$$SX(x, r_0, r_1) \;=\; \frac{r_1}{r_0 + x}$$

The constant product swap rate ensures that, if an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$ evolves into $\{r_0 + x : \tau_0, r_1 - y : \tau_1\}$ upon a swap, then the product between the reserves is preserved:

$$(r_0 + x)(r_1 - y) \;=\; (r_0 + x)\Big(r_1 - x \cdot \frac{r_1}{r_0 + x}\Big) \;=\; r_0 r_1$$

Overall, the behaviour of the transition rules discussed above highlights some landmark properties of AMMs, namely:

- since neither deposits nor redeems affect the net worth of the users performing them, the only way for users to increase their net worth is to perform swaps. Since, as we will see in Lemma 4.5, the *global* net worth is constant, this means that increasing ones' net worth results in a decrease of someone else's net worth;
- the internal exchange rate of an AMM is affected only by swap actions (provided that the swap rate function is homogeneous). This is a natural behaviour, because swaps reflect the value of tokens perceived by users. We will show later in §5 that the constant sum/product/mean swap rate functions are homogeneous.

Example 2.2. Figure 1 shows a computation in our model. We discuss below the effect of the fired transactions, showing in Figure 2 the evolution of the token reserves in the AMM:

(1) $\mathsf{A} : \mathsf{dep}(70 : \tau_0, 70 : \tau_1)$. Starting from an initial state, A creates a new AMM, depositing $70 : \tau_0$ and $70 : \tau_1$. In return, A receives 70 units of the minted token $\{\tau_0, \tau_1\}$.

$$
\begin{array}{rl}
 & \mathsf{A}[70:\tau_0, 70:\tau_1] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \\
\xrightarrow{\mathsf{A}:\mathsf{dep}(70:\tau_0,70:\tau_1)} & \mathsf{A}[70:\{\tau_0,\tau_1\}] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \mid \{70:\tau_0, 70:\tau_1\} \\
\xrightarrow{\mathsf{B}:\mathsf{swap}(30,\tau_0,\tau_1)} & \mathsf{A}[70:\{\tau_0,\tau_1\}] \mid \mathsf{B}[0:\tau_0, 31:\tau_1] \mid \{100:\tau_0, 49:\tau_1\} \\
\xrightarrow{\mathsf{B}:\mathsf{swap}(21,\tau_1,\tau_0)} & \mathsf{A}[70:\{\tau_0,\tau_1\}] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \mid \{70:\tau_0, 70:\tau_1\} \\
\xrightarrow{\mathsf{A}:\mathsf{rdm}(30:\{\tau_0,\tau_1\})} & \mathsf{A}[30:\tau_0, 30:\tau_1, 40:\{\tau_0,\tau_1\}] \mid \mathsf{B}[30:\tau_0, 10:\tau_1] \mid \{40:\tau_0, 40:\tau_1\} \\
\xrightarrow{\mathsf{B}:\mathsf{swap}(30,\tau_0,\tau_1)} & \mathsf{A}[30:\tau_0, 30:\tau_1, 40:\{\tau_0,\tau_1\}] \mid \mathsf{B}[0:\tau_0, 27:\tau_1] \mid \{70:\tau_0, 23:\tau_1\} \\
\xrightarrow{\mathsf{A}:\mathsf{rdm}(30:\{\tau_0,\tau_1\})} & \mathsf{A}[82:\tau_0, 47:\tau_1, 10:\{\tau_0,\tau_1\}] \mid \mathsf{B}[0:\tau_0, 27:\tau_1] \mid \{18:\tau_0, 6:\tau_1\}
\end{array}
$$

Figure 1: Interactions between two users and an AMM.

(2) $\mathsf{B} : \mathsf{swap}(30, \tau_0, \tau_1)$. B swaps 30 units of τ_0 for an amount y of units of τ_1 determined by the swap rate. Since we are assuming the constant product swap rate, we obtain $y = 30 \cdot {}^{70}/_{70+30} = 21$. This swap rate function ensures that swaps preserve the product between the token reserves in the AMM: in Figure 2, we show indeed that the swap results in a traversal along the curve $k = 70 \cdot 70$ from $\{70 : \tau_0, 70 : \tau_1\}$ to $\{100 : \tau_0, 49 : \tau_1\}$.

(3) $\mathsf{B} : \mathsf{swap}(21, \tau_1, \tau_0)$. B reverses the effect of his previous action by swapping 21 units of τ_1 for $y = 21 \cdot {}^{100}/_{49+21} = 30$ of τ_0. Figure 2 shows that the swap results in a traversal from $\{100 : \tau_0, 49 : \tau_1\}$ to $\{70 : \tau_0, 70 : \tau_1\}$ along the curve $k = 70 \cdot 70$.

(4) $\mathsf{B} : \mathsf{rdm}(30 : \{\tau_0, \tau_1\})$. B redeems 30 units of the minted token $\{\tau_0, \tau_1\}$, accordingly reducing the token reserves in the AMM to $\{40 : \tau_0, 40 : \tau_1\}$. Note that the received tokens exhibit the same 1-to-1 ratio as after the initial deposit.

(5) $\mathsf{B} : \mathsf{swap}(30, \tau_0, \tau_1)$. B swaps 30 units of τ_0, receiving $y = 30 \cdot {}^{40}/_{40+30} \approx 17$ units of τ_1. Note that the swap rate, i.e. ${}^{40}/_{40+30} \approx 0.57$, has decreased w.r.t. the first swap, i.e. ${}^{70}/_{70+30} = 0.7$, even though the AMM has the same 1-to-1 reserves ratio. This is caused by the reduction in reserves occurred after A's redeem action: thus, the swap rate is sensitive not only to the ratio of reserves in the AMM, but also on their actual values.

(6) $\mathsf{A} : \mathsf{rdm}(30 : \{\tau_0, \tau_1\})$. A redeems 30 units of the minted token $\{\tau_0, \tau_1\}$, thereby extracting 52 units of τ_0 and 17 units of τ_1 from the AMM. Note that the ratio of redeemed tokens is no longer 1-to-1 as in the previous redeem action, as the prior swap has changed the ratio between the funds of τ_0 and τ_1 in the AMM.

Finally, observe that the supply of both τ_0 and τ_1 remains constant. We will show in Lemma 4.3 that the supply of atomic token types is always preserved. $\diamond$

3. Prices, exchange rates and net worth

In this section we introduce some economic notions which are pivotal for understanding the economic mechanisms of AMMs.

Token prices and exchange rates. We assume an external oracle that prices atomic tokens. Formally, we model this oracle as a function $P : \mathbb{T}_0 \to \mathbb{R}_{>0}$, assuming that the prices given by the oracle are constant along executions (see subsection 8.2 for a discussion about dynamic price updates). While the prices of atomic tokens are constant, that of minted

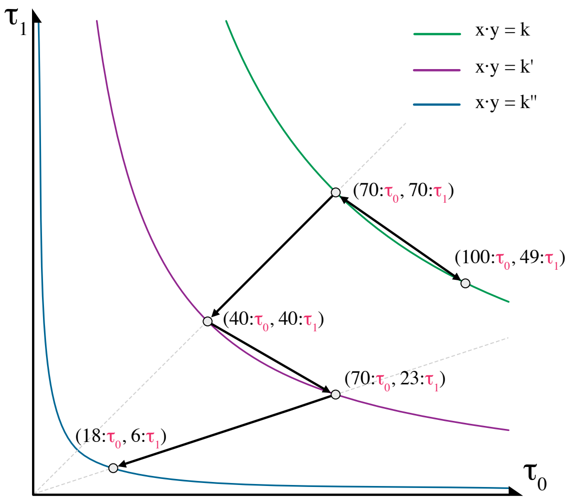


Figure 2: Evolution of reserves of AMM (τ_0, τ_1) along the trace in Figure 1.

tokens may vary at run-time as a function of the state. More precisely, the price $P_\Gamma\{\tau_0, \tau_1\}$ of a minted token $\{\tau_0, \tau_1\}$ depends both on the supply of the minted token in the users' wallets and on the reserves of τ_0 and τ_1 in the AMM:

$$P_\Gamma\{\tau_0, \tau_1\} \;=\; \frac{r_0 \cdot P\tau_0 + r_1 \cdot P\tau_1}{S_\Gamma\{\tau_0, \tau_1\}} \qquad \text{if } \{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma \tag{3.1}$$

For uniformity, we define $P_\Gamma\tau = P\tau$ when $\tau \in \mathbb{T}_0$. Lemma 4.2 will ensure that $r_0, r_1 > 0$ and $S_\Gamma\{\tau_0, \tau_1\} > 0$ in every reachable state Γ containing an AMM for the token pair τ_0, τ_1. Therefore, the price of the token $\{\tau_0, \tau_1\}$ is always defined and positive in reachable states.

The idea underlying Equation (3.1) is that the price of one unit of minted token must be equal to the value of the atomic tokens that can be obtained by redeeming the minted token. Indeed, by rule [Rdm] and Equation (2.1) we have that:

$$P_\Gamma\{\tau_0, \tau_1\} \;=\; \frac{r_0}{S_\Gamma\{\tau_0, \tau_1\}} P\tau_0 + \frac{r_1}{S_\Gamma\{\tau_0, \tau_1\}} P\tau_1 = RX^0_\Gamma(\tau_0, \tau_1) \cdot P\tau_0 + RX^1_\Gamma(\tau_0, \tau_1) \cdot P\tau_1$$

which substantiates our desideratum. This intuition will be formalized later in Lemma 4.6.

The ***exchange rate*** $X(\tau_0, \tau_1)$ between atomic token types τ_0 and τ_1 is the number of units of τ_1 that one can buy with 1 unit of τ_0 at the price given by the external oracle:

$$X(\tau_0, \tau_1) \;=\; \frac{P\tau_0}{P\tau_1} \tag{3.2}$$

Hence, assuming an exchange at the prices of the external oracle, a user paying x units of τ_0 would receive $x \cdot X(\tau_0, \tau_1)$ units of τ_1.

Note that the exchange rate between two token types only depends on external oracles, neglecting the state of AMMs. However, AMMs themselves can act as (decentralised) price oracles [AC20], since they induce an exchange rate based on the effect of swaps in the current state. More precisely, the ***internal exchange rate*** $X_\Gamma(\tau_0, \tau_1)$ between two atomic token types τ_0 and τ_1 in a state Γ is the limit of the swap rate function as x approaches 0: [3]

$$X_\Gamma(\tau_0, \tau_1) \;=\; \lim_{x \to 0} SX(x, r_0, r_1) \qquad \text{if } \{r_0 : \tau_0, r_1 : \tau_1\} \tag{3.3}$$

[3]This notion is also dubbed as marginal price [AC20] or spot exchange rate [XVPC22] in literature.

The intuition is similar to that in Equation (3.2): a user swapping x units of τ_0 for τ_1 through the AMM (for x *very small*) would expect to receive $x \cdot X_\Gamma(\tau_0, \tau_1)$ units of τ_1. We will see later in section 6 that rational users will perform actions that align the internal exchange rate to the one given by external oracles.

Slippage measures the discrepancy between the internal exchange rate and the actual ratio between the amounts of output and input tokens obtained upon the swap [XVPC22]:

$$\Delta X_\Gamma(x, \tau_0, \tau_1) \;=\; \frac{X_\Gamma(\tau_0, \tau_1)}{SX(x, r_0, r_1)} - 1 \qquad \text{if } \{r_0 : \tau_0, r_1 : \tau_1\} \tag{3.4}$$

Ideally, slippage should disadvantage large trades, i.e. trying to obtain a larger amount of tokens with a swap should make them more expensive, increasing the slippage. We will compute in sections 5.6-5.8 the internal exchange rate and the slippage of some common AMMs.

Example 3.1. Let $\Gamma = \mathsf{A}[82 : \tau_0, 47 : \tau_1, 10 : \{\tau_0, \tau_1\}] \mid \{18 : \tau_0, 6 : \tau_1\} \mid \mathsf{B}[\cdots]$ be the final state of the computation in Figure 1. We have that $S_\Gamma\{\tau_0, \tau_1\} = 10$, since only A's wallet contains units of the minted token. Assume that the prices of atomic tokens are $P\tau_0 = 5$ and $P\tau_1 = 9$. The price of the minted token $\{\tau_0, \tau_1\}$ is then:

$$P_\Gamma\{\tau_0, \tau_1\} = \frac{18 \cdot P\tau_0 + 6 \cdot P\tau_1}{10} = \frac{18 \cdot 5 + 6 \cdot 9}{10} = 14.4$$

The exchange rate between the two tokens is:

$$X(\tau_0, \tau_1) \;=\; \frac{P\tau_0}{P\tau_1} \;=\; \frac{5}{9} \;=\; 0.55$$

which means that to buy 1 unit of τ_0, one needs 0.55 units of τ_1. Note instead that the internal exchange rate is:

$$X_\Gamma(\tau_0, \tau_1) \;=\; \lim_{x \to 0} SX(x, 18, 6) = \frac{6}{18} \approx 0.33$$

We will see in Example 6.5 that the discrepancy between internal and oracle exchange rate can by exploited by users to increase their gain. The slippage of a $\mathsf{swap}(x, \tau_0, \tau_1)$ is:

$$\Delta X_\Gamma(x, \tau_0, \tau_1) \;=\; \frac{X_\Gamma(\tau_0, \tau_1)}{SX(x, 18, 6)} - 1 \;=\; \frac{x}{18}$$

from which we can see that the slippage grows with the input amount x. ◇

Net worth and gain. The ***net worth*** of a user A is a measure of A's wealth in tokens (both atomic and minted). Formally, we define the net worth of A in a state Γ as:

$$W_{\mathsf{A}}(\Gamma) \;=\; \begin{cases} \sum_{\tau \in \operatorname{dom} \sigma} \sigma\tau \cdot P_\Gamma \tau & \text{if } \mathsf{A}[\sigma] \in \Gamma \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Note that $W_{\mathsf{A}}(\Gamma) \in \mathbb{R}_{\geq 0}$, since balances σ are *finite* maps, and $P_\Gamma \tau$ is always defined.

The ***global net worth*** $W(\Gamma)$ in a state Γ is the sum of the net worth in users' wallets. Note that the token reserves in AMMs are not accounted for by $W(\Gamma)$, because their value is already recorded by minted tokens held in users' wallets. Indeed, the equality:

$$S_\Gamma\{\tau_0, \tau_1\} \cdot P_\Gamma\{\tau_0, \tau_1\} \;=\; r_0 \cdot P\tau_0 + r_1 \cdot P\tau_1$$

between the net worth of a minted token and the value of the AMM is a direct consequence of the definition of price in Equation (3.1).

We denote by $G_{\mathsf{A}}(\Gamma, \lambda)$ the ***gain*** of user A upon performing a sequence of transactions λ enabled in state Γ (if λ is not enabled in Γ, we stipulate that the gain is zero):

$$G_{\mathsf{A}}(\Gamma, \lambda) \; = \; W_{\mathsf{A}}(\Gamma') - W_{\mathsf{A}}(\Gamma) \qquad \text{if } \Gamma \xrightarrow{\lambda} \Gamma' \tag{3.6}$$

To maximize their gain, users can perform different interactions with the AMM, e.g., by investing tokens or trading units of differently priced token types.

The following lemma quantifies the gain of users upon firing a swap transaction. Note that this quantification does not depend on any of the properties of the swap rate function introduced later on in section 5: actually, it holds for any swap rate function.

Lemma 3.2 (Swap gain)**.** *Let* $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$*, and let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$ *be enabled in* Γ*. Then:*

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}) = \quad x \cdot \big(SX(x, r_0, r_1)\, P\tau_1 - P\tau_0\big) \cdot \left(1 - \frac{\sigma_{\mathsf{A}}\{\tau_0, \tau_1\}}{S_\Gamma\{\tau_0, \tau_1\}}\right) \qquad \textit{if } \mathsf{A}[\sigma_{\mathsf{A}}] \in \Gamma$$

$$G_{\mathsf{B}}(\Gamma, \mathsf{T}) = -x \cdot \big(SX(x, r_0, r_1) P\tau_1 - P\tau_0\big) \cdot \frac{\sigma_{\mathsf{B}}\{\tau_0, \tau_1\}}{S_\Gamma\{\tau_0, \tau_1\}} \qquad \textit{if } \mathsf{B}[\sigma_{\mathsf{B}}] \in \Gamma,\ \mathsf{B} \neq \mathsf{A}$$

A direct consequence of Lemma 3.2 is that if A performs a swap between τ_0 and τ_1 and she holds all the units of the minted token $\{\tau_0, \tau_1\}$, then her gain will be zero. Further, A maximizes her gain when she has no minted tokens of type $\{\tau_0, \tau_1\}$. The lemma also implies that if the user performing the swap has a positive gain, then all the users who hold units of $\{\tau_0, \tau_1\}$ will have a negative gain.

The following lemma states that a swap transaction on an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$ has a strictly positive gain *if and only if* the swap rate is strictly greater than the oracle exchange rate between τ_0 and τ_1. This holds for *any* swap rate function, under the condition that the user who performs the swap has no minted tokens of type $\{\tau_0, \tau_1\}$.

Lemma 3.3 (Swap rate *vs.* exchange rate)**.** *Let* $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$ *be such that* $\sigma\{\tau_0, \tau_1\} = 0$*, and let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$ *be enabled in* Γ*. Then:*

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}) \circ 0 \iff SX(x, r_0, r_1) \circ X(\tau_0, \tau_1) \qquad \text{for } \circ \in \{<, =, >\}$$

Example 3.4. Let $\Gamma_0 = \mathsf{A}[70 : \tau_0, 70 : \tau_1] \mid \mathsf{B}[30 : \tau_0, 10 : \tau_1]$ be the initial state of the computation in Figure 1. Let $P\tau_0 = 5$ and $P\tau_1 = 9$. The users' net worth in Γ_0 and in the final state $\Gamma = \mathsf{A}[82 : \tau_0, 47 : \tau_1, 10 : \{\tau_0, \tau_1\}] \mid \mathsf{B}[0 : \tau_0, 27 : \tau_1] \mid \{18 : \tau_0, 6 : \tau_1\}$ is as follows:

$$W_{\mathsf{A}}(\Gamma_0) = 70 \cdot P\tau_0 + 70 \cdot P\tau_1 = 980 \qquad W_{\mathsf{B}}(\Gamma_0) = 30 \cdot P\tau_0 + 10 \cdot P\tau_1 = 240$$
$$W_{\mathsf{A}}(\Gamma) = 82 \cdot P\tau_0 + 47 \cdot P\tau_1 + 10 \cdot P_\Gamma\{\tau_0, \tau_1\} = 977 \quad W_{\mathsf{B}}(\Gamma) = 27 \cdot P\tau_1 = 243$$

Note that A's net worth of has decreased w.r.t. the initial state, while that of B has increased: indeed, the gain of A upon the sequence of transactions λ is $G_{\mathsf{A}}(\Gamma, \lambda) = 977 - 980 = -3$, while that of B is $G_{\mathsf{B}}(\Gamma, \lambda) = 243 - 240 = 3$. One may think that B has been more successful than A, but this depends on the users' goals. Note, e.g., that A holds 10 units of the minted token $\{\tau_0, \tau_1\}$, whose price may increase in the future. $\diamond$

4. Structural properties of AMMs

We now establish some structural properties of AMMs, which do not depend on the design of the economic mechanisms, i.e. on the choice of the swap rate function. These structural properties are the basis for AMM interactions that occur in the wild, and that cumulatively give rise to complex emerging behaviours like arbitrage and MEV. Hence, establishing these structural properties is a preliminary sanity check for our AMM model. We will provide further support for the coherence between our model and actual AMMs by showing that the above-mentioned complex behaviours are expressible in our model (see section 6 and section 7).

First, we establish that the AMMs' transition system is deterministic. This follows from the fact that, given a state Γ and a transaction T, there is at most one applicable rule. Note that determinism is a crucial property for blockchains, since it ensures that all the nodes in the blockchain network are able to reconstruct a common state from a sequence of transactions. Therefore, it makes sense that determinism holds also for our AMM model.

Lemma 4.1 (Determinism)**.** *If* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ *and* $\Gamma \xrightarrow{\mathsf{T}} \Gamma''$*, then* $\Gamma' = \Gamma''$*.*

We can lift the statement to sequences of transactions by using a simple inductive argument. The same applies to other single-step results in this section.

Lemma 4.2 ensures that the reserves in an AMM cannot be zeroed, and that the same holds for the units of any minted token. Summing up, this ensures that the price of any minted token is always defined and positive.

Lemma 4.2 (Non depletion)**.** *For all states* Γ*, if* $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$ *then:*

(a) $r_i > 0$*, for* $i \in \{0, 1\}$*;*
(b) $S_\Gamma\{\tau_0, \tau_1\} > 0$.

4.1. **Preservation properties.** Lemma 4.3 ensures that transactions preserve the supply of *atomic* tokens. Minted tokens, instead, are preserved only by swap transactions, since deposit and redeem transactions, respectively, create and destroy minted tokens. This fact will be instrumental to prove the preservation of the net worth (see Lemma 4.5).

Lemma 4.3 (Preservation of token supply)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*. Then:*

(a) *for all* $\tau \in \mathbb{T}_0$*,* $S_\Gamma \tau = S_{\Gamma'} \tau$
(b) *if type*$(\mathsf{T}) = \mathsf{swap}$*, then for all* $\tau \in \mathbb{T}_1$*,* $S_\Gamma \tau = S_{\Gamma'} \tau$

Lemma 4.4 states that deposit and redeem transactions preserve the reserves ratio in AMMs, the redeem rate, and the price of minted tokens. These preservation properties will be exploited later on to determine the solution to the arbitrage game after deposits and redeems (see Theorems 6.8 and 6.10).

Lemma 4.4 (Preservation upon deposits/redeems)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*, with* $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$*. If type*$(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$*, then:*

(a) *if* $\{r'_0 : \tau_0, r'_1 : \tau_1\} \in \Gamma'$*, then* ${r_1}/{r_0} = {r'_1}/{r'_0}$
(b) $RX^i_\Gamma(\tau_0, \tau_1) = RX^i_{\Gamma'}(\tau_0, \tau_1)$*, for* $i \in \{0, 1\}$
(c) $P_\Gamma\{\tau_0, \tau_1\} = P_{\Gamma'}\{\tau_0, \tau_1\}$

Lemma 4.5 ensures that transactions (of any type) preserve the *global* net worth, whereas the net worth of individual users is preserved only by redeem and deposit transactions. A direct consequence of this preservation result is that users can increase their net worth only by performing swaps: Indeed, we will find in Theorem 6.3 that the solution of the arbitrage game only contains swap transactions. Furthermore, if a user has a positive gain, then some other user must have a loss.

Lemma 4.5 (Preservation of net worth)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. *Then:*
(a) *if* $type(\mathsf{T}) \neq \mathsf{swap}$ *then, for all* A: $W_{\mathsf{A}}(\Gamma) = W_{\mathsf{A}}(\Gamma')$
(b) $W(\Gamma) = W(\Gamma')$

The following lemma, which is a direct consequence of Lemma 4.5(a), supports the definition of the price of minted tokens in Equation (3.1): indeed, computing the net worth of a user A under that price definition corresponds to making A first redeem all her minted tokens, and then summing the price of the resulting atomic tokens.

Lemma 4.6. *Let* $\Gamma \xrightarrow{\lambda} \Gamma'$, *where* λ *contains only* rdm *actions of* A. *If* $\mathsf{A}[\sigma] \in \Gamma'$ *and* $\operatorname{dom}\sigma \cap \mathbb{T}_1 = \emptyset$, *then:*

$$W_{\mathsf{A}}(\Gamma) \;=\; \sum_{\tau \in \operatorname{dom}\sigma} \sigma\tau \cdot P\tau$$

Example 4.7. Let $\Gamma = \mathsf{A}[82 : \tau_0, 47 : \tau_1, 10 : \{\tau_0, \tau_1\}] \mid \{27 : \tau_0, 9 : \tau_1\} \mid \mathsf{B}[5 : \{\tau_0, \tau_1\}]$, and let $P\tau_0 = 5$ and $P\tau_1 = 9$. We have that $P_\Gamma\{\tau_0, \tau_1\} = 14.4$, and $W_{\mathsf{A}}(\Gamma) = 977$. Assume that A performs a transaction from Γ to redeem all 10 units of $\{\tau_0, \tau_1\}$ in her wallet. The resulting state is $\Gamma' = \mathsf{A}[100 : \tau_0, 53 : \tau_1] \mid \{9 : \tau_0, 3 : \tau_1\} \mid \cdots$. We compute A's net worth in Γ' using the oracle token prices:

$$W_{\mathsf{A}}(\Gamma') \;=\; 100 \cdot P\tau_0 + 53 \cdot P\tau_1 \;=\; 100 \cdot 5 + 53 \cdot 9 \;=\; 977$$

which is coherent with the net worth predicted by Lemma 4.6. ◇

4.2. **Liquidity.** Lemma 4.8 ensures that funds cannot be *frozen* in an AMM, i.e. that users can always redeem arbitrary amounts of the tokens deposited in an AMM, as long as the reserves are not zeroed. Note that, since $\{r_0 : \tau_0, r_1 : \tau_1\} = \{r_1 : \tau_1, r_0 : \tau_0\}$, the statement also holds when swapping r_0 with r_1.

Lemma 4.8 (Liquidity)**.** *Let* Γ *be such that* $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$. *Then, for all* $r_0' < r_0$, *there exist* $r_1' < r_1$, Γ' *and* λ *only containing* rdm *transactions such that* $\Gamma \xrightarrow{\lambda} \{r_0' : \tau_0, r_1' : \tau_1\} \mid \Gamma'$.

4.3. **Reordering of transactions.** In general, given two transactions T_0 and T_1 and a state Γ, executing $\mathsf{T}_0\mathsf{T}_1$ or $\mathsf{T}_1\mathsf{T}_0$ from Γ yields different states. However, under some conditions it is possible to invert the order of the two transactions, preserving the resulting state. This is always the case, e.g., of two transactions which operate on disjoint sets of tokens. Lemma 4.9 establishes sufficient conditions for preserving the state upon reordering. Besides the case cited before, this is always possible if both transactions are deposits, or if they are bot redeems (case (a) of the statement). Note that, in these cases, the assumption that $\mathsf{T}_0\mathsf{T}_1$ is enabled in Γ implies that also $\mathsf{T}_1\mathsf{T}_0$ is such. This is no longer true when one of the two transactions is a deposit and the other one is a redeem. For instance, if T_1 redeems the minted tokens obtained upon a deposit T_0, then T_1 may not be enabled in Γ because there are not enough minted tokens in the user's wallet. Therefore, case (b) of the statement uses the additional hypothesis that also $\mathsf{T}_1\mathsf{T}_0$ is enabled in Γ.

Lemma 4.9 (Reordering of transactions)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}_0\mathsf{T}_1} \Gamma_{01}$*. Then:*

(a) *if* $\mathit{tok}(\mathsf{T}_0) \cap \mathit{tok}(\mathsf{T}_1) = \emptyset$ *or* $\mathit{type}(\mathsf{T}_0) = \mathit{type}(\mathsf{T}_1) \in \{\mathsf{dep}, \mathsf{rdm}\}$*, then* $\Gamma \xrightarrow{\mathsf{T}_1\mathsf{T}_0} \Gamma_{01}$*;*

(b) *otherwise, if* $\mathit{type}(\mathsf{T}_0), \mathit{type}(\mathsf{T}_1) \neq \mathsf{swap}$ *and* $\Gamma \xrightarrow{\mathsf{T}_1\mathsf{T}_0} \Gamma_{10}$*, then* $\Gamma_{01} = \Gamma_{10}$*.*

As we shall see in section 6, it is actually desirable, and crucial for the economic mechanism of AMMs, that swaps interfere with other transactions that trade the same token type.

4.4. **Additivity of deposit and redeem actions.**

Deposit and redeem actions satisfy an additivity property: if a user performs two successive deposits (resp. redeems) on an AMM, then the same result can be obtained through a single deposit (resp. redeem). Instead, swap actions are not additive, in general: we will study sufficient conditions for the additivity of swap actions in section 5 (see Theorem 5.6).

Theorem 4.10 (Additivity)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}_0} \Gamma_0 \xrightarrow{\mathsf{T}_1} \Gamma_1$*. Then:*

(1) *if* $\mathsf{T}_0 = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ *and* $\mathsf{T}_1 = \mathsf{A} : \mathsf{dep}(v'_0 : \tau_0, v'_1 : \tau_1)$*, then:*

$$\Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0+v'_0:\tau_0, v'_1+v'_1:\tau_1)} \Gamma_1$$

(2) *if* $\mathsf{T}_0 = \mathsf{A} : \mathsf{rdm}(v : \tau)$ *and* $\mathsf{T}_1 = \mathsf{A} : \mathsf{rdm}(v' : \tau)$*, then:*

$$\Gamma \xrightarrow{\mathsf{A}:\mathsf{rdm}(v+v':\tau)} \Gamma_1$$

4.5. **Reversibility of deposit and redeem actions.**

The following theorem establishes that deposit and redeem transactions are *reversible*: more precisely, the effect of a deposit action can be reverted by a redeem action, and *vice versa*, the effect of a redeem action can be reverted by a deposit action. The only exception is a deposit action that creates an AMM, through the rule [DEP0]. Swap actions are not reversible, in general: we will study sufficient conditions for their reversibility in section 5 (see Theorem 5.9).

Theorem 4.11 (Reversibility)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*, where* $\mathit{type}(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$ *and for all* $\tau \in \mathbb{T}_1$*, if* $S_\Gamma \tau = 0$ *then* $S_{\Gamma'} \tau = 0$*. Then there exists* T^{-1} *such that* $\Gamma' \xrightarrow{\mathsf{T}^{-1}} \Gamma$*.*

In general, the study of reversible computation models, which dates back to [Ben73], is an active area of research, which has led to a wide range of applications in software systems [MSG$^+$20]. In particular, the reversibility of AMM actions has useful consequences on their behaviour. For instance, it guarantees that, starting from a "stable" state where no arbitrage is possible, after any transaction it is possible to return to the stable state. More in general, if the swap rate function satisfies the conditions of section 5 that ensure the additivity and reversibility also for swap actions, then for any sequence of transactions:

$$\Gamma_0 \xrightarrow{\mathsf{T}_1} \Gamma_1 \xrightarrow{\mathsf{T}_2} \cdots \Gamma_n$$

it is possible to fire another transaction and return to the state Γ_0. Indeed, by additivity we obtain that the effect of the sequence $\mathsf{T}_1 \cdots \mathsf{T}_n$ can be emulated by a single transaction T, and then reversibility ensures that T can be reversed, i.e.:

$$\Gamma_0 \xrightarrow{\mathsf{T}} \Gamma_n \xrightarrow{\mathsf{T}^{-1}} \Gamma_0$$

5. The swap rate function

In the previous section we have established some key structural properties of deposit and redeem actions, e.g. their additivity and reversibility. In general, these properties do not hold for swap actions: it is easy to find swap rate functions $SX \in \mathbb{R}^3_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ that make these properties false. Throughout this section we introduce some general properties of swap rate functions, and we discuss the properties they induce on the behaviour of AMMs. In sections 5.6-5.8 we then discuss the properties enjoyed by the swap rate functions used in some concrete AMM implementations. Coherently with these implementations, we assume that a swap rate function is defined and non-negative for all $x > 0$, and that the internal exchange rate (i.e., the limit of SX for x leading to 0, see (3.3)) is always defined.

5.1. **Output-boundedness.**

Output boundedness guarantees that an AMM has always enough output tokens τ_1 to send to the user who performs a $\mathsf{swap}(x, \tau_0, \tau_1)$.

Definition 5.1 (Output-boundedness)**.** A swap rate function SX is *output-bounded* when, for all x, r_0, r_1 such that $x \geq 0$ and $r_0, r_1 > 0$:

$$x \cdot SX(x, r_0, r_1) < r_1$$

The following lemma establishes sufficient conditions for a swap action to be enabled.

Lemma 5.2. *Let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, and let* $\mathsf{A}[\sigma] \in \Gamma$*. If* $S_\Gamma\{\tau_0, \tau_1\} > 0$*,* $\sigma(\tau_0) \geq x$ *and* SX *is output-bounded, then* T *is enabled in* Γ*.*

5.2. **Monotonicity.**

Consider a transaction $\mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$ on an AMM $\{r_0 : \tau_0, r_1 : \tau_1\}$. Without making any assumptions on the swap rate function, there is no relation between the effect of this transaction and that of a swap where the parameters have been varied. Monotonicity, instead, ensures that there exists a meaninful relation: the swap rate increases if we decrease the input amount x or the reserves of τ_0, and if we increase the reserves of τ_1. The intuition is that lower reserves of τ_0 in the AMM make the $x : \tau_0$ paid by A more “valuable” for the AMM, hence the AMM will output more units of τ_1 for the same input amount. Increasing the reserves of τ_1 in the AMM (keeping those of τ_0 unaltered) produces the same effect. Monotonicity on x also ensures that the internal exchange rate of the AMM is defined, for each token pair.

Definition 5.3 (Monotonicity)**.** A swap rate function SX is *monotonic* when:

$$x' \leq x,\ r_0' \leq r_0,\ r_1 \leq r_1' \implies SX(x, r_0, r_1) \leq SX(x', r_0', r_1')$$

Further, SX is *strictly monotonic* when, for $i \in \{0, 1, 2\}$ and $\lhd_i \in \{<, \leq\}$:

$$x' \lhd_0 x,\ r_0' \lhd_1 r_0,\ r_1 \lhd_2 r_1' \implies SX(x, r_0, r_1) \lhd_3 SX(x', r_0', r_1')$$

where:

$$\lhd_3 = \begin{cases} \leq & \text{if } \lhd_i = \leq \text{ for } i \in \{0,1,2\} \\ < & \text{otherwise} \end{cases}$$

Note that strict monotonicity trivially implies monotonicity. The following lemma relates monotonicity of the swap rate function with the gain of swap transactions, concretising the intuition given before from the point of view of A's gain.

Lemma 5.4. *Let* $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$ *and* $\Gamma' = \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta$*, with* $r'_0 \leq r_0$ *and* $r_1 \leq r'_1$*, and let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$ *be enabled in* Γ *and in* Γ'*. If* SX *is monotonic, then* $G_{\mathsf{A}}(\Gamma, \mathsf{T}) \leq G_{\mathsf{A}}(\Gamma', \mathsf{T})$.

5.3. **Additivity.** To extend the additivity property of Theorem 4.10 to swap actions, we must require that the swap rate function is additive.

Definition 5.5 (Additivity)**.** A swap rate function SX is *additive* when:

$$\alpha = SX(x, r_0, r_1),\ \beta = SX(y, r_0 + x, r_1 - \alpha x) \implies SX(x + y, r_0, r_1) = \frac{\alpha x + \beta y}{x + y}$$

The idea here is that a user fires a swap transaction (say, T_0) with input amount x in a state Γ, and then in the state reached after firing T_0, she fires another swap transaction (say, T_1) with input amount y on the same AMM. The definition of additivity requires that the swap rate of a swap transaction with input amount $x + y$ in Γ is in a given relation with the swap rates computed for T_0 and T_1 and with the input amounts x and y. Theorem 5.6 states that if this relation holds, then a single swap with input amount $x + y$ in Γ produces exactly the same effect of performing first T_0 and then T_1. Then, Lemma 5.7 allows us to compute the gain of this transaction as the sum of the gains of T_0 and T_1.

Theorem 5.6 (Additivity of swap)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}_0} \Gamma_0 \xrightarrow{\mathsf{T}_1} \Gamma_1$*, with* $\mathsf{T}_i = \mathsf{A} : \mathsf{swap}(x_i, \tau_0, \tau_1)$ *for* $i \in \{0, 1\}$*. If* SX *is additive, then:*

$$\Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}(x_0+x_1,\tau_0,\tau_1)} \Gamma_1$$

Lemma 5.7 (Additivity of swap gain)**.** *Let* $\mathsf{T}(x) = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, and let* $\Gamma \xrightarrow{\mathsf{T}(x_0)} \Gamma'$*. If* SX *is output-bounded and additive, then:*

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0 + x_1)) \;=\; G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) + G_{\mathsf{A}}(\Gamma', \mathsf{T}(x_1))$$

5.4. **Reversibility.** The reversibility property in Theorem 4.11 states that the effect of deposit and redeem transactions can be reverted. We now devise a property of swap rate functions that give the same guarantee for swap transactions.

Definition 5.8 (Reversibility)**.** A swap rate function SX is *reversible* when:

$$\alpha = SX(x, r_0, r_1) \implies SX(\alpha x, r_1 - \alpha x, r_0 + x) \;=\; \frac{1}{\alpha}$$

Consider now a state $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$, and let $\alpha = \lim_{x \to 0} SX(x, r_0, r_1)$ be the internal exchange between τ_0 and τ_1 in Γ. If the swap rate function is reversible, then:

$$\lim_{x\to 0} SX(x, r_1, r_0) \;=\; \lim_{x\to 0} SX(\alpha x, r_1 - \alpha x, r_0 + x) \;=\; \lim_{x\to 0} \frac{1}{\alpha} \;=\; \frac{1}{\alpha}$$

from which we obtain:

$$X_\Gamma(\tau_1, \tau_0) \;=\; \frac{1}{X_\Gamma(\tau_0, \tau_1)} \tag{5.1}$$

The intuition of Definition 5.8 is that, to reverse the effect of a swap transaction T that pays $x : \tau_0$ to receive $y : \tau_1$, one must fire a swap transaction T^{-1} that pays $y : \tau_1$ to receive $x : \tau_0$. Of course, this results in the same AMM state that we had before performing T. Writing α for the swap rate $SX(x, r_0, r_1)$, the [Swap] rule fixes $y = \alpha x$. Hence, assuming that in the initial state the AMM has reserves $r_0 : \tau_0$ and $r_1 : \tau_1$, after performing T its reserves will be $r_0 + x : \tau_0$ and $r_1 - \alpha x : \tau_1$. In this state, requiring that the swap rate for an input of $y : \tau_1$ is $\frac{1}{\alpha}$ (as done by Definition 5.8) implies that the AMM outputs $x : \tau_0$, reverting the reserves of the AMM to the initial values.

The following theorem formalises the intuition above, establishing that, when the swap rate function is reversible, swap transactions are reversible. Together with Theorem 4.11, all the AMM actions are reversible under this hypothesis.

Theorem 5.9 (Reversibility of swap)**.** *Let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, and let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*. If* SX *is reversible, then there exists* T^{-1} *such that* $\Gamma' \xrightarrow{\mathsf{T}^{-1}} \Gamma$*.*

Lemma 5.10 allows us to compute the gain of the reverse transaction T^{-1} in the state reached after performing T as a function of the gain of T. As expected by preservation of the global net worth, the gain of T^{-1} is the opposite of that of T.

Lemma 5.10. *Let* $\mathsf{T} = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, and let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*. If* SX *is reversible, then* $G_{\mathsf{A}}(\Gamma, \mathsf{T}) = -G_{\mathsf{A}}(\Gamma', \mathsf{T}^{-1})$*.*

5.5. **Homogeneity.** A swap rate function is homogeneous when the swap rate is not affected by a linear scaling of the three parameters. Homogeneity is useful to relate the swap rate before and after deposit or redeem transactions, since their effect is a linear scaling of the AMM reserves. Lemma 5.12 establishes one the the landmark properties of AMMs we have anticipated in section 2: when the swap rate function is homogeneous, deposits and redeems do not affect the internal swap rate.

Definition 5.11 (Homogeneity)**.** A swap rate function SX is *homogeneous* when, for $a > 0$:

$$SX(ax, ar_0, ar_1) \;=\; SX(x, r_0, r_1)$$

Lemma 5.12 (Preservation of internal exchange rate upon deposits/redeems)**.** *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$*, where* $tok(\mathsf{T}) = \{\tau_0, \tau_1\}$ *and* $type(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$*. If* SX *is homogeneous, then:*

$$X_\Gamma(\tau_0, \tau_1) = X_{\Gamma'}(\tau_0, \tau_1)$$

The following lemma shows that deposits increase swap rates, whilst redeems have the opposite effect. Dually, deposits decrease the slippage, while redeems increase it. In section 6 we will exploit this fact to show that deposits incentivize swaps, while redeems disincentivize them (see Theorems 6.6 and 6.9).

Lemma 5.13. *Let* $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$, *where* $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$, $\{r'_0 : \tau_0, r'_1 : \tau_1\} \in \Gamma'$ *and* $tok(\mathsf{T}) = \{\tau_0, \tau_1\}$. *If* SX *is homogeneous and strictly monotonic, then for all* $x > 0$*:*

(a) $type(\mathsf{T}) = \mathsf{dep} \implies SX(x, r_0, r_1) < SX(x, r'_0, r'_1)$ *and* $\Delta X_\Gamma(x, \tau_0, \tau_1) > \Delta X_{\Gamma'}(x, \tau_0, \tau_1)$

(b) $type(\mathsf{T}) = \mathsf{rdm} \implies SX(x, r_0, r_1) > SX(x, r'_0, r'_1)$ *and* $\Delta X_\Gamma(x, \tau_0, \tau_1) < \Delta X_{\Gamma'}(x, \tau_0, \tau_1)$

It is easy to find swap rate functions that violate the properties discussed before: for instance $SX(x, r_0, r_1) = 1/x$ violates output-boundedness, additivity, reversibility and homogeneity. In the rest of the section we discuss some notable swap rate functions, used in actual AMM implementations, showing that they satisfy most of our properties.

5.6. **Constant sum swap rate.** The *constant sum* function mandates the sum of the token reserves in an AMM to remain constant, i.e. $r_0 + r_1 = k$, where the constant k is fixed upon the first deposit in the AMM.

Theorem 5.14 (Constant sum swap rate)**.** *The* constant sum *swap rate function:*

$$SX(x, r_0, r_1) \;=\; 1$$

is monotonic, reversible, additive, and homogeneous. Furthermore, its internal swap rate and its slippage are given by:

$$X_\Gamma(\tau_0, \tau_1) \;=\; 1 \qquad\qquad \Delta X_\Gamma(x, \tau_0, \tau_1) \;=\; 0$$

Note that the constant sum function is *not* output-bounded, since the output amount may exceed the reserves of the output token. A positive aspect of constant sum AMMs is that they do not suffer from slippage. With constant sum AMMs, the internal exchange rate is always 1, and so there is zero slippage (see Equations (3.3) and (3.4)). A negative aspect is that constant sum AMMs do not allow the token reserves to grow unboundedly: indeed, the bound is fixed with the first deposit. This makes constant sum AMMs unsuitable for scenarios where one wants the liquidity of the AMM to increase over time, and to incentivise users to deposit through minted tokens. When the oracle and internal exchange rates are not aligned (i.e., when the prices of the two tokens are different), then rational users will drain the reserves of the most expensive token type held by the AMM. Despite these drawbacks, the constant sum swap rate is suitable situations where the two token types in the AMM are supposed to be equally prices, like for stablecoins. This is the case e.g. for mStable [mSt20].

5.7. **Constant product swap rate.** The constant product swap rate function (introduced before in Definition 2.1) enjoys all the properties discussed previously in this section.[4]

Theorem 5.15 (Constant product)**.** *The constant product swap rate function is output-bounded, strictly monotonic, reversible, additive, and homogeneous. Furthermore, its internal swap rate and its slippage are given by:*

$$X_\Gamma(\tau_0, \tau_1) \;=\; \frac{r_1}{r_0} \qquad\qquad \Delta X_\Gamma(x, \tau_0, \tau_1) \;=\; \frac{x}{r_0}$$

Compared to the constant sum swap rate, a point in favour of the constant product is output-boundedness, which allows users to add unbounded liquidity to the AMM. A point against is slippage, which grows linearly with the amount of the input token. Therefore, when the internal exchange rate is aligned with the oracle's, users are disincentivised from

[4]The existence of other classes of swap rate functions enjoying all the six properties is an open question.

performing large swaps. The most prominent AMM platform adopting the constant product is Uniswap v2 [uni21]. Curve [cur20] uses a hybrid swap rate function, which approximates a constant sum for an interval of input values x, and behaves as a constant product outside the interval. In this way, it achieves a small slippage within the interval, at the same time allowing unbounded liquidity thanks to output-boundedness.

5.8. **Constant mean swap rate.** The constant mean swap rate function generalises the constant product by associating weights $w_0, w_1 \in \mathbb{R}_{>0}$ to the token types held by the AMM, so to preserve the following equality:

$$r_0^{w_0} r_1^{w_1} = (r_0 + x)^{w_0} (r_1 + y)^{w_1} \qquad \text{where } y = x \cdot SX(x, r_0, r_1)$$

The following theorem shows that the constant mean function enjoys most of the properties of the constant product, except reversibility.

Theorem 5.16 (Constant mean swap rate)**.** *The* constant mean *swap rate function:*

$$SX(x, r_0, r_1) \;=\; \frac{r_1}{x}\left(1 - \left(\frac{r_0}{r_0 + x}\right)^{\frac{w_0}{w_1}}\right)$$

is output-bounded, monotonic, additive, and homogeneous. Furthermore, its internal swap rate and its slippage are given by:

$$X_\Gamma(\tau_0, \tau_1) \;=\; \frac{r_1 w_0}{r_0 w_1} \qquad\qquad \Delta X_\Gamma(x, \tau_0, \tau_1) \;=\; \frac{x w_0}{r_0 w_1\left(1 - \left(\frac{r_0}{r_0+x}\right)^{\frac{w_0}{w_1}}\right)} - 1$$

The most prominent AMM plaform using the constant mean swap rate is Balancer [bal19]. Users fix the weights w_0, w_1 of token types when an AMM is created; once fixed, these weights cannot be changed. The constant product swap rate can be seen as the special case of the constant mean where the two weights are equal.

6. The economic mechanism of AMMs

AMMs can be seen as games where users compete to increase their net worth. We now study the incentive mechanisms of AMMs from a game-theoretic perspective.

6.1. **Arbitrage.** The ***arbitrage game*** is a single-player, single-round game, where the player can perform a single move on a given AMM pair τ_0, τ_1 in order to maximize her gain. The initial game states have the form $\Gamma_0 = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$, where A is the player; the *moves* of A are all the possible transactions that can be fired by A (we also consider doing nothing as a possible move). More formally, a move is a sequence λ such that either $\lambda = \varepsilon$ (the empty sequence), or $\lambda = \mathsf{T}$ with $\mathit{wal}(\mathsf{T}) = \mathsf{A}$. The goal of A is to maximize her gain $G_{\mathsf{A}}(\Gamma_0, \lambda)$ on the AMM pair τ_0, τ_1. A *solution* to the game is a move λ that satisfies such goal. We study the arbitrage game under the assumption that A holds no minted tokens $\{\tau_0, \tau_1\}$. In this way, by Lemma 3.2, A's gain only depends on the input amount of A's swap, on the reserves of τ_0 and τ_1 in the AMM, and on their prices. In practice, AMM users are logically partitioned in two groups, e.g. liquidity providers (who perform deposits and redeems) and traders (who perform swaps), so basically here we are considering the arbitrage game from the traders' point of view. We further assume that A's balance is enough to allow A to perform the optimal swap. This is a common assumption in formulations of the arbitrage game: in practice, this can be achieved by borrowing the needed amount of the input token from a lending pool via a flash-loan [QZLG21, WWL+20]. Theorem 6.3 shows that a rational agent is incentivized to perform a swap to realign the internal and the oracle's exchange rate. The optimal solution to the arbitrage game can be approximated by multiple users who swap smaller amounts than the optimal one.

Before devising a solution to the arbitrage game, we examine the potential candidates for the solution. Observe that doing nothing (i.e., $\lambda = \varepsilon$) has clearly zero gain, as well as depositing or redeeming, as established by Lemma 4.5. Hence, if one of such moves is a solution, so are the other two: without loss of generality, we assume that A's move will be $\lambda = \varepsilon$ when there is no strategy which allows A to increase her gain.

We first show in Lemma 6.2 that, if a swap with input τ_0 and output τ_1 has a positive gain, then a swap with input τ_1 and output τ_0 will have a negative gain, whatever input amount is chosen. This holds whenever the swap rate function is monotonic and reversible. Lemma 6.1 is instrumental to prove Lemma 6.2, as it finds the needed relation between the swap rate function and the exchange rate. Passing from this relation to the gain of the swap transaction is obtained by means of Lemma 3.3.

Lemma 6.1. *If SX is strictly monotonic and reversible, then for all $x > 0$:*

$$SX(x, r_0, r_1) \geq X(\tau_0, \tau_1) \implies \forall y > 0.\ SX(y, r_1, r_0) < X(\tau_1, \tau_0)$$

Lemma 6.2 (Unique direction for swap gain)**.** *Let $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$ be such that $\sigma\{\tau_0, \tau_1\} = 0$, and let $\mathsf{T}_d(x) = \mathsf{A} : \mathsf{swap}(x, \tau_d, \tau_{1-d})$, for $x > 0$ and $d \in \{0, 1\}$. If SX is output-bounded, strictly monotonic and reversible, then for all $y > 0$ such that $\sigma\tau_{1-d} \geq y$:*

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}_d(x)) > 0 \implies G_{\mathsf{A}}(\Gamma, \mathsf{T}_{1-d}(y)) < 0$$

Theorem 6.3 devises a general solution to the arbitrage game, determining the swap transaction that maximizes A's gain. This is the transaction $\mathsf{A} : \mathsf{swap}(x_0, \tau_0, \tau_1)$ such that, in the state Γ' reached after performing it from the initial state, the internal exchange rate between τ_0 and τ_1 is aligned to the oracle's exchange rate. By Lemma 3.3, no move from Γ' can increase A's gain, i.e. the solution for the arbitrage game in Γ' is to do nothing. Lemma 6.2 guarantees that swaps in the other direction are not solutions, since they decrease A's gain. Note that if the internal exchange rate is already aligned to the oracle's, or if A has not enough balance to perform the optimal swap, then the solution to the arbitrage problem is to do nothing.

Theorem 6.3 (Arbitrage)**.** *Let $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$ be such that $\sigma\{\tau_0, \tau_1\} = 0$. For all $x > 0$, let $\mathsf{T}(x) = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$. Let x_0 be such that:*

$$X_{\Gamma'}(\tau_0, \tau_1) = X(\tau_0, \tau_1) \qquad \textit{where } \Gamma \xrightarrow{\mathsf{T}(x_0)} \Gamma' \tag{6.1}$$

If SX is output-bounded, strictly monotonic, additive and reversible, then:

$$\forall x \neq x_0 \ :\ G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) \ > \ G_{\mathsf{A}}(\Gamma, \mathsf{T}(x))$$

Furthermore, if an x_0 satisfying Equation (6.1) *exists, it is unique.*

An implicit desideratum on these solutions is that, given a specific instance of the swap rate function, they are efficiently computable: this is the case, e.g., for the constant product, for which Lemma 6.4 finds a closed formula for the arbitrage solution.

Lemma 6.4 (Arbitrage and constant product)**.** *Let $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\}$, and let:*

$$x_0 \ = \ \sqrt{\frac{P_{\tau_1}}{P_{\tau_0}} r_0 r_1} - r_0 \tag{6.2}$$

If SX is the constant product swap rate and $x_0 > 0$, then $\mathsf{A} : \mathsf{swap}(x_0, \tau_0, \tau_1)$ *is the solution to the arbitrage game in* Γ*.*

Example 6.5. Consider an initial state $\Gamma = \mathsf{A}[\sigma] \mid \{18 : \tau_0, 6 : \tau_1\} \mid \Delta$. Assuming the constant product swap rate, and $P\tau_0 = 3$, $P\tau_1 = 4$, we have that:

$$X_\Gamma(\tau_0, \tau_1) \;=\; 6/18 \;<\; 3/4 \;=\; X(\tau_0, \tau_1)$$

$$X_\Gamma(\tau_1, \tau_0) \;=\; 18/6 \;>\; 4/3 \;=\; X(\tau_1, \tau_0)$$

By Theorem 6.3 it follows that the solution to the arbitrage game is $\mathsf{T}(x) = \mathsf{A} : \mathsf{swap}(x, \tau_1, \tau_0)$, for suitable x. By Lemma 6.4, we find that the optimal input value is:

$$x_1 \;=\; \sqrt{\frac{3}{4} \cdot 18 \cdot 6} - 6 = 3$$

and the corresponding output value is $x_1 \cdot SX(x_1, 6, 18) = 6$. We then obtain:

$$\Gamma \xrightarrow{\mathsf{T}(x_1)} \Gamma' \;=\; \mathsf{A}[\sigma - 3 : \tau_1 + 6 : \tau_0] \mid \{12 : \tau_0, 9 : \tau_1\}$$

This action maximizes A's gain $G_\mathsf{A}(\Gamma, \mathsf{T}(x_1)) = W_\mathsf{A}(\Gamma') - W_\mathsf{A}(\Gamma) = 6P\tau_0 - 3P\tau_1 = 6$. Any other action would result in a lower gain for A. Note that the internal exchange rate in Γ' is aligned to the oracle's: $X_{\Gamma'}(\tau_0, \tau_1) = 9/12 = 3/4 = X(\tau_0, \tau_1)$. ⋄

6.2. **Swaps after deposits.** We show in Theorem 6.6 that deposits incentivise swaps. Namely, if a user B performs a deposit on an AMM for the token pair τ_0, τ_1, and then a *different* user A performs a swap in the resulting state, then A's gain is increased w.r.t. the gain that she would have obtained by performing the same transaction *before* B's deposit. The intuition is that larger amounts of tokens in an AMM provide decrease the slippage, therefore attracting users interested in swaps.

Theorem 6.6 (Swap after deposit)**.** *Let* $\mathsf{T}_{\mathsf{swap}}$ *and* $\mathsf{T}_{\mathsf{dep}}$ *be two transactions such that* $wal(\mathsf{T}_{\mathsf{swap}}) = \mathsf{A} \neq wal(\mathsf{T}_{\mathsf{dep}})$ *and, for* $\ell \in \{\mathsf{swap}, \mathsf{dep}\}$*,* $type(\mathsf{T}_\ell) = \ell$ *and* $tok(\mathsf{T}_\ell) = \{\tau_0, \tau_1\}$*. Let* Γ *be such that both* $\mathsf{T}_{\mathsf{swap}}$ *and* $\mathsf{T}_{\mathsf{dep}}\mathsf{T}_{\mathsf{swap}}$ *are enabled in* Γ*. If the swap rate function is homogeneous and strictly monotonic, then:*

$$G_\mathsf{A}(\Gamma, \mathsf{T}_{\mathsf{dep}}\mathsf{T}_{\mathsf{swap}}) > G_\mathsf{A}(\Gamma, \mathsf{T}_{\mathsf{swap}})$$

Example 6.7. Let $\Gamma = \mathsf{A}[5 : \tau_0] \mid \{5 : \tau_0, 10 : \tau_1\} \mid \Delta$, let $\mathsf{T}_{\mathsf{dep}} = \mathsf{B} : \mathsf{dep}(40 : \tau_0, 80 : \tau_1)$, and let $\mathsf{T}_{\mathsf{swap}} = \mathsf{A} : \mathsf{swap}(5, \tau_0, \tau_1)$. Assuming the constant product swap rate, we have that:

$$\Gamma \xrightarrow{\mathsf{T}_{\mathsf{swap}}} \Gamma_s = \mathsf{A}[5 : \tau_1] \mid \{10 : \tau_0, 5 : \tau_1\} \mid \Delta$$

$$\Gamma \xrightarrow{\mathsf{T}_{\mathsf{dep}}} \Gamma_d = \mathsf{A}[5 : \tau_0] \mid \{45 : \tau_0, 90 : \tau_1\} \mid \Delta' \xrightarrow{\mathsf{T}_{\mathsf{swap}}} \Gamma_{ds} = \mathsf{A}[9 : \tau_1] \mid \{50 : \tau_0, 81 : \tau_1\} \mid \Delta'$$

Now, assuming $P\tau_0 = 1$ and $P\tau_1 = 1$, we have the following gains for A:

$$G_\mathsf{A}(\Gamma, \mathsf{T}_{\mathsf{dep}}\mathsf{T}_{\mathsf{swap}}) \;=\; 4 \;>\; 0 \;=\; G_\mathsf{A}(\Gamma, \mathsf{T}_{\mathsf{swap}})$$

as correctly predicted by Theorem 6.6. Note that in the state Γ before the deposit, A has zero gain from her swap, while the same transaction has a positive gain after the deposit. ⋄

Theorem 6.8 finds the solution of the arbitrage game after a deposit of another user. More precisely, let λ be the solution in Γ, and let λ' be the solution in the state Γ' reached after a deposit. If λ is empty, then also λ' is such. If λ is a swap with input τ_0 and output τ_1, then also λ' is such (but for the input amount).

Theorem 6.8 (Arbitrage after deposit)**.** *Let* $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$*, and let:*

$$\Gamma \xrightarrow{\mathsf{B}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \Gamma_d \qquad \textit{where } \Gamma_d = \mathsf{A}[\sigma'] \mid \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta' \textit{ and } \mathsf{B} \neq \mathsf{A}$$

Let λ *and* λ_d *be the solutions of the arbitrage game in* Γ *and in* Γ_d*, respectively. If SX is output-bounded, strictly monotonic, additive, reversible, and homogeneous, then:*

(1) *if* $\lambda = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, then*

$$\lambda_d = \mathsf{A} : \mathsf{swap}(ax, \tau_0, \tau_1) \qquad G_{\mathsf{A}}(\Gamma_d, \lambda_d) = a\, G_{\mathsf{A}}(\Gamma, \lambda) \qquad \textit{where } \; a = \frac{r_1 + v_1}{r_1}$$

(2) *if* $\lambda = \varepsilon$*, then* $\lambda_d = \varepsilon$*.*

6.3. **Swaps after redeems.** We now study swaps and arbitrage after redeems. Conversely to what we have shown before in Theorem 6.6, we find that redeems disincentivise swaps (Theorem 6.9). Similarly to Theorem 6.8, if the solution to the arbitrage game in a state Γ is a swap, then after a redeem in Γ the solution is still a swap which only differs in the input amount (Theorem 6.10).

Theorem 6.9 (Swap after redeem)**.** *Let* $\mathsf{T}_{\mathsf{swap}}$ *and* $\mathsf{T}_{\mathsf{rdm}}$ *be two transactions such that* $wal(\mathsf{T}_{\mathsf{swap}}) = \mathsf{A} \neq wal(\mathsf{T}_{\mathsf{rdm}})$ *and, for* $\ell \in \{\mathsf{swap}, \mathsf{rdm}\}$*,* $type(\mathsf{T}_\ell) = \ell$ *and* $tok(\mathsf{T}_\ell) = \{\tau_0, \tau_1\}$*. Let* Γ *be such that both* $\mathsf{T}_{\mathsf{swap}}$ *and* $\mathsf{T}_{\mathsf{rdm}}\mathsf{T}_{\mathsf{swap}}$ *are enabled in* Γ*. If the swap rate function is homogeneous and strictly monotonic, then:*

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{rdm}}\mathsf{T}_{\mathsf{swap}}) < G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{swap}})$$

Theorem 6.10 (Arbitrage after redeem)**.** *Let* $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$*, and let:*

$$\Gamma \xrightarrow{\mathsf{B}:\mathsf{rdm}(v:\{\tau_0, \tau_1\})} \Gamma_d \qquad \textit{where } \Gamma_d = \mathsf{A}[\sigma'] \mid \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta' \textit{ and } \mathsf{B} \neq \mathsf{A}$$

Let λ *and* λ_d *be the solutions of the arbitrage game in* Γ *and in* Γ_d*, respectively. If SX is output-bounded, strictly monotonic, additive, reversible, and homogeneous, then:*

(1) *if* $\lambda = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$*, then*

$$\lambda_d = \mathsf{A} : \mathsf{swap}(ax, \tau_0, \tau_1) \qquad G_{\mathsf{A}}(\Gamma_d, \lambda_d) = a\, G_{\mathsf{A}}(\Gamma, \lambda) \qquad \textit{where } \; a = 1 - \frac{v}{S_\Gamma\{\tau_0, \tau_1\}}$$

(2) *if* $\lambda = \varepsilon$*, then* $\lambda_d = \varepsilon$*.*

7. Maximal extractable value

Maximal Extractable Value (MEV) refers to a class of attacks to smart contracts where miners/validators exploit their power to reorder, drop or insert transactions in a block to "extract" value from the *mempool* (i.e., the set of transactions sent to the blockchain network, but not appearing yet in a block). Empirical research has shown that AMMs are routinely targeted by MEV attacks [DGK+20,QZG21,ZQC+21,ZQT+21], and indeed recent versions of the Ethereum protocol implementation include a MEV extraction mechanism [mev22]. This has negative effects on AMM users, as well as on transaction fees and network congestion.

We show that our AMM model makes it possible to faithfully express MEV attacks. Consider a constant product AMM for two token types τ_0, τ_1 with the same price, e.g. $P\tau_0 = P\tau_1 = 1$, and consider a state:

$$\Gamma \;=\; \mathsf{M}[\cdots] \mid \mathsf{A}[50 : \tau_0] \mid \{10 : \tau_0, 10 : \tau_1\} \mid \cdots$$

where we use A to impersonate a honest user, and M for a miner, acting as an adversary. By Lemma 3.3 we know that the AMM is in equilibrium in Γ, because, for each $x > 0$:

$$SX(x, 10, 10) = \frac{10}{10 + x} < 1 = X(\tau_0, \tau_1)$$

Therefore, neither a miner nor any other user can increase their net worth in Γ.

Assume now that A sends the transaction $\mathsf{T}_{\mathsf{A}} = \mathsf{A} : \mathsf{swap}(50, \tau_0, \tau_1)$ to the blockchain network. Before being included in a block, T_{A} is added to the mempool, from where miners gather transactions to construct blocks. Any miner owning enough token units can increase their gain by firing A's transaction within a *sandwich* of M's swaps. For instance, assume that M's wallet is $\mathsf{M}[40 : \tau_0, 1 : \tau_1]$. Then M can construct a block:

$$\lambda \;=\; \mathsf{M} : \mathsf{swap}(40, \tau_0, \tau_1) \;\; \mathsf{T}_{\mathsf{A}} \;\; \mathsf{M} : \mathsf{swap}(9, \tau_1, \tau_0)$$

We have that $\Gamma \xrightarrow{\lambda} \Gamma'$, where:

$$\begin{aligned}
\Gamma \; & \xrightarrow{\mathsf{M}:\mathsf{swap}(40,\tau_0,\tau_1)} \mathsf{M}[0 : \tau_0, 9 : \tau_1] \mid \mathsf{A}[50 : \tau_0] \mid \{50 : \tau_0, 2 : \tau_1\} \mid \cdots \\
& \xrightarrow{\mathsf{A}:\mathsf{swap}(50,\tau_0,\tau_1)} \mathsf{M}[0 : \tau_0, 9 : \tau_1] \mid \mathsf{A}[0 : \tau_0, 1 : \tau_1] \mid \{100 : \tau_0, 1 : \tau_1\} \mid \cdots \\
& \xrightarrow{\mathsf{M}:\mathsf{swap}(9,\tau_1,\tau_0)} \mathsf{M}[90 : \tau_0, 0 : \tau_1] \mid \mathsf{A}[0 : \tau_0, 1 : \tau_1] \mid \{10 : \tau_0, 10 : \tau_1\} \mid \cdots \; = \Gamma'
\end{aligned}$$

This results in a positive gain for M, since:

$$\begin{aligned}
G_{\mathsf{M}}(\Gamma, \lambda) &= W_{\mathsf{M}}(\Gamma') - W_{\mathsf{M}}(\Gamma) = 90 \cdot P\tau_0 - (40 \cdot P\tau_0 + 1 \cdot P\tau_1) = 49 \\
G_{\mathsf{A}}(\Gamma, \lambda) &= W_{\mathsf{A}}(\Gamma') - W_{\mathsf{A}}(\Gamma) = 1 \cdot P\tau_1 - 50 \cdot P\tau_0 = -49
\end{aligned}$$

Summing up, M has managed to extract value from A's transaction in the mempool, improving her gain to the detriment of A's net worth.

The mechanism of *guarded transactions*, which allows users to specify a lower bound to the amount of tokens outputted upon a swap (see section 8), is a partial countermeasure against MEV attacks. For instance, in the scenario above A could have sent a guarded transaction $\mathsf{T}'_{\mathsf{A}} = \mathsf{A} : \mathsf{swap}(50 : \tau_0, 8.3 : \tau_1)$, which would have ensured A to receive at least $8.3 : \tau_1$ upon the swap. This would have neutralised the sandwich attack described before, since after the first M's transaction, T'_{A} is no longer valid. Even though guarded transactions mitigate the issue of not knowing the state where one's transaction will be fired, they are not a complete defence against MEV attacks. Indeed, in [BCL22] it is shown that adversaries can craft sandwiches that extract value from *any* non-empty mempool of swap and dep (guarded) transactions. Further analyses the effect of MEV on constant-function AMMs are developed in [KDC22]. Several approaches to prevent MEV attacks are discussed in [HW22, ByCD+21].

8. Variants of the basic model

Our AMM model abstracts from implementation-specific features, and from the features that are orthogonal to the core functionality of AMMs (e.g., governance). We discuss below some extensions and variants of our model to make it closer to actual implementations, and their impact on our theory.

8.1. **Fees.** In actual AMM implementations, the swap rate — and consequently, the semantics of [SWAP] actions — also depends on a *trading fee* $1 - \phi$. For instance, incorporating this fee in the constant product swap rate function is usually done as follows:

$$SX_{\phi}(x, r_0, r_1) \;=\; \frac{\phi\, r_1}{r_0 + \phi\, x} \qquad \text{where } \phi \in [0,1]$$

In this case, when the trading fee is zero (i.e., $\phi = 1$), the swap rate preserves the product between AMM reserves; a higher fee, instead, results in reduced amounts of output tokens received from swap actions. Intuitively, the AMM retains a portion of the swapped amounts, but the overall reserves are still distributed among all minted tokens, thereby increasing the redeem rate of minted tokens. The structural properties in section 4 are not affected by swap fees.

8.2. **Price updates.** An underlying assumption of our model is that the price of atomic tokens is constant, and consequently that exchange rates are stable. In the wild, prices and exchange rates can vary over time, possibly making the net worth of users holding minted tokens decrease — a phenomenon commonly referred to as *impermanent loss* [imp20].

Introducing price updates in our AMM model is straightforward: it suffices to extend states Γ with price oracles, parameterize with Γ the exchange rate X, and extend the AMM semantics with a rule to non-deterministically update token prices. Most of the structural properties in section 4 would not be affected by this extension: the exceptions are determinism (Lemma 4.1) and net worth preservation (Lemma 4.5(b), while part (a) would still be true for deposits and redeems). Technically, also the properties about swaps and incentives in section 5 and section 6 are preserved, although this happens because most of these properties assume sequences of deposits, redeems and swaps. If we allow these actions to be interleaved with price updates, some properties no longer hold: notably, the optimality of the solution λ to the arbitrage problem (Theorem 6.3) is lost if λ is front-run by a price update that alters the exchange rates, since this affects the condition provided by Theorem 6.3.

In practice, the assumption of constant exchange rates assumed by Theorem 6.3 may hold in the case of exchanges between stable coins [mak20]. Here, arbitrage ensures the alignment between swap rates and exchange rates, so users are hence incentivized to provide liquidity to AMMs, as the redeem rate is likely to increase over time.

8.3. **Guarded transactions.** The semantics of AMMs in section 2 defines how the state evolves upon transactions. In practice, when a user emits a transaction, she cannot predict the exact state in which it will be actually committed. This may lead to unexpected or unwanted behaviours. For instance, the gain of a swap transaction sent by A may be reduced if the transaction is front-run by a redeem transaction sent by B, as established by Theorem 6.9. The problem here is that redeems decrease the swap rate (by Lemma 5.13), and consequently the amount of output tokens received by A. As a partial countermeasure to this issue, Uniswap allows users to specify a lower bound y^{min} to the amount of received tokens. In our model, we could formalise this behaviour by amending the [SWAP] rule as follows:

$$\frac{\sigma\tau_0 \geq x > 0 \qquad y = x \cdot SX(x, r_0, r_1) \qquad y^{min} \leq y < r_1}{\mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}(x:\tau_0, y^{min}:\tau_1)} \mathsf{A}[\sigma - x : \tau_0 + y : \tau_1] \mid \{r_0 + x : \tau_0, r_1 - y : \tau_1\} \mid \Gamma} \text{[Swap]}$$

Similar countermeasures apply to [Rdm] and [Dep] rules. For redeems, the user can enforce lower bounds v_0^{min}, v_1^{min} on the amount of received tokens τ_0, τ_1 as follows:

$$\frac{\sigma\{\tau_0, \tau_1\} \geq v > 0 \qquad v < S_\Gamma\{\tau_0, \tau_1\} \qquad v_i = v \cdot RX^i_\Gamma(\tau_0, \tau_1) \qquad v_i^{min} \leq v_i}{\Gamma \;=\; \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma' \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:\{\tau_0,\tau_1\}, v_0^{min}:\tau_0, v_1^{min}:\tau_1)} \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : \{\tau_0, \tau_1\}] \mid \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \mid \Gamma'} \text{[Rdm]}$$

Amending the [Dep] rule is more complex, since here we must define ranges for the deposited amounts v_0, v_1, and we must preserve the ratio between the AMM reserves. A possible way to achieve this behaviour is the following rule:

$$\frac{\sigma\tau_i \geq v_i > 0 \quad v = \frac{v_i}{RX^i_\Gamma(\tau_0,\tau_1)} \quad (v_0, v_1) = \begin{cases} (v_0^{max}, v_0^{max} \cdot \frac{r_1}{r_0}) & \text{if } v_1^{min} \leq v_0^{max} \cdot \frac{r_1}{r_0} \leq v_1^{max} \\ (v_1^{max} \cdot \frac{r_0}{r_1}, v_1^{max}) & \text{if } v_0^{min} \leq v_1^{max} \cdot \frac{r_0}{r_1} \leq v_0^{max} \end{cases}}{\Gamma \;=\; \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma' \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0^{min}, v_0^{max}:\tau_0, v_1^{min}, v_1^{max}:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : \{\tau_0, \tau_1\}] \mid \{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\} \mid \Gamma'} \text{[Dep]}$$

These amendments, which are coherent with Uniswap implementation [uni21], preserve all the properties, both structural and economic, established in the previous sections, modulo a restatement of the properties which have transactions in their hypotheses. For instance, in Theorem 6.8, the scaling factor a will be computed on the actual deposited value, rather than on the parameter of the transaction. Note that, although the new rules can disable some transactions which were enabled with the rules in section 2, this does not affect the transactions reordering result (Lemma 4.9).

8.4. **Other variants.** There are further differences between our model and the existing AMM platforms, that could be accounted for in extensions of our model. Uniswap implements flash-loans as part of the swap actions: namely, the user can optionally borrow available pair funds [uni20a] whilst returning these within the same *atomic group* of actions. Further, Uniswap implements an exchange rate oracle, allowing smart contracts to interpret (averages of) recent swap rates as exchange rates [uni20b]. Balancer [bal19] extends token pairs to token *tuples*: a user can swap any two non-coinciding sets of supported tokens, such that the swap rate is maintained. In all AMM implementations, token balances are represented as integers: consequently, they are subject to rounding errors [rva18]. AMM platforms frequently implement a governance logic, which allow "governance token" holders to coordinate changes to AMM fee-rates or swap rate parameters.

9. Conclusions

We have proposed a theory of AMMs, which encompasses and generalizes the main functional and economic aspects of the mainstream AMM implementations, providing solid grounds for the design of future AMMs.

The core of our theory is a formal model of AMMs (section 2), based on a thorough inspection of leading AMM implementations like Uniswap [uni21], Curve [cur21b], and Balancer [bal19]. An original aspect of our model is that it is parametric with respect to the key economic mechanism — the *swap rate function* — that algorithmically determines exchange rates between tokens. Our model features an *executable semantics*, which can support future implementations and analysis tools; an open-source implementation of our semantics is available as a companion of this paper.

Building upon our model, we prove a set of properties characterizing both structural (section 4) and economic (section 3, section 6) aspects of AMMs. Structural properties include, e.g., that value cannot be created or destroyed (Lemma 4.5), that tokens cannot be frozen within an AMM (Lemma 4.8) and that some sequences of transactions can be reordered without affecting their semantics (Lemma 4.9). Concerning the economic properties, we address the *arbitrage problem*, the main game-theoretic foundation behind the economic incentives of AMMs. Theorem 6.3 provides sufficient conditions for the existence of solutions, and links the solutions to the expected relation between internal exchange rate and oracle's exchange rate. We show that deposits incentivize swaps, while redeems have the opposite effect. With respect to previous works, which focus on specific economic mechanisms, all our results are parametric with respect to the swap rate function. We identify indeed, for each property, a set of conditions on swap rate functions that are sufficient for the property to hold (section 5).

AMM platforms like Uniswap [uni21] and Curve [Ego19] have overtaken centralized cryptocurrency markets in size and usage. On the one hand, a better understanding of AMM design in cases where AMMs host the majority of the token's global swap volume is critical [AEC20]. On the other hand, the growth of AMMs is making them more attractive for malicious users, even if it is difficult to exactly quantify the effect of attacks.

This paper, together with our work on formalizing another DeFi archetype called *lending pool* [BCL21a], is the first step towards a general theory of DeFi [BCL21c]. We believe that a general theory encompassing interactions between different DeFi archetypes is crucial to be able to reason about their structural, economic and security aspects, as typical DeFi applications operate within a wider ecosystem, composed by a set of collaborating or competing agents, which interact through possibly separate execution environments.

Acknowledgements. Massimo Bartoletti is partially supported by Conv. Fondazione di Sardegna & Atenei Sardi project F75F21001220007 *ASTRID*. James Hsin-yu Chiang is supported by the PhD School of DTU Compute. Alberto Lluch Lafuente is partially supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). We thank the anonymous reviewers of COORDINATION 2021 and LMCS, and Elvis Sikora for their insightful comments on preliminary versions of this paper.

References

[AC20] Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *ACM Conference on Advances in Financial Technologies (AFT)*, pages 80–91. ACM, 2020. `https://arxiv.org/abs/2003.10001`. `doi:10.1145/3419614.3423251`.

[AEC20] Guillermo Angeris, Alex Evans, and Tarun Chitra. When does the tail wag the dog? Curvature and market making. *arXiv preprint arXiv:2012.08040*, 2020. URL: `https://arxiv.org/abs/2012.08040`.

[AKC$^+$21] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An analysis of Uniswap markets. *Cryptoeconomic Systems*, 1(1), 2021. doi:10.21428/58320208.c9738e64.

[bal19] Balancer whitepaper, 2019. https://balancer.finance/whitepaper/.

[BCL21a] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. SoK: Lending pools in decentralized finance. In *Financial Cryptography Workshops*, volume 12676 of *LNCS*, pages 553–578. Springer, 2021. doi:10.1007/978-3-662-63958-0_40.

[BCL21b] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A Theory of Automated Market Makers in DeFi. In *Coordination Models and Languages*, volume 12717 of *LNCS*, pages 168–187. Springer, 2021. doi:10.1007/978-3-030-78142-2_11.

[BCL21c] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Towards a theory of decentralized finance. In *Financial Cryptography Workshops*, volume 12676 of *LNCS*, pages 227–232. Springer, 2021. doi:10.1007/978-3-662-63958-0_20.

[BCL22] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing extractable value from Automated Market Makers. In *Financial Cryptography*, volume 13411 of *LNCS*, pages 3–19. Springer, 2022. doi:10.1007/978-3-031-18283-9_1.

[Ben73] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17:525–532, November 1973.

[ByCD$^+$21] Carsten Baum, James Hsin yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. SoK: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Report 2021/1628, 2021. https://ia.cr/2021/1628.

[CAE22] Tarun Chitra, Guillermo Angeris, and Alex Evans. Differential privacy in constant function market makers. 13411:149–178, 2022. doi:10.1007/978-3-031-18283-9_8.

[cur20] Curve website, 2020. URL: https://www.curve.fi.

[cur21a] Curve computation of invariant constant, 2021. https://github.com/curvefi/curve-contract/blob/a1b5a797790d3f5ef12b0e358892a0ce47c12f85/contracts/pool-templates/base/SwapTemplateBase.vy#L206.

[cur21b] Curve token pair implementation, 2021. https://github.com/curvefi/curve-contract/blob/a1b5a797790d3f5ef12b0e358892a0ce47c12f85/contracts/pool-templates/base/SwapTemplateBase.vy.

[cur22] Curve statistics, 2022. https://www.curve.fi/dailystats.

[def22] Documented timeline of exchange hacks, 2022. https://cryptosec.info/exchange-hacks/.

[DGK$^+$20] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE Symposium on Security and Privacy*, pages 910–927. IEEE, 2020. doi:10.1109/SP40000.2020.00040.

[DKP21] Vincent Danos, Hamza El Khalloufi, and Julien Prat. Global order routing on exchange networks. In *Financial Cryptography Workshops*, volume 12676 of *LNCS*, pages 207–226. Springer, 2021. doi:10.1007/978-3-662-63958-0_19.

[EAC21] Alex Evans, Guillermo Angeris, and Tarun Chitra. Optimal fees for geometric mean market makers. In *Financial Cryptography Workshops*, volume 12676 of *LNCS*, pages 65–79. Springer, 2021. doi:10.1007/978-3-662-63958-0_6.

[Ego19] Michael Egorov. Stableswap - efficient mechanism for stablecoin, 2019. https://curve.fi/files/stableswap-paper.pdf.

[EMC20] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In *Financial Cryptography*, pages 170–189, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-43725-1_13.

[HW22] Lioba Heimbach and Roger Wattenhofer. SoK: Preventing transaction reordering manipulations in decentralized finance. *CoRR*, abs/2203.11520, 2022. arXiv:2203.11520, doi:10.48550/arXiv.2203.11520.

[imp20] Uniswap Documentation: Understanding Returns, 2020. https://uniswap.org/docs/v2/advanced-topics/understanding-returns/.

[KDC22] Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. Towards a Theory of Maximal Extractable Value I: Constant Function Market Makers. *CoRR*, abs/2207.11835, 2022. arXiv:2207.11835, doi:10.48550/arXiv.2207.11835.

[KFG21] Bhaskar Krishnamachari, Qi Feng, and Eugenio Grippo. Dynamic curves for decentralized autonomous cryptocurrency exchanges. In *International Symposium on Foundations and Applications of Blockchain (FAB)*, volume 92 of *OASIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/OASIcs.FAB.2021.5.

[mak20] Makerdao website, 2020. https://https://makerdao.com.

[mev22] MEV-geth, 2022. https://github.com/flashbots/mev-geth.

[moo20a] Mooniswap implementation, 2020. https://github.com/1inch-exchange/mooniswap/blob/02dccfab2ddbb8a409400288cb13441763370350/contracts/Mooniswap.sol.

[moo20b] Mooniswap whitepaper, 2020. https://mooniswap.exchange/docs/MooniswapWhitePaper-v1.0.pdf.

[MSG$^+$20] Claudio Antares Mezzina, Rudolf Schlatte, Robert Glück, Tue Haulund, James Hoey, Martin Holm Cservenka, Ivan Lanese, Torben Æ. Mogensen, Harun Siljak, Ulrik Pagh Schultz, and Irek Ulidowski. Software and reversible systems: A survey of recent activities. In *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, volume 12070 of *LNCS*, pages 41–59. Springer, 2020. doi:10.1007/978-3-030-47361-7_2.

[mSt20] mStable — introducing constant sum bonding curves for tokenised assets, 2020. https://medium.com/mstable/introducing-constant-sum-bonding-curves-for-tokenised-assets-6e18879cdc5b.

[QZA$^+$21] Kaihua Qin, Liyi Zhou, Yaroslav Afonin, Ludovico Lazzaretti, and Arthur Gervais. CeFi vs. DeFi - comparing centralized to decentralized finance. *CoRR*, abs/2106.08157, 2021. URL: https://arxiv.org/abs/2106.08157, arXiv:2106.08157.

[QZG21] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? 2021. URL: https://arxiv.org/abs/2101.05511, arXiv:2101.05511.

[QZLG21] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the DeFi ecosystem with flash loans for fun and profit. In *Financial Cryptography*, volume 12674 of *LNCS*, pages 3–32. Springer, 2021. doi:10.1007/978-3-662-64322-8_1.

[rva18] Formal specification of constant product market maker model & implementation, 2018. https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf.

[sus21] SushiSwap token pair implementation, 2021. https://github.com/sushiswap/sushiswap/blob/94ea7712daaa13155dfab9786aacf69e24390147/contracts/uniswapv2/UniswapV2Pair.sol.

[uni20a] Uniswap flash loan implementation, 2020. https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol#L172.

[uni20b] Uniswap oracle template, 2020. https://github.com/Uniswap/uniswap-v2-periphery/blob/dda62473e2da448bc9cb8f4514dadda4aeede5f4/contracts/examples/ExampleOracleSimple.sol.

[uni21] Uniswap token pair implementation, 2021. https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol.

[uni22] Uniswap statistics, 2022. https://info.uniswap.org.

[vir18] Improving frontrunning resistance of x*y=k market makers, 2018. https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281.

[WPG$^+$21] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2021. arXiv:2101.08778.

[WWL$^+$20] Dabao Wang, Siwei Wu, Ziling Lin, Lei Wu, Xingliang Yuan, Yajin Zhou, Haoyu Wang, and Kui Ren. Towards understanding flash loan and its applications in DeFi ecosystem. *arXiv preprint arXiv:2010.12252*, 2020. https://arxiv.org/abs/2010.12252.

[XVPC22] Jiahua Xu, Nazariy Vavryk, Krzysztof Paruch, and Simon Cousaert. Sok: Decentralized exchanges (DEX) with automated market maker (AMM) protocols. *ACM Comput. Surv.*, nov 2022. doi:10.1145/3570639.

[ZQC$^+$21] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in DeFi protocols. In *IEEE Symp. on Security and Privacy*, pages 919–936. IEEE, 2021. doi:10.1109/SP40001.2021.00113.

[ZQT$^+$21] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-Frequency Trading on Decentralized On-Chain Exchanges. In *IEEE Symp. on Security and Privacy*, pages 428–445. IEEE, 2021. doi:10.1109/SP40001.2021.00027.

Appendix A. Proofs for Section 3

Proof of Lemma 3.2. Let Γ and T be as in the hypotheses, let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$, and let $y = x \cdot SX(x, r_0, r_1)$. By definition of gain (Equation 3.6), we have that:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}) \;=\; W_{\mathsf{A}}(\Gamma') - W_{\mathsf{A}}(\Gamma)$$

By definition of net worth (Equation 3.5), we have that:

$$\begin{aligned}
W_{\mathsf{A}}(\Gamma) &= \sigma_{\mathsf{A}}(\tau_0) \cdot P\tau_0 \;+\; \sigma_{\mathsf{A}}(\tau_1) \cdot P\tau_1 \\
&\quad + \sigma_{\mathsf{A}}\{\tau_0, \tau_1\} \cdot \frac{r_0 \cdot P\tau_0 + r_1 \cdot P\tau_1}{S_\Gamma\{\tau_0, \tau_1\}} \\
&\quad + \textstyle\sum_{\tau \notin \{\tau_0, \tau_1, \{\tau_0, \tau_1\}\}} \sigma_{\mathsf{A}}(\tau) \cdot P_\Gamma \tau \\
W_{\mathsf{A}}(\Gamma') &= (\sigma_{\mathsf{A}}(\tau_0) - x) \cdot P\tau_0 \;+\; (\sigma_{\mathsf{A}}(\tau_1) + y) \cdot P\tau_1 \\
&\quad + \sigma_{\mathsf{A}}\{\tau_0, \tau_1\} \cdot \frac{(r_0 + x) \cdot P\tau_0 + (r_1 - y) \cdot P\tau_1}{S_{\Gamma'}\{\tau_0, \tau_1\}} \\
&\quad + \textstyle\sum_{\tau \notin \{\tau_0, \tau_1, \{\tau_0, \tau_1\}\}} \sigma_{\mathsf{A}}(\tau) \cdot P_{\Gamma'} \tau
\end{aligned}$$

Since $S_\Gamma\{\tau_0, \tau_1\} = S_{\Gamma'}\{\tau_0, \tau_1\}$ and $P_\Gamma \tau = P_{\Gamma'} \tau$ for all $\tau \neq \{\tau_0, \tau_1\}$:

$$\begin{aligned}
W_{\mathsf{A}}(\Gamma') - W_{\mathsf{A}}(\Gamma) &= y \cdot P\tau_1 - x \cdot P\tau_0 + \sigma_{\mathsf{A}}\{\tau_0, \tau_1\} \frac{x \cdot P\tau_0 - y \cdot P\tau_1}{S_\Gamma\{\tau_0, \tau_1\}} \\
&= \big(y \cdot P\tau_1 - x \cdot P\tau_0\big)\Big(1 - \frac{\sigma_{\mathsf{A}}\{\tau_0, \tau_1\}}{S_\Gamma\{\tau_0, \tau_1\}}\Big) \\
&= x \cdot \big(SX(x, r_0, r_1)\, P\tau_1 - P\tau_0\big)\Big(1 - \frac{\sigma_{\mathsf{A}}\{\tau_0, \tau_1\}}{S_\Gamma\{\tau_0, \tau_1\}}\Big)
\end{aligned}$$

Using similar calculations, for $\mathsf{B} \neq \mathsf{A}$, we obtain:

$$G_{\mathsf{B}}(\Gamma, \mathsf{T}) = \sigma_{\mathsf{B}}\{\tau_0, \tau_1\} \frac{x \cdot P\tau_0 - y \cdot P\tau_1}{S_\Gamma\{\tau_0, \tau_1\}}$$

□

Proof of Lemma 3.3. Let $y = x \cdot SX(x, r_0, r_1)$. Since $\sigma\{\tau_0, \tau_1\} = 0$, by Lemma 3.2 we have that:

$$\begin{aligned}
G_{\mathsf{A}}(\Gamma, \mathsf{T}) \circ 0 &\iff y\ P\tau_1 - x\ P\tau_0 \circ 0 \\
&\iff \frac{y}{x} \circ \frac{P\tau_0}{P\tau_1} \\
&\iff SX(x, r_0, r_1) \circ X(\tau_0, \tau_1)
\end{aligned}$$

□

Appendix B. Proofs for Section 4

Proof of Lemma 4.1. Straightforward inspection of the rules [Dep0], [Dep], [Rdm], [Swap] in section 2. □

Proof of Lemma 4.2. For item (a), we proceed by induction on the length of a computation $\Gamma_0 \rightarrow^* \Gamma$, where Γ_0 is initial. The base case (computation of zero steps) is trivial, since initial states does not contain AMMs. For the inductive case, note that rule [Dep0] requires that the initial reserves of an AMM are strictly greater than zero. The rules that decrease the token reserves in AMMs, i.e. [Rdm] and [Swap], have premises that ensure that the reserves cannot be zeroed.

For item (b), we proceed by induction on the length of a computation $\Gamma_0 \rightarrow^* \Gamma$, where Γ_0 is initial. The base case is trivial, since initial states do not contain AMMs. For the inductive case, we assume that Γ satisfies the property, and we prove that it is preserved by a transition $\Gamma \rightarrow \Gamma'$. Assume that Γ' contains an AMM $\{r'_0 : \tau_0, r'_1 : \tau_1\}$. By item ((a)), $r'_0 > 0$ and $r'_1 > 0$. There are the following cases, depending on the rule used to infer $\Gamma \rightarrow \Gamma'$:

- [Dep0], [Dep]. Trivial, because deposits can only increase the supply of minted tokens.
- [Swap]. Trivial, because swap actions do not affect the supply of minted tokens.
- [Rdm]. Assume that $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$. By contradiction, suppose that the [Rdm] action burns all the supply of the minted token, i.e. it burns $v = S_\Gamma\{\tau_0, \tau_1\}$ units. The rule premise requires $v > 0$, and it implies:

$$r'_0 \;=\; r_0 - v\frac{r_0}{S_\Gamma\{\tau_0,\tau_1\}} \;=\; 0 \qquad\qquad r'_1 \;=\; r_1 - v\frac{r_1}{S_\Gamma\{\tau_0,\tau_1\}} \;=\; 0$$

  Therefore, we would have $r'_0 = r'_1 = 0$ — contradiction.

□

Proof of Lemma 4.3. By cases on the rule used in the transition $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. It is straightforward to check that, in all the rules, the changes applied to atomic tokens cancel out. Further, the [Swap] rule does not affect the supply of minted tokens. □

Proof of Lemma 4.4. Let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$, where $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$ and $\{r'_0 : \tau_0, r'_1 : \tau'_1\} \in \Gamma'$. If $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, then by the [Dep] rule it must be $r'_i = r_i + v_i$ for $i \in \{0, 1\}$. Furthermore, by the premises of [Dep], we obtain:

$$r_1 v_0 = r_1 v \cdot RX^0_\Gamma(\tau_0, \tau_1) = v \cdot \frac{r_0 r_1}{S_\Gamma\{\tau_0,\tau_1\}} = r_0 v \cdot RX^1_\Gamma(\tau_0,\tau_1) = r_0 v_1$$

Therefore:

$$\frac{r_1 + v_1}{r_0 + v_0} \;=\; \frac{(\frac{r_0 v_1}{v_0}) + v_1}{r_0 + v_0} \;=\; \frac{r_0 v_1 + v_0 v_1}{(r_0 + v_0) v_0} \;=\; \frac{(r_0 + v_0) v_1}{(r_0 + v_0) v_0} \;=\; \frac{v_1}{v_0} \;=\; \frac{r_1}{r_0} \tag{B.1}$$

If $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$, then by rule [Rdm] it must be, for $i \in \{0, 1\}$:

$$r'_i \;=\; r_i - v_i \;=\; r_i - v RX^i_\Gamma(\tau_0,\tau_1) \;=\; r_i - v\frac{r_i}{S_\Gamma\{\tau_0,\tau_1\}}$$

Therefore, since $S_\Gamma\{\tau_0,\tau_1\} = S_{\Gamma'}\{\tau_0,\tau_1\} + v$:

$$\frac{r_1 - v_1}{r_0 - v_0} \;=\; \frac{r_1 - v\frac{r_1}{S_\Gamma\{\tau_0,\tau_1\}}}{r_0 - v\frac{r_0}{S_\Gamma\{\tau_0,\tau_1\}}} \;=\; \frac{r_1(S_{\Gamma'}\{\tau_0,\tau_1\} + v) - v r_1}{r_0(S_{\Gamma'}\{\tau_0,\tau_1\} + v) - v r_0} \;=\; \frac{r_1}{r_0} \tag{B.2}$$

Summing up, (B.1) and (B.2) give item (a).

For item (b), if $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, then by the [Dep] rule it must be $r'_i = r_i + v_i$ for $i \in \{0, 1\}$, and $S_{\Gamma'}\{\tau_0, \tau_1\} = S_\Gamma\{\tau_0, \tau_1\} + \frac{v_i}{r_i} S_\Gamma\{\tau_0, \tau_1\}$. Therefore:

$$RX^i_{\Gamma'}(\tau_0, \tau_1) = \frac{r_i + v_i}{S_{\Gamma'}\{\tau_0, \tau_1\}} = \frac{r_i + v_i}{S_\Gamma\{\tau_0, \tau_1\}(1 + \frac{v_i}{r_i})} = \frac{(r_i + v_i) r_i}{S_\Gamma\{\tau_0, \tau_1\}(r_i + v_i)} = RX^i_\Gamma(\tau_0, \tau_1)$$

Otherwise, if $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$, then by rule [Rdm] it must be, for $i \in \{0, 1\}$:

$$r'_i \;=\; r_i - v_i \;=\; r_i - v RX^i_\Gamma(\tau_0, \tau_1) \;=\; r_i - v \frac{r_i}{S_\Gamma\{\tau_0, \tau_1\}}$$

Therefore:

$$\begin{aligned} RX^i_{\Gamma'}(\tau_0, \tau_1) &= \frac{r_i - v_i}{S_\Gamma\{\tau_0, \tau_1\} - v} = \frac{r_i - v \frac{r_i}{S_\Gamma\{\tau_0, \tau_1\}}}{S_\Gamma\{\tau_0, \tau_1\} - v} = \frac{r_i S_\Gamma\{\tau_0, \tau_1\} - v r_i}{(S_\Gamma\{\tau_0, \tau_1\} - v) S_\Gamma\{\tau_0, \tau_1\}} \\ &= \frac{r_i}{S_\Gamma\{\tau_0, \tau_1\}} = RX^i_\Gamma(\tau_0, \tau_1) \end{aligned}$$

For item (c), if $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, we have that:

$$\begin{aligned} P_{\Gamma'}\{\tau_0, \tau_1\} &= \frac{r'_0 \cdot P\tau_0 + r'_1 \cdot P\tau_1}{S_{\Gamma'}\{\tau_0, \tau_1\}} && \text{by Equation (3.1)} \\ &= \frac{(1 + \frac{v_0}{r_0}) \cdot r_0 \cdot P\tau_0 + (1 + \frac{v_1}{r_1}) \cdot r_1 \cdot P\tau_1}{S_\Gamma\{\tau_0, \tau_1\} + \frac{v_i}{r_i} S_\Gamma\{\tau_0, \tau_1\}} \\ &= \frac{(1 + \frac{v_i}{r_i}) \cdot r_0 \cdot P\tau_0 + (1 + \frac{v_i}{r_i}) \cdot r_1 \cdot P\tau_1}{(1 + \frac{v_i}{r_i}) \cdot S_\Gamma\{\tau_0, \tau_1\}} && \text{since } \tfrac{v_0}{r_0} = \tfrac{v_1}{r_1} \\ &= P_\Gamma\{\tau_0, \tau_1\} && \text{by Equation (3.1)} \end{aligned}$$

The proof for the case $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$ is similar. $\square$

Proof of Lemma 4.5. Let $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. We first prove item (a). Depending on the rule used to fire the transition, we have the following cases:

- [Dep0]. Let $\mathsf{T} = \mathsf{B} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$. We have that:

$$\begin{aligned} \Gamma &= \mathsf{B}[\sigma] \mid \Gamma_0 \\ \Gamma' &= \mathsf{B}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : \{\tau_0, \tau_1\}] \mid \{v_0 : \tau_0, v_1 : \tau_1\} \mid \Gamma_0 \end{aligned}$$

  If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected. Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned} W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) - v_0 P\tau_0 - v_1 P\tau_1 + v_0 P_\Gamma\{\tau_0, \tau_1\} \\ &= W_{\mathsf{A}}(\Gamma) - v_0 P\tau_0 - v_1 P\tau_1 + v_0 \frac{v_0 P\tau_0 + v_1 P\tau_1}{v_0} && \text{by Equation (3.1)} \\ &= W_{\mathsf{A}}(\Gamma) \end{aligned}$$

- [Dep]. Let $\mathsf{T} = \mathsf{B} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$. We have that:

$$\begin{aligned} \Gamma &= \mathsf{B}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma_0 \\ \Gamma' &= \mathsf{B}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : \{\tau_0, \tau_1\}] \mid \{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\} \mid \Gamma_0 \end{aligned}$$

  where:

$$v = \frac{v_0 \cdot S_\Gamma\{\tau_0, \tau_1\}}{r_0}$$

If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected (note that the value of minted tokens in A's wallet is preserved by deposits, by Lemma 4.4(c)). Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned}
W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) - v_0 P\tau_0 - v_1 P\tau_1 + v P_\Gamma\{\tau_0, \tau_1\} \\
&= W_{\mathsf{A}}(\Gamma) - v_0 P\tau_0 - v_1 P\tau_1 + v \frac{r_0 P\tau_0 + r_1 P\tau_1}{S_\Gamma\{\tau_0, \tau_1\}} && \text{by Equation (3.1)} \\
&= W_{\mathsf{A}}(\Gamma) - v_0 P\tau_0 - v_1 P\tau_1 + \frac{v_0}{r_0}\Big(r_0 P\tau_0 + r_1 P\tau_1\Big) \\
&= W_{\mathsf{A}}(\Gamma) - v_1 P\tau_1 + \frac{v_0}{r_0} r_1 P\tau_1 \\
&= W_{\mathsf{A}}(\Gamma) && \text{since } r_1 v_0 = r_0 v_1
\end{aligned}$$

- [Swap]. This case cannot happen, since we are assuming $\mathit{type}(\mathsf{T}) \neq \mathsf{swap}$.
- [Rdm]. Let $\mathsf{T} = \mathsf{B} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$. We have that:

$$\begin{aligned}
\Gamma &= \mathsf{B}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Gamma_0 \\
\Gamma' &= \mathsf{B}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : \{\tau_0, \tau_1\}] \mid \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \mid \Gamma_0
\end{aligned}$$

where:

$$v_0 = \frac{v \cdot r_0}{s} \qquad v_1 = \frac{v \cdot r_1}{s} \qquad s = S_\Gamma\{\tau_0, \tau_1\}$$

If $\mathsf{B} \neq \mathsf{A}$, then A's net worth is unaffected (note that the value of minted tokens in A's wallet is preserved by redeems, by Lemma 4.4(c)). Otherwise, if $\mathsf{B} = \mathsf{A}$, then:

$$\begin{aligned}
W_{\mathsf{A}}(\Gamma') &= W_{\mathsf{A}}(\Gamma) + v_0 P\tau_0 + v_1 P\tau_1 - v P_\Gamma\{\tau_0, \tau_1\} \\
&= W_{\mathsf{A}}(\Gamma) + v_0 P\tau_0 + v_1 P\tau_1 - \frac{v \cdot r_0}{s} P\tau_0 - \frac{v \cdot r_1}{s} P\tau_1 \\
&= W_{\mathsf{A}}(\Gamma) + \frac{v \cdot r_0}{s} P\tau_0 + \frac{v \cdot r_1}{s} P\tau_1 - \frac{v \cdot r_0}{s} P\tau_0 - \frac{v \cdot r_1}{s} P\tau_1 \\
&= W_{\mathsf{A}}(\Gamma)
\end{aligned}$$

We now prove item (b), i.e. that the *global* net worth is preserved by *any* transactions. First, we recall from section 3 the definition of global net worth. Let:

$$\Gamma \;=\; \mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid \{r_1 : \tau_1, r'_1 : \tau'_1\} \mid \cdots \mid \{r_k : \tau_k, r'_k : \tau'_k\}$$

Then, the global net worth of Γ is:

$$W(\Gamma) \;=\; \sum_{i=1}^{n} W_{\mathsf{A}_i}(\Gamma)$$

We have the following cases:

- [Dep0], [Dep], [Rdm]. These rules affect the token reserves in AMMs, which do not contribute to the global net worth, and the balances of users, which we know to be preserved. Therefore, the global net worth is preserved.
- [Swap]. Let $\mathsf{A} : \mathsf{swap}(v, \tau_0, \tau_1)$ be the fired transaction. We have that:

$$\begin{aligned}
\Gamma &= \mathsf{A}[\sigma] \;\mid\; \{r_0 : \tau_0, r_1 : \tau_1\} \;\mid\; \Gamma_0 \\
\Gamma' &= \mathsf{A}[\sigma - v : \tau_0 + v' : \tau_1] \;\mid\; \{r_0 + v : \tau_0, r_1 - v' : \tau_1\} \;\mid\; \Gamma_0
\end{aligned}$$

The global net worth in Γ' can be computed in terms of the global net worth in Γ, by removing the value of the $v : \tau_0$ paid by A to the AMM, adding the value of the $v_1 : \tau_1$

obtained by A through the swap, and then adding the difference between the value of the minted tokens in Γ' and in Γ, i.e.:

$$S_{\Gamma'}\{\tau_0,\tau_1\}P_{\Gamma'}\{\tau_0,\tau_1\} - S_{\Gamma}\{\tau_0,\tau_1\}P_{\Gamma}\{\tau_0,\tau_1\}$$

By Lemma 4.3, we have that $S_{\Gamma'}\{\tau_0,\tau_1\} = S_{\Gamma}\{\tau_0,\tau_1\}$. Therefore:

$$\begin{aligned}
W(\Gamma') &= W(\Gamma) - vP\tau_0 + v'P\tau_1 + S_{\Gamma}\{\tau_0,\tau_1\}\big(P_{\Gamma'}\{\tau_0,\tau_1\} - P_{\Gamma}\{\tau_0,\tau_1\}\big) \\
&= W(\Gamma) - vP\tau_0 + v'P\tau_1 \\
&\quad + S_{\Gamma}\{\tau_0,\tau_1\} \cdot \Big(\frac{r_0P\tau_0 + r_1P\tau_1 + vP\tau_0 - v'P\tau_1}{S_{\Gamma}\{\tau_0,\tau_1\}} - \frac{r_0P\tau_0 + r_1P\tau_1}{S_{\Gamma}\{\tau_0,\tau_1\}}\Big) \\
&= W(\Gamma)
\end{aligned}$$

□

Proof of Lemma 4.6. Direct consequence of Lemma 4.5(a) and of the hypothesis that A does not hold minted tokens in Γ'. □

Proof of Lemma 4.8. Let $\Gamma^0 = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta^0$ be a reachable state. We define below a procedure to construct a sequence of transitions:

$$\Gamma^0 \xrightarrow{\mathsf{T}_1} \cdots \xrightarrow{\mathsf{T}_n} \Gamma^n \qquad \text{where} \;\; \Gamma^n = \{r_0^i : \tau_0, r_1^i : \tau_1\} \mid \Delta^n$$

By Lemma 4.2, we have that $r_0^i > 0$, $r_1^i > 0$, and $S_{\Gamma^i}\{\tau_0,\tau_1\} > 0$ for all i. At step i:

(1) Let $x = r_0^i - r_0'$ be the amount of τ_0 that users must redeem from the AMM, and let:

$$v = \frac{x}{r_0^i} S_{\Gamma^i}\{\tau_0,\tau_1\}$$

(2) if there exists some $\mathsf{A}[\sigma] \in \Gamma^i$ such that $\sigma(\{\tau_0,\tau_1\}) \geq v$, then A can fire $\mathsf{A} : \mathsf{rdm}(v : \{\tau_0,\tau_1\})$, obtaining, for some $r_1' \leq r_1$:

$$\begin{aligned}
\{r_0^i : \tau_0, r_1^i : \tau_1\} \mid \Delta^i \to \Gamma' &= \Big\{r_0^i - v\frac{r_0^i}{S_{\Gamma^i}\{\tau_0,\tau_1\}}, r_1' : \tau_1\Big\} \mid \cdots \\
&= \{r_0' : \tau_0, r_1' : \tau_1\} \mid \cdots
\end{aligned}$$

(3) otherwise, pick an $\mathsf{A}[\sigma] \in \Gamma^i$ such that $\sigma(\{\tau_0,\tau_1\}) = v' \geq 0$, fire $\mathsf{A} : \mathsf{rdm}(v' : \{\tau_0,\tau_1\})$.

Note that the procedure always terminates: since $S_{\Gamma^i}\{\tau_0,\tau_1\} > 0$ for all i, either step (2) or (3) can be performed; further, the number of performed transactions is bounded by the number of users, which is finite. □

Proof of Lemma 4.9. Assume that $\Gamma \xrightarrow{\mathsf{T}_0} \Gamma_0 \xrightarrow{\mathsf{T}_1} \Gamma_{01}$. We have the following exhaustive cases on the type of the transactions T_0 and T_1:

(1) $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{dep}(v_0 : \tau_0, v_0' : \tau_0')$.

(a) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{dep}(v_1 : \tau_1, v_1' : \tau_1)$. Both transactions are dep, so we are in case (a) of the statement. If $\{\tau_0, \tau_0'\} \neq \{\tau_1, \tau_1'\}$, then the thesis is straightforward, since $\mathsf{T}_0, \mathsf{T}_1$ operate on different AMMs. Otherwise, let:

$$a_0 = 1 + \tfrac{v_0}{r_0} \quad m_0 = \tfrac{v_0}{r_0} S_\Gamma\{\tau_0, \tau_1\} \quad a_{01} = 1 + \tfrac{v_1}{a_0 r_0} \quad m_{01} = \tfrac{v_1}{a_0 r_0} S_{\Gamma_0}\{\tau_0, \tau_1\}$$

$$a_1 = 1 + \tfrac{v_1}{r_0} \quad m_1 = \tfrac{v_1}{r_0} S_\Gamma\{\tau_0, \tau_1\} \quad a_{10} = 1 + \tfrac{v_0}{a_1 r_0} \quad m_{10} = \tfrac{v_0}{a_1 r_0} S_{\Gamma_1}\{\tau_0, \tau_1\}$$

We have that:

$$\begin{aligned}
& \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \\
\xrightarrow{\mathsf{T}_0}\ & \mathsf{A}[\sigma - v_0 : \tau_0 - v_0' : \tau_1 + m_0 : \{\tau_0, \tau_1\}] \mid \{a_0 r_0 : \tau_0, a_0 r_1 : \tau_1\} \mid \Delta \\
\xrightarrow{\mathsf{T}_1}\ & \mathsf{A}[\sigma - (v_0 + v_1) : \tau_0 - (v_0' + v_1') : \tau_1 + (m_0 + m_{01}) : \{\tau_0, \tau_1\}] \mid \\
& \{a_{01} a_0 r_0 : \tau_0, a_{01} a_0 r_1 : \tau_1\} \mid \Delta
\end{aligned}$$

Inverting the two transactions, we obtain:

$$\begin{aligned}
& \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \\
\xrightarrow{\mathsf{T}_1}\ & \mathsf{A}[\sigma - v_1 : \tau_0 - v_1' : \tau_1 + m_1 : \{\tau_0, \tau_1\}] \mid \{a_1 r_0 : \tau_0, a_1 r_1 : \tau_1\} \mid \Delta \\
\xrightarrow{\mathsf{T}_0}\ & \mathsf{A}[\sigma_1 - (v_0 + v_1) : \tau_0 - (v_0' + v_1') : \tau_1 + (m_1 + m_{10}) : \{\tau_0, \tau_1\}] \mid \\
& \{a_{10} a_1 r_0 : \tau_0, a_{10} a_1 r_1 : \tau_1\} \mid \Delta
\end{aligned}$$

We have that $a_{01} a_0 = a_{10} a_1$, since:

$$a_{10} a_1 = \left(1 + \tfrac{v_0}{a_1 r_0}\right) a_1 = \frac{a_1 r_0 + v_0}{r_0} = \frac{(1 + \frac{v_1}{r_0}) r_0 + v_0}{r_0} = \frac{r_0 + v_0 + v_1}{r_0}$$

$$a_{01} a_0 = \left(1 + \tfrac{v_1}{a_0 r_0}\right) a_0 = \frac{a_0 r_0 + v_1}{r_0} = \frac{(1 + \frac{v_0}{r_0}) r_0 + v_1}{r_0} = \frac{r_0 + v_0 + v_1}{r_0}$$

Furthermore, we have that $m_0 + m_{01} = m_1 + m_{10}$, since:

$$\begin{aligned}
m_{10} + m_1 &= \frac{v_0 v_1 + a_1 r_0 v_1 + r_0 v_0}{a_1 r_0^2} S_\Gamma\{\tau_0, \tau_1\} \\
&= \frac{v_0 v_1 + v_1 (r_0 + v_1) + r_0 v_0}{(r_0 + v_1) r_0} S_\Gamma\{\tau_0, \tau_1\} \\
&= \frac{(v_0 + v_1)(r_0 + v_1)}{(r_0 + v_1) r_0} S_\Gamma\{\tau_0, \tau_1\} \;=\; \frac{v_0 + v_1}{r_0} S_\Gamma\{\tau_0, \tau_1\} \\
m_{01} + m_0 &= \frac{v_0 v_1 + a_0 r_0 v_0 + r_0 v_1}{a_0 r_0^2} S_\Gamma\{\tau_0, \tau_1\} \\
&= \frac{v_0 v_1 + v_0 (r_0 + v_0) + r_0 v_1}{(r_0 + v_0) r_0} S_\Gamma\{\tau_0, \tau_1\} \\
&= \frac{(v_0 + v_1)(r_0 + v_0)}{(r_0 + v_0) r_0} S_\Gamma\{\tau_0, \tau_1\} \;=\; \frac{v_0 + v_1}{r_0} S_\Gamma\{\tau_0, \tau_1\}
\end{aligned}$$

Summing up, we have shown that $\Gamma_{01} = \Gamma_{10}$.

(b) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swap}(v_1, \tau_1, \tau'_1)$. Then, we are in case (a) of the statement, with $\mathit{tok}(\mathsf{T}_0)$ disjoint from $\mathit{tok}(\mathsf{T}_1)$. The thesis is straightforward by analysis of the rules.

(c) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{rdm}(v_1 : \{\tau_1, \tau'_1\})$. There are two subcases. If we are in case (a), then $\mathsf{T}_0, \mathsf{T}_1$ operate on different AMMs, and so the thesis is straightforward. Otherwise, if we are in case (b) of the statement, by hypothesis we know that $\mathsf{T}_1\mathsf{T}_0$ is enabled in Γ, leading to a state Γ_{10}. If $\{\tau_0, \tau'_0\} \neq \{\tau_1, \tau'_1\}$, then the thesis is straightforward. Otherwise, the proof is done by computing the states Γ_{01} and Γ_{10} and showing they are equal, similarly to what we have done in case (1a).

(2) $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{rdm}(v_0 : \{\tau_0, \tau'_0\})$.

(a) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{dep}(v_1 : \tau_1, v'_1 : \tau'_1)$. Symmetric to case (1c).

(b) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swap}(v_1, \tau_1, \tau'_1)$. Then, we are in case (a) of the statement, where $\{\tau_1, \tau'_1\}$ and $\{\tau_0, \tau'_0\}$ are disjoint. Then, the thesis is straightforward.

(c) $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{rdm}(v_1 : \{\tau_1, \tau'_1\})$. Then, we are in case (a) of the statement. If $\mathit{tok}(\mathsf{T}_0)$ is disjoint from $\mathit{tok}(\mathsf{T}_1)$, then the thesis is straightforward. Otherwise, note that the tokens paid by the AMM in response of T_0 and T_1 only depend on the ratio between the amounts of τ_0 and τ'_0 initially held by the AMM, which are constrained to preserve the ratio.

(3) $\mathsf{T}_0 = \mathsf{A}_0 : \mathsf{swap}(v_0, \tau_0, \tau'_0)$. The only case not covered by the previous items is when $\mathsf{T}_1 = \mathsf{A}_1 : \mathsf{swap}(v_1, \tau_1, \tau'_1)$. Then, we are in case (a) of the statement, where $\{\tau_1, \tau'_1\}$ and $\{\tau_0, \tau'_0\}$ are disjoint. The thesis is straightforward.

□

Proof of Theorem 4.10. For item 1, there are two cases, depending on whether T_0 is fired through rule [Dep0] or [Dep]. If T_0 is fired through rule [Dep], let $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$. We have that:

$$\Gamma_0 = \{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\} \mid \Delta_0 \qquad r_1 v_0 = r_0 v_1 \tag{B.3}$$
$$\Gamma_1 = \{(r_0 + v_0) + v'_0 : \tau_0, (r_1 + v_1) + v'_1 : \tau_1\} \mid \Delta_1 \qquad (r_1 + v_1)v'_0 = (r_0 + v_0)v'_1 \tag{B.4}$$

We must just check that the premises for firing $\mathsf{A} : \mathsf{dep}(v_0 + v'_0 : \tau_0, v'_1 + v'_1 : \tau_1)$ are satisfied:

$$\begin{aligned}
r_1(v_0 + v'_0) &= r_1 v_0 + r_1 v'_0 \\
&= r_0 v_1 + r_1 v'_0 && \text{by (B.3)} \\
&= r_0 v_1 + r_1 \Big(\frac{r_0 + v_0}{r_1 + v_1}\Big) v'_1 && \text{by (B.4)} \\
&= r_0 v_1 + r_1 \frac{r_0}{r_1} v'_1 && \text{by (B.1)} \\
&= r_0(v_1 + v'_1)
\end{aligned}$$

The case where T_0 is fired through rule [Dep0] is similar:

$$\begin{aligned}
&\Gamma_0 = \{v_0 : \tau_0, v_1 : \tau_1\} \mid \Delta_0 \\
&\Gamma_1 = \{v_0 + v'_0 : \tau_0, v_1 + v'_1 : \tau_1\} \mid \Delta_1 \qquad v_1 v'_0 = v_0 v'_1
\end{aligned}$$

The premises of [Dep0] when firing $\mathsf{A} : \mathsf{dep}(v_0 + v'_0 : \tau_0, v'_1 + v'_1 : \tau_1)$ are trivially satisfied, hence the thesis follows.

For item 2, let $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$ let $\tau = \{\tau_0, \tau_1\}$, and let $s = S_\Gamma \tau$. By rule [Rdm], we have that:

$$\Gamma_0 = \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \mid \Delta_0 \qquad v_i = v \cdot \frac{r_i}{s} \tag{B.5}$$

$$\Gamma_1 = \{(r_0 - v_0) - v'_0 : \tau_0, (r_1 - v_1) - v'_1 : \tau_1\} \mid \Delta_1 \qquad v'_i = v' \cdot \frac{r_i - v_i}{s - v} \tag{B.6}$$

Therefore, for $i \in \{0, 1\}$, we have that:

$$\begin{aligned}
r_i - v_i - v'_i &= r_i - v \cdot \frac{r_i}{s} - v' \cdot \frac{r_i - v \cdot \frac{r_i}{s}}{s - v} && \text{by (B.5), (B.6)}\\
&= r_i - v \cdot \frac{r_i(s-v)}{s(s-v)} - v' \cdot \frac{sr_i - v \cdot r_i}{s(s-v)}\\
&= r_i - \frac{vr_i(s-v) + v'(sr_i - vr_i)}{s(s-v)}\\
&= r_i - \frac{vr_i(s-v) + v'r_i(s-v)}{s(s-v)}\\
&= r_i - (v + v') \cdot \frac{r_i}{s}
\end{aligned}$$

from which the thesis follows. □

Proof of Theorem 4.11. By cases on the rule used to deduce $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$. The premise that $S_\Gamma \tau = 0$ implies $S_{\Gamma'} \tau = 0$ excludes the case [Dep0], so we have two cases:

- [Dep]. We have that $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$, and:

  $$\begin{aligned}
  \Gamma' &= \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : \{\tau_0, \tau_1\}] \mid \{r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1\} \mid \Delta\\
  &= \mathsf{A}[\sigma'] \mid \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta
  \end{aligned}$$

  where $v = \frac{v_i}{r_i} \cdot s$, with $s = S_\Gamma\{\tau_0, \tau_1\}$. Let $\mathsf{T}^{-1} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$. We have that:

  $$\Gamma' \xrightarrow{\mathsf{T}^{-1}} \mathsf{A}[\sigma' + v'_0 : \tau_0 + v'_1 : \tau_1 - v : \{\tau_0, \tau_1\}] \mid \{r'_0 - v'_0 : \tau_0, r'_1 - v'_1 : \tau_1\} \mid \Delta \;=\; \Gamma''$$

  where, for $i \in \{0, 1\}$ and $s' = S_\Gamma\{\tau_0, \tau_1\} = s + v$:

  $$v'_i = v \cdot \frac{r'_i}{s'} = v \cdot \frac{r_i + v_i}{s + v} = \left(\frac{v_i}{r_i} \cdot s\right) \cdot \frac{r_i + v_i}{s + \left(\frac{v_i}{r_i} \cdot s\right)} = \frac{v_i s(r_i + v_i)}{r_i s + v_i s} = v_i$$

  Since $v_i = v'_i$ for $i \in \{0, 1\}$, we conclude that $\Gamma'' = \Gamma$.
- [Rdm]. We have that $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$, $\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$, and:

  $$\begin{aligned}
  \Gamma' &= \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : \{\tau_0, \tau_1\}] \mid \{r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1\} \mid \Delta\\
  &= \mathsf{A}[\sigma'] \mid \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta
  \end{aligned}$$

  where $v_i = v \cdot \frac{r_i}{s}$, for $i \in \{0, 1\}$ and $s = S_\Gamma\{\tau_0, \tau_1\}$. Let $\mathsf{T}^{-1} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$. We have that:

  $$\Gamma' \xrightarrow{\mathsf{T}^{-1}} \mathsf{A}[\sigma' - v_0 : \tau_0 - v_1 : \tau_1 + v' : \{\tau_0, \tau_1\}] \mid \{r'_0 + v_0 : \tau_0, r'_1 + v_1 : \tau_1\} \mid \Delta \;=\; \Gamma''$$

where $v' = \frac{v_i}{r'_i} \cdot s'$, with $s' = S_{\Gamma'}\{\tau_0, \tau_1\} = s - v$. We have that:

$$v' = \frac{v_i}{r'_i} \cdot s' = \frac{v \cdot \frac{r_i}{s}}{r_i - v \cdot \frac{r_i}{s}} \cdot (s - v) = \frac{v \cdot r_i}{sr_i - vr_i} \cdot (s - v) = \frac{v}{s - v} \cdot (s - v) = v$$

Since $v' = v$, we conclude that $\Gamma'' = \Gamma$.

□

Appendix C. Proofs for Section 5

Proof of Lemma 5.2. The condition $S_\Gamma\{\tau_0, \tau_1\} > 0$ ensures that Γ contains an AMM for the pair τ_0, τ_1. The premise $\sigma(\tau_0) \geq x$ ensures that A has enough units of the input token τ_0. Output-boundedness implies the premise $x \cdot SX(x, r_0, r_1) < r_1$ of [Swap]. □

Proof of Lemma 5.4. Straightforward by Definition 5.3 and Lemma 3.2. □

Proof of Theorem 5.6. Let $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$. We have that:

$$\Gamma_0 = \{r_0 + x_0 : \tau_0, r_1 - y_0 : \tau_1\} \mid \Delta_0 \qquad y_0 = x_0 \cdot SX(x_0, r_0, r_1)$$
$$\Gamma_1 = \{r_0 + x_0 + x_1 : \tau_0, r_1 - y_0 - y_1 : \tau_1\} \mid \Delta_1 \qquad y_1 = x_1 \cdot SX(x_1, r_0 + x_0, r_1 - y_0)$$

Since SX is additive, we have that:

$$SX(x_0 + x_1, r_0, r_1) = \frac{y_0 + y_1}{x_0 + x_1}$$

Therefore, rule [Swap] gives the thesis:

$$\Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}(x_0+x_1,\tau_0,\tau_1)} \{r_0 + x_0 + x_1 : \tau_0, r_1 - (y_0 + y_1) : \tau_1\} \mid \Delta_1$$

□

Proof of Lemma 5.7. Since SX is output-bounded, then by Lemma 5.2, $\mathsf{T}(x_0)$ and $\mathsf{T}(x_0 + x_1)$ are enabled in Γ, and $\mathsf{T}(x_1)$ is enabled in Γ'. Let:

$$\alpha = SX(x_0, r_0, r_1) \qquad \beta = SX(x_1, r_0 + x_0, r_1 - \alpha x_0)$$

By additivity of SX (Definition 5.5), we have that:

$$\gamma = SX(x_0 + x_1, r_0, r_1) = \frac{\alpha x_0 + \beta x_1}{x_0 + x_1} \tag{C.1}$$

Therefore:

$$\begin{aligned}
& G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0 + x_1)) - G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) \\
&= \gamma(x_0 + x_1)P\tau_1 - (x_0 + x_1)P\tau_0 - \alpha x_0 P\tau_1 + x_0 P\tau_0 && \text{(Lemma 3.2)} \\
&= \big((\gamma(x_0 + x_1) - \alpha x_0\big)P\tau_1 - x_1 P\tau_0 \\
&= \big(\alpha x_0 + \beta x_1 - \alpha x_0)\big)P\tau_1 - x_1 P\tau_0 && \text{(Equation C.1)} \\
&= \beta x_1 P\tau_1 - x_1 P\tau_0 \\
&= G_{\mathsf{A}}(\Gamma', \mathsf{T}(x_1)) && \text{(Lemma 3.2)}
\end{aligned}$$

□

Proof of Theorem 5.9. Let $\Gamma = \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta$, and let $y = x \cdot SX(x, r_0, r_1)$. By the [Swap] rule, there exists Δ' such that:

$$\Gamma' \;=\; \{r_0 + x : \tau_0, r_1 - y : \tau_1\} \mid \Delta'$$

Let $\mathsf{T}^{-1} = \mathsf{A} : \mathsf{swap}(y, \tau_1, \tau_0)$, and let $x' = y \cdot SX(y, r_1 - y, r_0 + x)$. For some Δ'', we have:

$$\Gamma' \xrightarrow{\mathsf{T}^{-1}} \{r_0 + x - x' : \tau_0, r_1 - y + y : \tau_1\} \mid \Delta''$$

By reversibility of the swap rate, we have that:

$$\frac{y}{x} = SX(x, r_0, r_1) \implies SX(y, r_1 - y, r_0 + x) = \frac{x}{y}$$

from which we obtain that:

$$x' \;=\; y \cdot SX(y, r_1 - y, r_0 + x) \;=\; y \cdot \frac{x}{y} \;=\; x$$

from which we obtain the thesis. □

Proof of Lemma 5.10. Straightforward from the definition of gain and from Theorem 5.9. □

Proof of Lemma 5.12. Let $\{r_0 : \tau_0, r_1 : \tau_1\} \in \Gamma$, $\{r_0' : \tau_0, r_1' : \tau_1\} \in \Gamma'$, and let $a = r_0'/r_0$. We have that:

$$\begin{aligned}
X_\Gamma(\tau_0, \tau_1) &= \lim_{x \to 0} SX(x, r_0, r_1) && \text{by Equation (3.3)}\\
&= \lim_{x \to 0} SX(ax, ar_0, ar_1) && \text{since } SX \text{ is homogeneous}\\
&= \lim_{x \to 0} SX(ax, r_0', r_1') && \text{by Lemma 4.4(a)}\\
&= X_{\Gamma'}(\tau_0, \tau_1) && \text{by Equation (3.3)}
\end{aligned}$$

□

Proof of Lemma 5.13. For item (a), let $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$. By rule [Dep], $r_i' = r_i + v_i$ for $i \in \{0, 1\}$, with $r_0 v_1 = r_1 v_0$. By Lemma 4.4(a), $r_0+v_0/r_1+v_1 = r_0/r_1$. Then:

$$r_0 + v_0 = \frac{r_1 + v_1}{r_1} r_0 = a\, r_0 \qquad r_1 + v_1 = \frac{r_1 + v_1}{r_1} r_1 = a\, r_1 \qquad \text{where } a = \tfrac{r_1 + v_1}{r_1}$$

Therefore:

$$\begin{aligned}
SX(x, r_0', r_1') &= SX(x, ar_0, ar_1)\\
&= SX(\tfrac{x}{a}, r_0, r_1) && \text{(homogeneity)}\\
&> SX(x, r_0, r_1) && \text{(strict monotonicity, } a > 1 \implies \tfrac{x}{a} < x)
\end{aligned}$$

The thesis $\Delta X_\Gamma(x, \tau_0, \tau_1) > \Delta X_{\Gamma'}(x, \tau_0, \tau_1)$ follows from this inequality and Lemma 5.12.
For item (b), let $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : \{\tau_0, \tau_1\})$. By rule [Rdm], for $i \in \{0, 1\}$:

$$r_i' \;=\; r_i - v_i \;=\; r_i - v \frac{r_i}{S_\Gamma\{\tau_0, \tau_1\}} \;=\; a\, r_i \qquad \text{where } a = 1 - \tfrac{v}{S_\Gamma\{\tau_0, \tau_1\}}$$

Therefore:

$$\begin{aligned} SX(x, r_0', r_1') &= SX(x, ar_0, ar_1) \\ &= SX(\tfrac{x}{a}, r_0, r_1) && \text{(homogeneity)} \\ &< SX(x, r_0, r_1) && \text{(strict monotonicity, } a < 1 \implies \tfrac{x}{a} < x) \end{aligned}$$

The thesis $\Delta X_{\Gamma}(x, \tau_0, \tau_1) < \Delta X_{\Gamma'}(x, \tau_0, \tau_1)$ follows from this inequality and Lemma 5.12. □

Proof of Theorem 5.15. For output-boundedness, let $x > 0$ and $r_0, r_1 > 0$. We have that:

$$SX(x, r_0, r_1) = \frac{r_1}{r_0 + x} < \frac{r_1}{x}$$

For monotonicity, Let $x' \leq x$, $r_0' \leq r_0$ and $r_1 \leq r_1'$. We have that:

$$SX(x', r_0', r_1') = \frac{r_1'}{r_0' + x'} \geq \frac{r_1}{r_0 + x} = SX(x, r_0, r_1)$$

The proof for strict monotonicity is similar.
For additivity, by Definition 2.1 we have that:

$$\begin{aligned} \alpha &= SX(x, r_0, r_1) = \frac{r_1}{r_0 + x} \\ \beta &= SX(y, r_0 + x, r_1 - \alpha x) = \frac{r_1 - \alpha x}{r_0 + x + y} = \frac{r_0 r_1}{(r_0 + x)(r_0 + x + y)} \end{aligned}$$

Therefore:

$$\begin{aligned} \frac{\alpha x + \beta y}{x + y} &= \frac{1}{x + y}\left(\frac{r_1 x}{r_0 + x} + \frac{r_0 r_1 y}{(r_0 + x)(r_0 + x + y)}\right) \\ &= \frac{1}{x + y} \frac{r_0 r_1 x + r_1 x^2 + r_1 x y + r_0 r_1 y}{(r_0 + x)(r_0 + x + y)} \\ &= \frac{r_1 (r_0 + x)(x + y)}{(x + y)(r_0 + x)(r_0 + x + y)} \\ &= \frac{r_1}{r_0 + x + y} \\ &= SX(x + y, r_0, r_1) \end{aligned}$$

For reversibility, let $\alpha = SX(x, r_0, r_1)$. By Definition 2.1, we have that:

$$SX(\alpha x, r_1 - \alpha x, r_0 + x) = \frac{r_0 + x}{(r_1 - \alpha x) + \alpha x} = \frac{r_0 + x}{r_1} = \left(\frac{r_1}{r_0 + x}\right)^{-1} = \frac{1}{\alpha}$$

For homogeneity, we have that:

$$SX(ax, ar_0, ar_1) = \frac{ar_1}{ar_0 + ax} = \frac{r_1}{r_0 + x} = SX(x, r_0, r_1)$$

The computations of the internal exchange rate and of the slippage are straightforward. □

Proof of Theorem 5.16. Output-boundedness, monotonicity and homogeneity are straightforward. For additivity, by Definition 5.16 we have that:

$$\alpha = SX(x, r_0, r_1) = \frac{r_1}{x}\left(1 - \left(\frac{r_0}{r_0 + x}\right)^{\frac{w_0}{w_1}}\right)$$
$$\beta = SX(y, r_0 + x, r_1 - \alpha x) = \frac{r_1 - \alpha x}{y}\left(1 - \left(\frac{r_0 + x}{r_0 + x + y}\right)^{\frac{w_0}{w_1}}\right)$$

Therefore:

$$\begin{aligned}
\frac{\alpha x + \beta y}{x + y} &= \frac{1}{x + y}\left(\alpha x + (r_1 - \alpha x)\left(1 - \left(\frac{r_0 + x}{r_0 + x + y}\right)^{\frac{w_0}{w_1}}\right)\right) \\
&= \frac{1}{x + y}\left(r_1 - r_1\left(\frac{r_0 + x}{r_0 + x + y}\right)^{\frac{w_0}{w_1}} + r_1\left(1 - \left(\frac{r_0}{r_0 + x}\right)^{\frac{w_0}{w_1}}\right)\left(\frac{r_0 + x}{r_0 + x + y}\right)^{\frac{w_0}{w_1}}\right) \\
&= \frac{1}{x + y}\left(r_1 - r_1\left(\frac{r_0}{r_0 + x}\right)^{\frac{w_0}{w_1}}\left(\frac{r_0 + x}{r_0 + x + y}\right)^{\frac{w_0}{w_1}}\right) \\
&= \frac{r_1}{x + y}\left(1 - \left(\frac{r_0}{r_0 + x + y}\right)^{\frac{w_0}{w_1}}\right) \\
&= SX(x + y, r_0, r_1)
\end{aligned}$$

□

Appendix D. Proofs for Section 6

Proof of Lemma 6.1. Assume that $SX(x, r_0, r_1) \geq X(\tau_0, \tau_1)$. Let $\alpha(z) = SX(z, r_1, r_0)$. We have that:

$$\begin{aligned}
SX(y, r_1, r_0) &< \lim_{z \to 0} SX(z, r_1, r_0) && \text{(strict monotonicity)} \\
&= \lim_{z \to 0} \frac{1}{SX(\alpha(z) \cdot z, r_0 - \alpha(z) \cdot z, r_1 + z)} && \text{(reversibility)} \\
&< \frac{1}{SX(x, r_0, r_1)} && \text{(strict monotonicity)} \\
&\leq \frac{1}{X(\tau_0, \tau_1)} && \text{(hypothesis)} \\
&= X(\tau_1, \tau_0) && \text{(def. of } X\text{)}
\end{aligned}$$

where in the second application of strict monotonicity, we have exploited the (asymptotic) inequalities $\alpha(z) \cdot z < x$ (where $\lim_{z \to 0} \alpha(z) \cdot z = 0$ follows from the existence of the internal exchange rate), $r_0 - \alpha(z) \cdot z < r_0$, and $r_1 + z > r_1$. □

Proof of Lemma 6.2. Let $y > 0$. Assume that $G_{\mathsf{A}}(\Gamma, \mathsf{T}_d(x)) > 0$. Then, $\mathsf{T}_d(x)$ is enabled in Γ, and so by Lemma 3.3, we have that $SX(x, r_d, r_{1-d}) > X(\tau_d, \tau_{1-d})$. Then, by Lemma 6.1 it follows that $SX(y, r_{1-d}, r_d) < X(\tau_{1-d}, \tau_d)$. Since $\sigma\tau_{1-d} \geq y$ and SX is output-bounded, then Lemma 5.2 implies that $\mathsf{T}_{1-d}(y)$ is enabled in Γ. By using again Lemma 3.3, concluding that $G_{\mathsf{A}}(\Gamma, \mathsf{T}_{1-d}(y)) < 0$. □

Proof of Theorem 6.3. Let x_0 and Γ' be as in the hypotheses, i.e.:

$$\Gamma \xrightarrow{\mathsf{T}(x_0)} \Gamma' = \mathsf{A}[\sigma'] \mid \{r_0 + x_0 : \tau_0, r_1 - \alpha x_0 : \tau_1\} \mid \Delta \qquad \text{where} \quad \begin{array}{l} \alpha = SX(x_0, r_0, r_1) \\ X_{\Gamma'}(\tau_0, \tau_1) = X(\tau_0, \tau_1) \end{array}$$

We have two cases, depending on whether $x > x_0$ or $x < x_0$.

- If $x > x_0$, let $x_1 > 0$ be such that $x = x_0 + x_1$. Since SX is output-bounded and additive, then by Lemma 5.7:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}(x)) \;=\; G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) + G_{\mathsf{A}}(\Gamma', \mathsf{T}(x_1)) \tag{D.1}$$

  We have that:

$$\begin{aligned} SX(x_1, r_0 + x_0, r_1 - \alpha x_0) &< \lim_{z \to 0} SX(z, r_0 + x_0, r_1 - \alpha x_0) && \text{(strict monotonicity)} \\ &= X_{\Gamma'}(\tau_0, \tau_1) && \text{def. } X_{\Gamma'} \\ &= X(\tau_0, \tau_1) && \text{(hypothesis)} \end{aligned}$$

  Then, by Lemma 3.3 we obtain $G_{\mathsf{A}}(\Gamma', \mathsf{T}(x_1)) < 0$. By Equation (D.1), we conclude that $G_{\mathsf{A}}(\Gamma, \mathsf{T}(x)) < G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0))$.

- If $x < x_0$, let $x_1 > 0$ be such that $x_0 = x + x_1$. Since SX is output-bounded, then by Lemma 5.2, $\mathsf{T}(x_0)$ and $\mathsf{T}(x)$ are enabled in Γ, and $\mathsf{T}(x_1)$ is enabled in the state Γ_1 reached after performing $\mathsf{T}(x_1)$, i.e.:

$$\Gamma \xrightarrow{\mathsf{T}(x)} \Gamma_1 \xrightarrow{\mathsf{T}(x_1)} \Gamma'$$

  Since SX is output-bounded and additive, then by Lemma 5.7:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) \;=\; G_{\mathsf{A}}(\Gamma, \mathsf{T}(x)) + G_{\mathsf{A}}(\Gamma_1, \mathsf{T}(x_1))$$

  Since SX is reversible, then by Theorem 5.9, $\mathsf{T}(x_1)$ has an inverse, which has the form $\mathsf{T}^{-1}(x_1) = \mathsf{A} : \mathsf{swap}(y_1, \tau_1, \tau_0)$ for some $y_1 > 0$. Then, by Lemma 5.10, $G_{\mathsf{A}}(\Gamma_1, \mathsf{T}(x_1)) = -G_{\mathsf{A}}(\Gamma', \mathsf{T}^{-1}(y_1))$, therefore:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0)) \;=\; G_{\mathsf{A}}(\Gamma, \mathsf{T}(x)) - G_{\mathsf{A}}(\Gamma', \mathsf{T}^{-1}(y_1)) \tag{D.2}$$

  We have that:

$$\begin{aligned} SX(y_1, r_1 - \alpha x_0, r_0 + x_0) &< \lim_{z \to 0} SX(z, r_1 - \alpha x_0, r_0 + x_0) && \text{(strict monotonicity)} \\ &= X_{\Gamma'}(\tau_1, \tau_0) && \text{def. } X_{\Gamma'} \\ &= \frac{1}{X_{\Gamma'}(\tau_0, \tau_1)} && \text{(Equation (5.1))} \\ &= \frac{1}{X(\tau_0, \tau_1)} && \text{(hypothesis)} \\ &= X(\tau_1, \tau_0) && \text{(def. } X) \end{aligned}$$

  Then, by Lemma 3.3 we obtain $G_{\mathsf{A}}(\Gamma', \mathsf{T}^{-1}(y_1)) < 0$. By Equation (D.2), we conclude that $G_{\mathsf{A}}(\Gamma, \mathsf{T}(x)) < G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0))$.

For uniqueness, by contradiction assume that there exists $x_1 \neq x_0$ satisfying Equation (6.1). Then, it should be $G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_1)) > G_{\mathsf{A}}(\Gamma, \mathsf{T}(x_0))$ — contradiction. □

Proof of Lemma 6.4. Let $\Gamma \xrightarrow{\mathsf{T}} \Gamma' = \mathsf{A}[\sigma'] \mid \{r_0 + x_0 : \tau_0, r_1 - x_0 \cdot SX(x_0, r_0, r_1) : \tau_1\}$. We have that:

$$
\begin{aligned}
X_{\Gamma'}(\tau_0, \tau_1) &= \frac{r_1 - x_0 \cdot SX(x_0, r_0, r_1)}{r_0 + x_0} && \text{by Theorem 5.15} \\
&= \frac{r_1 - x_0 \cdot \frac{r_1}{r_0 + x_0}}{r_0 + x_0} && \text{by Definition 2.1} \\
&= \frac{r_0 r_1}{(r_0 + x_0)^2} \\
&= \frac{r_0 r_1}{\frac{P_{\tau_1}}{P_{\tau_0}} r_0 r_1} && \text{by Equation (6.2)} \\
&= X(\tau_0, \tau_1) && \text{by Equation (3.2)}
\end{aligned}
$$

The thesis follows from Theorem 6.3. □

Proof of Theorem 6.6. Let:

$$\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \xrightarrow{\mathsf{T}_{\mathsf{dep}}} \Gamma' = \mathsf{A}[\sigma'] \mid \{r'_0 : \tau_0, r'_1 : \tau_1\} \mid \Delta'$$

The hypothesis $wal(\mathsf{T}_{\mathsf{swap}}) = \mathsf{A} \neq wal(\mathsf{T}_{\mathsf{rdm}})$ means that the user who performs the deposit is *not* A, hence the deposit does not affect the number of minted tokens in A's wallet. Then:

$$
\begin{aligned}
& G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{dep}} \mathsf{T}_{\mathsf{swap}}) \\
&\quad = G_{\mathsf{A}}(\Gamma', \mathsf{T}_{\mathsf{swap}}) \\
&\quad = x \cdot \left(SX(x, r'_0, r'_1)\, P_{\tau_1} - P_{\tau_0}\right) \cdot \left(1 - \frac{\sigma'\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && \text{(Lemma 3.2)} \\
&\quad > x \cdot \left(SX(x, r_0, r_1)\, P_{\tau_1} - P_{\tau_0}\right) \cdot \left(1 - \frac{\sigma'\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && \text{(Lemma 5.13(a))} \\
&\quad = x \cdot \left(SX(x, r_0, r_1)\, P_{\tau_1} - P_{\tau_0}\right) \cdot \left(1 - \frac{\sigma\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && (\sigma'\{\tau_0, \tau_1\} = \sigma\{\tau_0, \tau_1\}) \\
&\quad > x \cdot \left(SX(x, r_0, r_1)\, P_{\tau_1} - P_{\tau_0}\right) \cdot \left(1 - \frac{\sigma\{\tau_0, \tau_1\}}{S_{\Gamma}\{\tau_0, \tau_1\}}\right) && (S_{\Gamma'}\{\tau_0, \tau_1\} > S_{\Gamma}\{\tau_0, \tau_1\}) \\
&\quad = G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{swap}})
\end{aligned}
$$

□

Proof of Theorem 6.8. Let Γ and Γ_d be as in the statement. By rule [Dep], $r'_i = r_i + v_i$ for $i \in \{0, 1\}$. By Lemma 4.4(a), we have that ${r_0+v_0}/{r_1+v_1} = {r_0}/{r_1}$. Then:

$$r'_0 = r_0 + v_0 = \frac{r_1 + v_1}{r_1} r_0 = a\, r_0 \qquad r'_1 = r_1 + v_1 = \frac{r_1 + v_1}{r_1} r_1 = a\, r_1 \qquad \text{where } a = \frac{r_1 + v_1}{r_1}$$

For item (1), assume that $\lambda = \mathsf{A} : \mathsf{swap}(x, \tau_0, \tau_1)$ is a solution to the arbitrage game in Γ. By Theorem 6.3, it must be:

$$X_{\Gamma_s}(\tau_0, \tau_1) = X(\tau_0, \tau_1) \qquad \text{where } \Gamma \xrightarrow{\lambda} \Gamma_s \tag{D.3}$$

Let $x' = ax$, let $\mathsf{T}' = \mathsf{A} : \mathsf{swap}(x', \tau_0, \tau_1)$, and let $\Gamma_d \xrightarrow{\mathsf{T}'} \Gamma_{ds}$. We have that:

$$
\begin{aligned}
& X_{\Gamma_{ds}}(\tau_0, \tau_1) \\
&\quad = \lim_{z \to 0} SX\left(z, r_0' + x', r_1' - x' \cdot SX\left(x', r_0', r_1'\right)\right) \\
&\quad = \lim_{z \to 0} SX\left(z, ar_0 + ax, ar_1 - ax \cdot SX\left(ax, ar_0, ar_1\right)\right) \\
&\quad = \lim_{z \to 0} SX\left(z, ar_0 + ax, ar_1 - ax \cdot SX\left(x, r_0, r_1\right)\right) && \text{(homogeneity)} \\
&\quad = \lim_{z \to 0} SX\left(z, r_0 + x, r_1 - x \cdot SX\left(x, r_0, r_1\right)\right) && \text{(homogeneity)} \\
&\quad = X_{\Gamma_s}(\tau_0, \tau_1) && \text{(def. } X_{\Gamma_s}) \\
&\quad = X(\tau_0, \tau_1) && \text{(Equation (D.3))}
\end{aligned}
$$

Therefore, Theorem 6.3 implies that T' is a solution to the arbitrage game in Γ_d. We compute the gain of T' in Γ_d as follows:

$$
\begin{aligned}
G_{\mathsf{A}}(\Gamma_d, \mathsf{T}') &= x' \cdot \left(SX\left(x', r_0', r_1'\right) P\tau_1 - P\tau_0\right) \\
&= ax \cdot \left(SX\left(ax, ar_0, ar_1\right) P\tau_1 - P\tau_0\right) \\
&= ax \cdot \left(SX\left(x, r_0, r_1\right) P\tau_1 - P\tau_0\right) && \text{(homogeneity)} \\
&= a\, G_{\mathsf{A}}(\Gamma, \mathsf{T})
\end{aligned}
$$

For item (2), assume that ε is a solution to the arbitrage game in Γ. By contradiction, assume that $\lambda_d = \mathsf{A} : \mathsf{swap}(x', \tau_0, \tau_1)$ is a solution in Γ_d. By Theorem 6.3, it must be:

$$X_{\Gamma_{ds}}(\tau_0, \tau_1) = X(\tau_0, \tau_1) \tag{D.4}$$

The chain of equations above shows that $X_{\Gamma_{ds}}(\tau_0, \tau_1) = X_{\Gamma_s}(\tau_0, \tau_1)$. By Equation (D.4), this implies that $X_{\Gamma_{ds}}(\tau_0, \tau_1) = X(\tau_0, \tau_1)$. Hence, by Theorem 6.3, ε cannot be a solution to the arbitrage game in Γ — contradiction. □

Proof of Theorem 6.9. Let:

$$\Gamma = \mathsf{A}[\sigma] \mid \{r_0 : \tau_0, r_1 : \tau_1\} \mid \Delta \xrightarrow{\mathsf{T}_{\mathsf{rdm}}} \Gamma' = \mathsf{A}[\sigma'] \mid \{r_0' : \tau_0, r_1' : \tau_1\} \mid \Delta'$$

The hypothesis $\mathit{wal}(\mathsf{T}_{\mathsf{swap}}) = \mathsf{A} \neq \mathit{wal}(\mathsf{T}_{\mathsf{rdm}})$ means that the user who performs the redeem is *not* A, hence the redeem does not affect the number of minted tokens in A's wallet. Then:

$$
\begin{aligned}
& G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{rdm}} \mathsf{T}_{\mathsf{swap}}) \\
&\quad = G_{\mathsf{A}}(\Gamma', \mathsf{T}_{\mathsf{swap}}) \\
&\quad = x \cdot \left(SX\left(x, r_0', r_1'\right) P\tau_1 - P\tau_0\right) \cdot \left(1 - \frac{\sigma'\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && \text{(Lemma 3.2)} \\
&\quad < x \cdot \left(SX\left(x, r_0, r_1\right) P\tau_1 - P\tau_0\right) \cdot \left(1 - \frac{\sigma'\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && \text{(Lemma 5.13(b))} \\
&\quad = x \cdot \left(SX\left(x, r_0, r_1\right) P\tau_1 - P\tau_0\right) \cdot \left(1 - \frac{\sigma\{\tau_0, \tau_1\}}{S_{\Gamma'}\{\tau_0, \tau_1\}}\right) && (\sigma'\{\tau_0, \tau_1\} = \sigma\{\tau_0, \tau_1\}) \\
&\quad < x \cdot \left(SX\left(x, r_0, r_1\right) P\tau_1 - P\tau_0\right) \cdot \left(1 - \frac{\sigma\{\tau_0, \tau_1\}}{S_{\Gamma}\{\tau_0, \tau_1\}}\right) && (S_{\Gamma'}\{\tau_0, \tau_1\} < S_{\Gamma}\{\tau_0, \tau_1\}) \\
&\quad = G_{\mathsf{A}}(\Gamma, \mathsf{T}_{\mathsf{swap}})
\end{aligned}
$$

$\square$

Proof of Theorem 6.10. Let Γ and Γ_d be as in the statement. By rule [RDM], it must be, for $i \in \{0,1\}$:

$$r'_i \;=\; r_i - v_i \;=\; r_i - vRX^i_\Gamma(\tau_0, \tau_1) \;=\; r_i - v\frac{r_i}{S_\Gamma\{\tau_0, \tau_1\}} \;=\; ar_i \qquad \text{where } a = 1 - \frac{v}{S_\Gamma\{\tau_0, \tau_1\}}$$

The rest of the proof follows exactly that of Theorem 6.8. $\square$

SoK: Lending Pools in Decentralized Finance

Publication Information

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. “SoK: Lending Pools in Decentralized Finance.” *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25.* Springer Berlin Heidelberg, 2021.

Contribution

- Co-author.

SoK: Lending Pools in Decentralized Finance

Massimo Bartoletti[1], James Hsin-yu Chiang[2], Alberto Lluch Lafuente[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
[2] Technical University of Denmark, DTU Compute, Copenhagen, Denmark

Abstract. Lending pools are decentralized applications which allow mutually untrusted users to lend and borrow crypto-assets. These applications feature complex, highly parametric incentive mechanisms to equilibrate the loan market. This complexity makes the behaviour of lending pools difficult to understand and to predict: indeed, ineffective incentives and attacks could potentially lead to emergent unwanted behaviours. Reasoning about lending pools is made even harder by the lack of executable models of their behaviour: to precisely understand how users interact with lending pools, eventually one has to inspect their implementations, where the incentive mechanisms are intertwined with low-level implementation details. Further, the variety of existing implementations makes it difficult to distill the common aspects of lending pools. We systematize the existing knowledge about lending pools, leveraging a new formal model of interactions with users, which reflects the archetypal features of mainstream implementations. This enables us to prove some general properties of lending pools, such as the correct handling of funds, and to precisely describe vulnerabilities and attacks. We also discuss the role of lending pools in the broader context of decentralized finance.

1 Introduction

The emergence of permissionless, public blockchains has given birth to an entire ecosystem of *crypto-tokens* representing digital assets. Facilitated and accelerated by smart contracts and standardized token interfaces [1], these so-called *decentralized finance* (DeFi) applications promise an open alternative to the traditional financial system. One of the main DeFi applications are *lending pools*, which incentivize users to lend some of their crypto-assets to borrowers. Unlike in traditional finance, all the parameters of a loan, like its interests, maturity periods or token prices, are determined by a smart contract, which also includes mechanisms to incentivize honest behaviour (e.g., loans are eventually repaid), economic growth and stability. Existing lending pool platforms are already handling large volumes of crypto-assets: as of writing, the two main platforms currently hold \$1.7B [17] and \$1.4B [5] worth of tokens in their smart contracts.

Lending pools are inherently hard to design. Besides the typical difficulty of implementing secure smart contracts [2–4, 35], lending pools feature complex economic incentive mechanisms, which make it difficult to understand when a lending pool actually achieves the economic goals it was designed for. As a matter of fact, a recent failure of the oracle price feed used by the Compound lending pool platform led to \$100M of collateral being (incorrectly) liquidated [19]. Indeed, most current literature in DeFi is devoted to study the economic impact of these incentive mechanisms [39, 40, 46–48, 50].

The problem is made even more complex by the absence of abstract operational descriptions of the behaviour of lending pools. Current descriptions are either high-level economic models [46, 47, 50], or the actual implementations. While, on the one hand, economic models are useful to understand the macroscopic financial aspects of lending pools, on the other hand they do not precisely describe the interactions between a lending pool and its users. Still, understanding these interactions is crucial to determine if a lending pool is vulnerable to attacks where some users deviate from the expected behaviour. Implementations, instead, reflect the exact actual behaviour, but at a level of detail that makes high-level understanding and reasoning unfeasible.

Contributions This paper presents a systematic analysis of the behaviour of lending pools, of their properties, vulnerabilities, and of the related literature. Based on a thorough inspection of the implementations of the two main lending pool platforms, Compound [16] and Aave [8], we synthesise a formal, operational model of the interactions between users and lending pools, encompassing their incentive mechanisms. More specifically, our contributions are:

1. a formal model of lending pools, which precisely describes their interactions as transitions of a state machine. Our model captures all the typical transactions of lending pools, and all the main economic features, like collateralization, exchange rates, token price, and interest accrual (Section 3);
2. the formalization and proof of fundamental behavioural properties of lending pools, which were informally stated in literature, and are expected to be satisfied by any implementation (Section 4);
3. the formalization of relevant properties of the incentive mechanisms of lending pools, and a discussion of their vulnerabilities and attacks (Section 5);
4. a thorough discussion on the interplay between lending pools and other DeFi archetypes, like stable coins and automatic market makers (Section 6).

Overall, our contributions help address the aforementioned challenges in the design of lending pools. Firstly, our formal model provides a precise understanding of the behaviour of lending pools, abstracting from low-level implementation details. Our model is faithful to mainstream lending pool implementations like Compound [16] and Aave [8]; still, for the sake of clarity, we have introduced high-level abstractions over low-level details: we discuss the differences between our model and the actual lending pool platforms in Section 7. Secondly, our formalisation of the properties of the incentive mechanisms of lending pools makes it easier to understand and analyse their vulnerabilities and attacks. In this regard, our model is directly amenable for its interpretation as an *executable specification*, thus paving the way for automated analysis techniques, which may include mechanised proofs of contract properties and agent-based simulations of lending pools and other DeFi contracts.

2 Background

Lending pools (in short, LPs) are financial applications which create a market of loans of crypto-assets, providing incentive mechanisms to equilibrate the market. We now overview the main features of LPs; a glossary of LP terms is in Table 1.

Users can lend assets to a LP by transferring *tokens* from their accounts to the LP. In return, they receive a *claim*, represented as tokens *minted* by the LP, which can later be redeemed for an equal or increased amount of tokens, of the same *token type* of the original deposit. Lending is incentivized by interest or fees: the depositor speculates that the claim will be redeemable for a value greater than that of the original deposit. Users can redeem claims by transferring minted tokens to the LP, which pays back the original tokens (with accrued interest) to the redeemer, simultaneously burning the minted tokens. However, redeeming claims is not always possible, as the LP could not have a sufficient balance of the original tokens, as these may have been lent to other users.

User initiate a *loan* by borrowing tokens deposited to a LP. To incentivise users to eventually repay the loan, borrowing requires to provide a *collateral*. Collaterals can be either tokens deposited to the LP when the loan is initiated, and locked for the whole loan duration, or they can be tokens held by the borrower but *seizable* by the LP when a user fails to repay a loan. An unpaid loan of A can be *liquidated* by B, who pays (part of) A's loan in return for a discounted amount of A's collateral. For this to be possible, the value of the collateral must be greater than that of the loan. To incentivize deposits, loans *accrue* interest, which increase a user's loan amount by the *interest rate*.

Token	A digital representation of some asset, transferable between users.
Token type	A set of tokens. Tokens of a given type are interchangeable (or *fungible*), whereas tokens of different token types are not.
Native token	The default token type of a blockchain (e.g., ETH for Ethereum).
Token price	The price of a token type τ is the amount of units of a given native cryptocurrency (or fiat currency) needed to buy one unit of τ.
Exchange rate	Given two token types τ and τ', the ratio τ/τ' at which a user can exchange units of token type τ' for units of τ in a blockchain interaction.
Lender	A user who transfers units of a token type in return for a *claim* on a full repayment in the future, which may include additional fees or interest.
Claim	A right to token units in the future. Claims are represented as tokens, which are *minted* and destroyed as claims are created and redeemed.
Minting	Creation of tokens performed by the LP upon deposits.
Borrower	A user who wishes to obtain a *loan* of token type τ. The borrower is required to hold *collateral* of another token τ' to secure the loan.
Collateral	A user balance of tokens which can be seized if the user does not adequately repay a loan.
Collateralization	The ratio of deposited *collateral* value over the borrower's total loan value.
Liquidation	When the *collateralization* of user A falls below a minimum threshold it is *undercollateralized*: here, a user B can repay a fraction of A's loan, in return for a discounted amount of A's collateral *seized* by B.
Interest rate	The rate of loan growth when accruing interest.

Table 1: Glossary of financial terms used in Lending Pools.

3 Lending pools

In this section we introduce a formal, operational model of lending pools. We do this incrementally, starting from a basic model of blockchains, on top of which we will specify the behaviour lending pools.

3.1 A basic model of blockchains

We assume a set of *users* $\mathbb{A}$, ranged over by $\mathsf{A}, \mathsf{A}', \ldots$, and a set of *token types* $\mathbb{T}$, ranged over by $\tau, \tau', \ldots$. We denote with $\mathbb{T}_f \subseteq \mathbb{T}$ the subset of tokens types that can be freely transferred between users, only assuming a sufficient balance of the sender ($\mathbb{T}_f$ includes e.g. the native blockchain tokens).

We render blockchain states as partial maps $\sigma \in \mathbb{A} \rightharpoonup (\mathbb{T} \rightharpoonup \mathbb{Q}^+)$, where $\sigma\mathsf{A}$ represents A's token balance (a partial map from token types to nonnegative rational numbers). Hereafter, we abbreviate $\sigma\mathsf{A}$ as σ_A. We use the standard notation $f\{{}^v/_x\}$ to update a partial map f at point x: namely, $f\{{}^v/_x\}(x) = v$, while $f\{{}^v/_x\}(y) = f(y)$ for $y \neq x$.

Given a partial map $f \in \mathbb{T} \rightharpoonup \mathbb{Q}^+$, a token type $\tau \in \mathbb{T}$ and a partial binary operation $\circ \in \mathbb{Q}^+ \times \mathbb{Q}^+ \rightharpoonup \mathbb{Q}^+$, we define the partial map $f \circ v : \tau$ as follows:

$$f \circ v : \tau = \begin{cases} f\{{}^{f(\tau)\,\circ\, v}/_\tau\} & \text{if } \tau \in \operatorname{dom} f \text{ and } f(\tau) \circ v \text{ is defined} \\ f\{{}^v/_\tau\} & \text{if } \tau \notin \operatorname{dom} f \end{cases} \tag{1}$$

We adopt the notation $v : \tau$ to denote v units of token τ throughout the paper.

We model the interaction between users and the blockchain as a state transition system, with labels ℓ which represent transactions. Our basic model has only one kind of transaction, $\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v : \tau)$, which represents the transfer of $v : \tau$ from A to B. Its effect on the state is specified by the following rule:

$$\frac{①\ \sigma_\mathsf{A}(\tau) \geq v \qquad ②\ \tau \in \mathbb{T}_f \qquad ③\ \sigma'_\mathsf{A} = \sigma_\mathsf{A} - v : \tau \qquad ④\ \sigma'_\mathsf{B} = \sigma_\mathsf{B} + v : \tau}{\sigma \xrightarrow{\mathsf{Trf}_\mathsf{A}(\mathsf{B}, v:\tau)} \sigma\{{}^{\sigma'_\mathsf{A}}/_\mathsf{A}\}\{{}^{\sigma'_\mathsf{B}}/_\mathsf{B}\}} \ [\textsc{Trf}]$$

We decorate rule preconditions with circled numbers, e.g. ①, to simplify their reference in the text. Rule [Trf] states that the transfer is permitted whenever the sender has a sufficient balance ①, and the transferred token type is free ②.

3.2 Lending pool states

We now extend our basic blockchain model with lending pools, focussing on the common features implemented by the main platforms. We make our model parametric w.r.t. platform-specific features, like e.g. interest rate models, and we abstract from some advanced features, like e.g. governance (see Section 7 for a discussion on the differences between our model and the existing platforms).

We model states Γ as terms of the form $\sigma \mid \pi \mid p$, where σ is the token balance of users, π is the lending pool state, and $p \in \mathbb{T}_f \to \mathbb{Q}^+$ models an oracle who prices the free tokens. Lending pool states π are triples (π_f, π_l, π_m), where:

$\mathsf{Dep}_{\mathsf{A}}(v : \tau)$	A deposits v units of a free token τ, receiving minted tokens
$\mathsf{Bor}_{\mathsf{A}}(v : \tau)$	A borrows v units of free token τ
Int	All loans accrue interest
$\mathsf{Rep}_{\mathsf{A}}(v : \tau)$	A repays v units on A's loan in τ
$\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$	A redeems v units of minted τ, receives deposited tokens
$\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$	A repays v units of B's loan in τ, seizing $v' : \tau'$ from B
$\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$	A transfers v units of minted τ to B
$\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$	A transfers v units of free τ to B

Table 2: Lending pool actions.

Actions	σ_{A}				σ_{B}			π_f		π_l B	π_m		p	
	τ_0	τ_1	τ_0'	τ_1'	τ_0	τ_1	τ_1'	τ_0	τ_1	τ_0	τ_0	τ_1	τ_0	τ_1
0. Initial State	100	–	–	–	–	50	–	–	–	–	–	–	1	1
1. $\mathsf{Dep}_{\mathsf{A}}(50 : \tau_0)$	**50**	–	**50**	–	–	50	–	50	–	–	τ_0'**:50**	–	1	1
2. $\mathsf{Dep}_{\mathsf{B}}(50 : \tau_1)$	50	–	50	–	–	0	**50**	50	50	–	τ_0':50	τ_1'**:50**	1	1
3. $\mathsf{Bor}_{\mathsf{B}}(30 : \tau_0)$	50	–	50	–	**30**	0	50	20	50	**30**	τ_0':50	τ_1':50	1	1
4. Int	50	–	50	–	30	0	50	20	50	**34**	τ_0':50	τ_1':50	1	1
5. $\mathsf{Rep}_{\mathsf{B}}(5 : \tau_0)$	50	–	50	–	**25**	0	50	**25**	50	**29**	τ_0':50	τ_1':50	1	1
6. Px	50	–	50	–	25	0	50	25	50	29	τ_0':50	τ_1':50	**1.3**	1
7. $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, 13 : \tau_0, 19 : \tau_1')$	**37**	–	50	**19**	25	0	**31**	**38**	50	**16**	τ_0':50	τ_1':50	1.3	1
8. $\mathsf{Rdm}_{\mathsf{A}}(10 : \tau_0')$	**48**	–	**40**	19	25	0	31	**27**	50	16	τ_0'**:40**	τ_1':50	1.3	1

Table 3: Interactions between two users and a lending pool.

- $\pi_f \in \mathbb{T}_{\mathsf{f}} \rightharpoonup \mathbb{Q}^+$ records the balance of free token types deposited in the LP;
- $\pi_l \in \mathbb{A} \rightharpoonup (\mathbb{T}_{\mathsf{f}} \rightharpoonup \mathbb{Q}^+)$ records the amount and type of tokens lent to users;
- $\pi_m \in \mathbb{T}_{\mathsf{f}} \rightharpoonup ((\mathbb{T} \setminus \mathbb{T}_{\mathsf{f}}) \times \mathbb{Q}^+)$ records the amount of tokens minted by the LP upon deposits. Namely, $\pi_m(\tau) = (\tau', n)$ means that n units of a token type τ', minted by the LP to represent claims of deposited tokens of type τ, are currently held by users. We require that different free tokens are associated to different minted tokens:

$$\pi_m(\tau_1) = (\tau', n_1) \ \wedge \ \pi_m(\tau_2) = (\tau', n_2) \implies \tau_1 = \tau_2 \tag{2}$$

We denote with $\mathbb{T}_\pi$ the set of tokens minted by a LP in state π. For a minted token $\tau' \in \mathbb{T}_\pi$, we denote with $u_\pi(\tau')$ the *underlying* free token. Formally:

$$\mathbb{T}_\pi = \{ fst(\pi_m(\tau)) \mid \tau \in \mathbb{T}_{\mathsf{f}} \} \qquad\qquad u_\pi(\tau') = \tau \quad \text{if } fst(\pi_m(\tau)) = \tau' \tag{3}$$

Note that (2) ensures that $u_\pi(\tau_1') \neq u_\pi(\tau_2')$ when $\tau_1' \neq \tau_2'$. We say that a state $\sigma \mid \pi \mid p$ is *initial* if π_f, π_l, π_m have empty domain, and $\mathrm{dom}\,\sigma_{\mathsf{A}} \subseteq \mathbb{T}_{\mathsf{f}}$ for all A.

3.3 An overview of lending pools behaviour

Lending pools support several actions, summarized in Table 2. Before formalizing their behaviour, we give some intuition through an example involving two users A and B (see Table 3). A and B start by depositing 50 units of free tokens τ_0 and τ_1, for which they receive equal amounts of freshly minted tokens τ_0' and τ_1'.

Next, B borrows $30 : \tau_0$. Here, the 50 minted tokens of type τ_1' in B's balance serve as *collateral* for the loan. The *collateralization* of B is the ratio between the *value* of B's balance of τ_1' and the value of B's loan of τ_0 (the value of a token balance is the product between the number of units of the token and its price). Assuming a minimum collateralization threshold of $C_{min} = 1.5$ and equal token prices for τ_0 and τ_1, B could borrow up to 33 units of τ_1, given the collateral of $50 : \tau_1'$. Nonetheless, B decides to leave some margin to manage future price volatility and the accrual of interest, which can both negatively affect collateralization. In action 4, interest accrues on the loan made by B. Here, the interest rate is 12%, so B's loan amount grows from 30 to 34 units of τ_1. In action 5, B repays 5 units of τ_0 to reduce the risk of becoming *liquidated*, which can occur when B's collateralization falls below the threshold $C_{min} = 1.5$.

Despite this effort, the price is updated in action 6, such that $p(\tau_0)$ increases by 30% relative to $p(\tau_1)$, thereby decreasing the relative value of B's collateral to B's loan. As a result, the collateralization of B drops below the threshold C_{min}. In action 7, A liquidates $13 : \tau_0$ of B's loan, restoring B's collateralization to C_{min}, and simultaneously seizing $19 : \tau_1'$ from B's balance. The exchange of $13 : \tau_0$ for $19 : \tau_1'$ implies a liquidation discount, which ensures that the liquidation is profitable for the user performing it.

In action 8, A then *redeems* $10 : \tau_0'$, receiving $11 : \tau_0$ in exchange. Here, each unit of τ_0' is now exchanged for more than 1 unit of τ_0, due to accrued interest.

3.4 Lending pool transitions

We now present the full set of rules which formalize the behaviour of lending pools. To illustrate them, we provide an extended running example (Tables 4–9).

Deposit A user A can deposit v units of a token τ by performing the transaction $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$, provided that the balance is sufficient ①. In return, A receives v' units of a token τ' minted by the LP. Upon the first deposit of τ, the LP creates a fresh (non-free) token type τ' ②; freshness ensures that condition (2) is preserved by the new state. For further deposits of τ, the LP mints new units of τ'. In both cases, the amount of minted units of τ' are recorded in the π_m component of the state ⑤. Note that premises ② and ⑤ require that τ must be a free token type. The amount v' ③ is the ratio between the deposited amount v and the exchange rate $ER_\pi(\tau)$ between τ and τ', defined in (4).

$$\dfrac{\textcircled{1}\ \sigma_{\mathsf{A}}(\tau) \geq v \qquad \textcircled{2}\ \tau' := \begin{cases} \text{fresh} \notin \mathbb{T}_{\mathsf{f}} & \text{if } \tau \notin \operatorname{dom} \pi_m \\ \pi_m(\tau) & \text{otherwise} \end{cases} \qquad \textcircled{3}\ v' := {}^{v}/_{ER_\pi(\tau)} \\ \textcircled{4}\ \pi_f' := \pi_f + v : \tau \qquad \textcircled{5}\ \pi_m' := \begin{cases} \pi_m\{{}^{(\tau', v')}/_{\tau}\} & \textit{if } \tau \notin \operatorname{dom} \pi_m \\ \pi_m\{{}^{(\tau', v''+v')}/_{\tau}\} & \textit{if } \pi_m(\tau) = (\tau', v'') \end{cases}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Dep}_{\mathsf{A}}(v:\tau)} \sigma\{{}^{\sigma_{\mathsf{A}} - v:\tau + v':\tau'}/_{\mathsf{A}}\} \mid (\pi_f', \pi_l, \pi_m') \mid p} \ [\textsc{Dep}]$$

The main idea of the exchange rate is that, while initially there is a 1/1 correspondence between minted and deposited tokens, when interest is accrued this relation changes to the benefit of lenders. For a free token τ, the exchange

Table 4: Running example: deposit actions

Actions	σ_{A}				σ_{B}				σ_{C}		π_f			π_m		
	τ_0	τ_1	τ_0'	τ_1	τ_0	τ_2	τ_0'	τ_2'	τ_2	τ_2'	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2
0. Initial state	100	300	-	-	50	50	-	-	100	-	-	-	-	-	-	-
1. $\mathsf{Dep}_{\mathsf{A}}(100 : \tau_0)$	**0**	300	**100**	-	50	50	-	-	100	-	**100**	-	-	τ_0'**:100**	-	-
2. $\mathsf{Dep}_{\mathsf{A}}(150 : \tau_1)$	0	**150**	100	**150**	50	50	-	-	100	-	100	**150**	-	τ_0':100	τ_1'**:150**	-
3. $\mathsf{Dep}_{\mathsf{B}}(50 : \tau_0)$	0	150	100	150	**0**	50	**50**	-	100	-	**150**	150	-	τ_0'**:150**	τ_1':150	-
4. $\mathsf{Dep}_{\mathsf{B}}(50 : \tau_2)$	0	150	100	150	0	**0**	50	**50**	**100**	-	150	150	**50**	τ_0':150	τ_1':150	τ_2'**:50**
5. $\mathsf{Dep}_{\mathsf{C}}(100 : \tau_2)$	0	150	100	150	0	0	50	50	**0**	100	150	150	**150**	τ_0':150	τ_1':150	τ_2'**:50**

rate $ER_\pi(\tau)$ represents the share of deposited units of τ over the units of the associated minted tokens. If any loans remain pending, not all minted tokens can be redeemed, as only a fraction of the deposited free tokens remain in the LP balance. Formally:

$$ER_\pi(\tau) = \frac{\pi_f(\tau) + \sum_{\mathsf{A}} (\pi_l\ \mathsf{A})\ \tau}{snd(\pi_m(\tau))}\ \textit{if}\ \pi_f(\tau) > 0 \qquad ER_\pi(\tau) = 1\ \textit{if}\ \pi_f(\tau) = 0 \tag{4}$$

where we assume that the items A for which $\pi_l\ \mathsf{A}$ or $(\pi_l\ \mathsf{A})\tau$ are undefined do not contribute to the summation (we will adopt this convention through the paper).

Table 4 exemplifies users depositing funds to the LP. In transaction 1, A deposits 100 units of τ_0. Since this is the first deposit of τ, the LP mints exactly 100 units of a *fresh* token type, say τ_0', and transfers these units to A. In transaction 2, A deposits 150 units of τ_1; similarly to the previous case, A receives 150 units of a fresh token type τ_1'. In transaction 3, B deposits 50 units of τ_0. Since τ_0 was already deposited, the LP mints 50 units of the existing token type τ_0', and transfers them to B. Finally, in transactions 4 and 5 B and C deposit units of τ_2; after that, the balances of tokens τ_0, τ_1, τ_2 in the LP total 150 units.

Borrow Any user can borrow units of a free token type τ from the LP, provided that the LP has a sufficient balance of τ ①, and that the user has enough minted tokens to use as collateral ④. More specifically, we require that the *collateralization* of the user is above a constant threshold $C_{min} > 1$.

$$\frac{\text{①}\ \pi_f(\tau) \geq v > 0 \qquad \text{②}\ f_{\mathsf{A}} = \begin{cases} \pi_l\mathsf{A} + v : \tau & \text{if } \mathsf{A} \in \operatorname{dom} \pi_l \\ \{^{v}/_{\tau}\} & \text{otherwise} \end{cases} \\ \text{③}\ \pi' := (\pi_f - v : \tau,\ \pi_l\{^{f_{\mathsf{A}}}/_{\mathsf{A}}\},\ \pi_m) \qquad \text{④}\ C_{\sigma|\pi'|p}(\mathsf{A}) \geq C_{min}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Bor}_{\mathsf{A}}(v:\tau)} \sigma\{^{\sigma_{\mathsf{A}}+v:\tau}/_{\mathsf{A}}\} \mid \pi' \mid p}\ [\text{Bor}]$$

To define the collateralization of users, we introduce a few auxiliary notions. The value $V^l(\mathsf{A})$ of A's loans is the sum (over all free token types τ) of the *value* of τ-tokens lent to A (the value is the product between token amount and price). For instance, if A has borrowed only 10 tokens of type τ, and the price of $1 : \tau$ is $2 : \tau_n$, then the value of A's loan is $20 : \tau_n$. Formally:

$$V^l_\Gamma(\mathsf{A}) \ = \ \sum_{\tau \in \mathbb{T}_f} (\pi_l\ \mathsf{A})\tau \cdot p(\tau) \qquad \text{if } \Gamma = \sigma \mid \pi \mid p \tag{5}$$

Table 5: Running example: borrow actions

Actions	σ_{B}					σ_{C}				π_l			π_f			p			C_Γ	
										B	C									
	τ_0	τ_1	τ_2	τ_0'	τ_1'	τ_0	τ_1	τ_2	τ_2'	τ_1	τ_0	τ_1	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
5. $\mathsf{Dep}_{\mathsf{C}}(100 : \tau_2)$	0	-	0	50	50	-	-	0	100	-	-	-	150	150	**150**	1	1	1	-	-
6. $\mathsf{Bor}_{\mathsf{B}}(50 : \tau_1)$	0	**50**	0	50	50	-	-	0	100	**50**	-	-	150	**100**	150	1	1	1	**2.0**	-
7. $\mathsf{Bor}_{\mathsf{C}}(30 : \tau_0)$	0	50	0	50	50	**30**	-	0	100	50	**30**	-	**120**	100	150	1	1	1	2.0	**3.3**
8. $\mathsf{Bor}_{\mathsf{C}}(30 : \tau_1)$	0	50	0	50	50	30	**30**	0	100	50	30	**30**	120	**70**	150	1	1	1	2.0	**1.7**

The value $V^m(\mathsf{A})$ of minted tokens held by A is the summation (over all minted token types τ) of the *value* of A's balance of minted tokens. To determine the value of a minted token τ', its price is equated to that of the underlying free token τ, as minted tokens do not exist in the domain of p:

$$V^m_\Gamma(\mathsf{A}) \;=\; \textstyle\sum_{\tau \in \mathbb{T} \setminus \mathbb{T}_{\mathsf{f}}} \sigma_{\mathsf{A}}(\tau) \cdot ER_\pi(u_\pi(\tau)) \cdot p(u_\pi(\tau)) \qquad \text{if } \Gamma = \sigma \mid \pi \mid p \tag{6}$$

The collateralization of a user is the ratio of the value of minted to lent tokens:

$$C_\Gamma(\mathsf{A}) \;=\; V^m_\Gamma(\mathsf{A}) \;/\; V^l_\Gamma(\mathsf{A}) \tag{7}$$

We exemplify Bor transactions in Table 5. Users B and C borrow amounts of τ_0 and τ_1 at steps 6–8, keeping their collateralization above C_{min}, which is assumed to be 1.5. C's collateralization decreases from 3.3 to 1.7 upon step 8: this is due to the increase in $V^l(\mathsf{C})$, whilst $V^m(\mathsf{C})$ remains constant at 100.

As we have seen, user collateralization depends on the amount of minted tokens he possesses, the amount of tokens, and the price of all tokens involved. Therefore, collateralization is potentially sensitive to all actions that can affect those values. This includes both interest accrual and changes in token prices (which are unpredictable). Borrowers must therefore maintain a safety margin in order to protect against potential liquidation.

Interest Accrual Interest accrual models the periodic application of interest to loan amounts and can be executed in any state. The action applies a token-specific interest $I_\pi(\tau)$ to each loan, updating the π_l mapping for *all* users.

$$\frac{\pi_l'(\mathsf{A}) := f_{\mathsf{A}}' \;\text{ if } \mathsf{A} \in \operatorname{dom} \pi_l, \text{ where } \; f_{\mathsf{A}}'(\tau) := (I_\pi(\tau) + 1) \cdot (\pi_l\, \mathsf{A})\tau \;\text{ if } \tau \in \operatorname{dom}(\pi_l \mathsf{A})}{\sigma \mid \pi \mid p \;\xrightarrow{\mathsf{Int}}\; \sigma \mid (\pi_f, \pi_l', \pi_m) \mid p} \;[\textsc{Int}]$$

Existing lending pool platforms deploy different algorithmic interest rate models [47]. We leave our model parametric w.r.t. interest rates, and only require that the interest rate is positive, a property that all models in [47] satisfy:

$$I_\pi(\tau) > 0 \tag{8}$$

We extend our running example with three interest updates in Table 6, resulting in the increase of all loan amounts. Each subsequent execution of Int *decreases* the collateralization of users B and C, since the V^l of both borrowers *increases* as interest is applied (7).

Table 6: Running example: interest accrual

Actions	π_l			I_π			p			C_Γ	
	B	C									
	τ_1	τ_0	τ_1	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
8. $\mathsf{Bor}_{\mathsf{C}}(30 : \tau_1)$	50	30	**30**	2.0%	5.3%	0%	1	1	1	2.00	**1.67**
9. Int	**53**	**31**	**32**	2.1%	5.5%	0%	1	1	1	**1.89**	**1.59**
10. Int	**56**	**32**	**34**	2.1%	5.6%	0%	1	1	1	**1.79**	**1.52**
11. Int	**59**	**33**	**36**	2.2%	5.8%	0%	1	1	1	**1.69**	**1.45**

Table 7: Running example: repay actions

Actions	σ_{A}				σ_{B}					σ_{C}				π_f			π_l			C_Γ	
																	B	C			
	τ_0	τ_1	τ_0'	τ_1'	τ_0	τ_1	τ_2'	τ_0'	τ_2'	τ_0	τ_1	τ_2	τ_2'	τ_0'	τ_1'	τ_2'	τ_1	τ_0	τ_1	B	C
11. Int	0	150	100	150	0	50	0	50	50	30	30	0	100	120	70	150	**59**	**33**	**36**	**1.7**	**1.5**
12. $\mathsf{Rep}_{\mathsf{C}}(15 : \tau_0)$	0	150	100	150	0	50	0	50	50	**15**	30	0	100	**135**	70	150	59	**18**	36	1.7	**1.9**

Repay A user with a loan can repay part of it by executing a Rep transaction:

$$\frac{\text{①}\ \sigma_{\mathsf{A}}(\tau) \geq v > 0 \qquad \text{②}\ (\pi_l\, \mathsf{A})\, \tau \geq v \qquad \text{③}\ \pi_l' = \pi_l\{^{\pi_l \mathsf{A} - v:\tau}/_{\mathsf{A}}\}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Rep}_{\mathsf{A}}(v:\tau)} \sigma\{^{\sigma_{\mathsf{A}} - v:\tau}/_{\mathsf{A}}\} \mid (\pi_f + v : \tau,\ \pi_l',\ \pi_m) \mid p} \text{[Rep]}$$

This increases the collateralization of the repaying user, as V^l is reduced (7). Users must always maintain a sufficient collateralization, to cope with adverse effects of interest accruals and price updates.

In Table 7, C is suffering from low collateralization after the last interest accrual in transaction 11. Here, $C_\Gamma(\mathsf{C})$ is equal to $C_{min} = 1.5$. The subsequent repayment of 15 units of τ_0 increases C's collateralization back to 1.9.

Redeem A user without any loans can redeem minted tokens τ ① for the underlying tokens if enough units of $u_\pi(\tau)$ remain in the LP ②. A user with a *non-zero* loan amount of any token can only redeem minted tokens such that the resulting collateralization is not below C_{min} ③. This constraint does not apply to users without loans, as minted tokens are not used as collateral.

$$\frac{\begin{array}{l} \text{①}\ \sigma_{\mathsf{A}}(\tau) \geq v > 0 \qquad v' := v \cdot ER_\pi(u_\pi(\tau)) \qquad \text{②}\ \pi_f(u_\pi(\tau)) \geq v' \\ \text{③}\ (\exists \tau'.(\pi_l \mathsf{A})\tau' > 0) \Rightarrow C_{\sigma'|\pi'|p}(\mathsf{A}) \geq C_{min} \qquad \sigma_{\mathsf{A}}' := \sigma_{\mathsf{A}} - v : \tau + v' : u_\pi(\tau) \\ \pi_f' := \pi_f - v' : u_\pi(\tau) \qquad \pi_m' := \pi_m\{^{(\tau, v'' - v)}/_{u_\pi(\tau)}\} \text{ where } (\tau, v'') := \pi_m(u_\pi(\tau)) \end{array}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Rdm}_{\mathsf{A}}(v:\tau)} \sigma\{^{\sigma_{\mathsf{A}}'}/_{\mathsf{A}}\} \mid (\pi_f',\ \pi_l,\ \pi_m') \mid p} \text{[Rdm]}$$

We exemplify Rdm transactions in Table 8. From Table 7, B has a non-zero loan amount, hence he can only redeem $11 : \tau_2'$ before his collateralization decreases to $C_{min} = 1.5$, at which B cannot further redeem. Since A has no loans, she can redeem as many tokens τ_0' as the LP balance permits. For A's redeeming of $50 : \tau_0'$ for $51 : \tau_0$ the exchange rate is > 1, because of the accrued interest during the prior execution of Int. By contrast, the exchange rate for B is 1, as no loan exists on τ_2', and thus no interest was accrued. The tokens τ_2' and τ_0' returned to the LP by B and A are burnt and subtracted from π_m.

Table 8: Running example: redeem actions

Actions	σ_{A}				σ_{B}					π_f			π_m			C_Γ	
	τ_0	τ_1	τ_0'	τ_1'	τ_0	τ_1	τ_2	τ_0'	τ_2'	τ_0	τ_1	τ_2	τ_0	τ_1	τ_2	B	C
12. $\mathsf{Rep}_{C}(15 : \tau_0)$	0	150	100	150	0	50	0	50	50	**135**	70	150	τ_0':150	τ_1':150	τ_2':150	1.7	**1.9**
13. $\mathsf{Rdm}_{B}(11 : \tau_2')$	0	150	100	150	0	50	**11**	50	**39**	135	70	**139**	τ_0':150	τ_1':150	τ_2'**:139**	**1.5**	1.9
14. $\mathsf{Rdm}_{A}(50 : \tau_0')$	**51**	150	**50**	150	0	50	11	50	39	**84**	70	139	τ_0'**:100**	τ_1':150	τ_2':139	1.5	1.9

Liquidation When the collateralization of a user B is below the threshold C_{min} $\textcircled{6}$, another user A can *liquidate* part of B's loan $\textcircled{8}$, in return for a discounted amount of minted tokens *seized* from B $\textcircled{10}$. A can execute Liq if it has enough balance to repay a fraction of the lent token $\textcircled{1}$, and if B has a sufficient balance of seizable, minted tokens $\textcircled{4}$. The maximum seizable amount is bounded by $\textcircled{4}$ or the resulting collateralization of B $\textcircled{7}$, which cannot exceed C_{min}. After this threshold, B's collateralization is restored, and B is no longer liquidatable.

$$\frac{\begin{array}{lll} \textcircled{1}\ \sigma_{A}(\tau) \geq v & \textcircled{2}\ (\pi_l\, B)\, \tau \geq v & \textcircled{3}\ \tau' \in \mathbb{T}_\pi \\ \textcircled{4}\ \sigma_{B}(\tau') \geq v' & \textcircled{5}\ v' = v \cdot \frac{p(\tau)}{p(u_\pi(\tau'))} \cdot r_{liq} & \\ \textcircled{6}\ C_{\sigma|\pi|p}(B) < C_{min} & \textcircled{7}\ C_{\sigma'|\pi'|p}(B) \leq C_{min} & \\ \textcircled{8}\ \pi_l' := \pi_l\, B - v : \tau & \textcircled{9}\ \sigma_{A}' := \sigma_{A} - v : \tau + v' : \tau' & \textcircled{10}\ \sigma_{B}' := \sigma_{B} - v' : \tau' \end{array}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Liq}_{A}(B, v:\tau, v':\tau')} \sigma\{\sigma_{A}'/A\}\{\sigma_{B}'/B\} \mid (\pi_f, \pi_l', \pi_m) \mid p} \text{[LIQ]}$$

For the execution of Liq, where $v : \tau$ and $v' : \tau'$ are repaid and seized amounts respectively, the constraint on v and v' is given in $\textcircled{5}$, where:

$$C_{min} > r_{liq} > 1 \tag{9}$$

The constraint $r_{liq} > 1$ implies a discount applied to the seized amount received by the liquidator, as more value is received than repaid:

For the liquidations in Table 9, we set $r_{liq} = 1.1$. After the price update in action 15, both B and C are undercollateralized. C is liquidated by A in transaction 16, which restores $C_\Gamma(C)$ to 1.5. By contrast, $C_\Gamma(B)$ is 0.9 after the price update. Subsequent liquidations by A seize units of both τ_0' and τ_2' until B's balance of minted tokens is empty. However, B still has a loan amount of $11 : \tau_1$, which is *unrecoverable*. Both B and potential liquidators have no incentive to repay or liquidate given the lack of collateral.

Table 9: Running example: liquidation actions

Actions	σ_{A}					σ_{B}					σ_{C}				π_f			π_l			p		C_Γ	
																		B	C					
	τ_0	τ_1	τ_0'	τ_1'	τ_2'	τ_0	τ_1	τ_2	τ_0'	τ_2'	τ_0	τ_1	τ_2	τ_2'	τ_0	τ_1	τ_2	τ_1	τ_0	τ_1	τ_0, τ_1	τ_2	B	C
15. Px	51	150	50	150	-	0	50	11	50	39	15	30	0	100	84	70	139	59	18	36	1	**1.7**	**0.9**	**1.3**
16. $\mathsf{Liq}_{A}(C, 27 : \tau_1, 50 : \tau_2')$	51	**123**	50	150	**50**	0	50	11	50	39	15	30	0	**50**	84	**97**	139	59	18	**9**	1	1.7	0.9	**1.5**
17. $\mathsf{Liq}_{A}(B, 27 : \tau_1, 50 : \tau_0')$	51	**96**	**100**	150	50	0	50	11	**0**	39	15	30	0	50	84	**124**	139	**32**	18	9	1	1.7	**0.7**	1.5
18. $\mathsf{Liq}_{A}(B, 21 : \tau_1, 39 : \tau_2')$	51	**75**	100	150	**89**	0	50	11	0	**0**	15	30	0	50	84	**145**	139	**11**	18	9	1	1.7	**0**	1.5

Transfer of minted tokens Minted tokens can be transferred between users. Unlike free tokens transfers (rule [Trf] at page 4), this requires that the sender retains a collateralization level above C_{min}.

$$\frac{\sigma_{\mathsf{A}}(\tau) \geq v \qquad \tau \in \mathbb{T}_{\pi} \qquad \sigma' = \sigma\{{}^{\sigma_{\mathsf{A}} - v:\tau}/_{\mathsf{A}}\}\{{}^{\sigma_{\mathsf{B}} + v:\tau}/_{\mathsf{B}}\} \qquad C_{\sigma' \mid \pi \mid p}(\mathsf{A}) \geq C_{min}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v:\tau)} \sigma' \mid \pi \mid p} \text{[Mtrf]}$$

Price updates Finally, the price oracle can be updated non-deterministically:

$$\sigma \mid \pi \mid p \xrightarrow{\mathsf{Px}} \sigma \mid \pi \mid p' \quad \text{[Px]}$$

4 Fundamental properties of lending pools

We now establish some fundamental properties of lending pools. These properties hold for all *reachable states*, i.e. states Γ such that $\Gamma_0 \to^* \Gamma$ for some initial Γ_0.

The first property states that the component π_m of the state correctly records the balance of all minted tokens held by users. This is formalized by Lemma 1.

Lemma 1. *Let $\sigma \mid \pi \mid p$ be a reachable state. For all $\tau \in \mathbb{T}_{\pi}$:*

$$\textstyle\sum_{\mathsf{A}} \sigma_{\mathsf{A}}(\tau) \;=\; snd(\pi_m(u_{\pi}(\tau))) \tag{10}$$

Another crucial property is that the exchange rate of a minted token must either strictly increase, when users are borrowing the underlying token, or remain stable otherwise. This guarantees a depositor that her deposit will grow.

Lemma 2. *Let $\sigma \mid \pi \mid p$ be a reachable state, let $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$, and let $\tau \in \mathbb{T}_{\pi}$. Then: (a) if $\ell = \mathsf{Int}$ and $\exists \mathsf{A} : (\pi_l\, \mathsf{A})\, \tau > 0$, then $ER_{\pi}(\tau) < ER_{\pi'}(\tau)$; (b) otherwise, $ER_{\pi}(\tau) = ER_{\pi'}(\tau)$.*

As a direct consequence of Lemma 2 we have that, in any computation, the exchange rate of any token type is increasing.

We also establish a preservation property of the *supply* of any free token τ, i.e. the sum of all user balances and lending balance of τ:

$$sply_{\Gamma}(\tau) \;=\; \pi_f(\tau) + \textstyle\sum_{\mathsf{A}} \sigma_{\mathsf{A}}(\tau) \qquad \text{if } \Gamma = \sigma \mid \pi \mid p \tag{11}$$

Lemma 3 establishes that the supply of any free token is constant.

Lemma 3. *Let $\Gamma_0 \to^* \Gamma$, for Γ_0 initial. For all $\tau \in \mathbb{T}_{\mathsf{f}}$: $sply_{\Gamma_0}(\tau) = sply_{\Gamma}(\tau)$.*

While the supply of an LP remains constant, users act to increase their own share. We define the *net worth* $W_{\Gamma}(\mathsf{A})$ as the value of the amount of tokens in A's wallet or lent by A, minus the value of A's loans. Formally, if $\Gamma = \sigma \mid \pi \mid p$:

$$W_{\Gamma}(\mathsf{A}) = \textstyle\sum_{\tau \in \mathbb{T}_{\mathsf{f}}} \big(\sigma_{\mathsf{A}}(\tau) + \sigma_{\mathsf{A}}(fst(\pi_m(\tau)) \cdot ER_{\pi}(\tau) - (\pi_l\, \mathsf{A}\, \tau)\big) \cdot p(\tau)$$

The net worth of a user can be increased in short or long sequences of transitions. In general, there is no winning strategy (in the game-theoretic sense) for a single user that wants to increase her net worth, unless she can control price updates: actually, with just one price update the net worth of any user can be reduced to 0. However, under certain conditions, winning strategies can be found. We consider first a simple 1-player game where a user can choose her next action to improve her net worth in the next state. Here, liquidation is the only action by an honest user A that increases her net worth in just one transition.

Lemma 4. *Let Γ be a reachable state and $\Gamma \xrightarrow{\ell} \Gamma'$ with $\ell = \_{\mathsf{A}}(\cdots)$. Then: (a) $W_{\Gamma}(\mathsf{A}) > W_{\Gamma'}(\mathsf{A})$ if $\ell = \mathsf{Liq}_{\mathsf{A}}(\cdots)$; (b) $W_{\Gamma}(\mathsf{A}) = W_{\Gamma'}(\mathsf{A})$ otherwise.*

Since this is the winning strategy for all users, but liquidations may be limited by loan or collateral amounts, an adversary who has the power to drop or re-order transactions can potentially monopolize liquidations for itself. We refer to Section 6 for additional discussion of such attacks.

We now consider a slightly extended game, where A guesses that the next (adversarial) action is going to be ℓ, resulting in $\Gamma_0 \xrightarrow{\ell} \Gamma_1$ but can still perform an action ℓ' before ℓ, resulting in $\Gamma_0 \xrightarrow{\ell'} \Gamma_0' \xrightarrow{\ell} \Gamma_1'$. The goal of A is to choose ℓ' such that $W_{\Gamma_1'}(\mathsf{A}) > W_{\Gamma_1}(\mathsf{A})$. We show that if $\ell = \mathsf{Int}$, i.e. A is expecting interest accrual to happen next, her choice is limited to deposits, repays and liquidations.

Lemma 5. *Let Γ_0 be a reachable state, and let $\Gamma_0 \xrightarrow{\ell} \Gamma_1$ and $\Gamma_0 \xrightarrow{\ell'} \Gamma_0' \xrightarrow{\ell} \Gamma_1'$ be such that $\ell = \mathsf{Int}$ and $\ell' = \_{\mathsf{A}}(\cdots)$. Then: (a) $W_{\Gamma_1'}(\mathsf{A}) \geq W_{\Gamma_1}(\mathsf{A})$ if ℓ' is one of $\mathsf{Liq}_{\mathsf{A}}(\cdots)$ or $\mathsf{Dep}_{\mathsf{A}}(\cdots)$ or $\mathsf{Rep}_{\mathsf{A}}(\cdots)$; (b) $W_{\Gamma_1'}(\mathsf{A}) \leq W_{\Gamma_1}(\mathsf{A})$ otherwise.*

Overall, Lemmas 4 and 5 determine the set of actions to consider (together with their parameter) to maximize improvements in short-term net worth.

5 Lending pool safety, vulnerabilities and attacks

We discuss further properties of lending pools, focusing on potential risks which could lead to unsecured loans or exploitations by malicious actors. In particular, we focus on user collateralization and the availability of free token funds in lending pools (utilization). In the case where these can be targeted by an attacker, the motivation is to limit the lending pool functionality (denial-of-service) or cause the victim to incur losses, which in some cases may imply a gain for the attacker. We restrict our attention to attacker models where the attacker has the ability to perform some of the actions of the LP model, or even update the price oracle. More powerful attackers that can drop or reorder transactions are discussed in Section 6.

5.1 Collateralization bounds and risks

The lending pool design assumes that loans are *secured* by collateral: liquidations thereof are incentivised in order to recover loans should the borrowing users fail to repay. However, collateral liquidation is exposed to risks. Firstly, the incentive to liquidate is only effective, if the liquidator values the seized collateral higher than the value of the repaid loan amount, implying a profit. Secondly, large fluctuations in token price may reduce the relative value of the collateral such that the loan becomes partially unrecoverable. Furthermore, an attacker with the ability to update token prices can force users to become undercollateralized and then seize the collateral of victims without repaying any loans.

LP-minted token risk The lending pool must determine the appropriate levels of collateralization based on token prices given by the price oracle. However, the value of LP-minted tokens is indeterminable since they are not featured in $\mathrm{dom}\,(p)$. The definition of collateralization in eq. (7) values units of LP-minted tokens at the same price as their underlying counterpart, as do lending pool implementations [10, 21]. However, since LP-minted tokens represent claims on free tokens, which are only redeemable if sufficient funds remain in the lending pool (② in [RDM]), it is possible that users value minted tokens at a lower price than their underlying counterparts when LP-minted tokens cannot be redeemed during times of low lending pool funds (utilization). Lending pool designs do not account for this and thus run the risk of incorrectly pricing LP-minted tokens and collateral.

Safe collateralization Assuming a correct valuation of LP-minted tokens, undercollateralized loans should be swiftly liquidated, given the incentivization provided by the liquidation discount. Furthermore, the user collateral value should be high enough, such that the user's loan amount is sufficiently repaid by liquidations to recover the user collateralization back to C_{min}. Therefore, we introduce two notions of safe collateralization.

Inspired by [48], we say that a LP state is *ε-collateralization safe* when the ratio of the loan value of undercollateralized accounts to the total loan value of the lending pool is below the threshold ε:

$$\frac{\sum_{C_\Gamma(\mathsf{A})<C_{min}} V^l_\Gamma(\mathsf{A})}{\sum_{\mathsf{A}} V^l_\Gamma(\mathsf{A})} \leq \varepsilon \qquad (12)$$

If the liquidation incentive is effective, a value below ε should not persist, as users are quick to execute liquidations. The efficiency of lending pool liquidations has been studied in [50]. We note that sufficiently large volumes of seized collateral which are immediately sold on external markets may delay further liquidations, as investigated in [46], due to the external market's finite capacity to absorb such a sell-off.

However, *ε-collateralization safety* does not account for undercollateralized loans which are *non-recoverable*, as previously illustrated in the example of Table 9. The set of non-recoverable, undercollateralized accounts are those with a collateralization below r_{liq}. The non-recoverable loan value of an account is given by V^{nrl}_Γ. It represents the remaining loan value of a user A should it be fully liquidated, such that no further collateral can be seized.

$$V_\Gamma^{nrl}(\mathsf{A}) = \begin{cases} V_\Gamma^{l}(\mathsf{A}) - \frac{V_\Gamma^{m}(\mathsf{A})}{r_{liq}} & \textit{iff } C_\Gamma(\mathsf{A}) < r_{liq} \\ 0 & \textit{otherwise} \end{cases} \tag{13}$$

Equation (13) illustrates that for the case where an account collateralization is below r_{liq}, the discounted value of the collateral can no longer equal or exceed the remaining loan value, a consequence of (7) and (9). We say that a LP state is *strongly ε-collateralization safe* when the fraction of the total loan value of a lending pool which is not recoverable is below ε:

$$\frac{\sum_{\mathsf{A}} V_\Gamma^{nrl}(\mathsf{A})}{\sum_{\mathsf{A}} V_\Gamma^{l}(\mathsf{A})} \leq \varepsilon \tag{14}$$

The condition (14) is actually stronger than (12), i.e. if a state is strongly ε-collateralization safe, then it is also ε-collateralization safe. Given equal denominators of (12) and (14), this is a consequence of comparing numerators: here, it can be observed that the numerator of (12) is greater than that of (14), as $V_\Gamma^{l}(\mathsf{A})$ is necessarily greater than $V_\Gamma^{nrl}(\mathsf{A})$ by definition and the set $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < C_{min}\}$ is a superset of $\{\mathsf{A} \mid C_\Gamma(\mathsf{A}) < r_{liq}\}$ (13).

Strong price volatility is a risk to ε-collateralization safety, as a sharp drop in price can immediately reduce a previously sufficiently collateralized user to become undercollateralized below the threshold of C_{min}: such an immediate drop leaves the user with no opportunity to maintain its collateralization with repayments.

Attacks on safe collateralization Malicious agents which can perform price updates can therefore influence the evolution of the LP to lead it to a state that is not ε-collateralization safe or strongly ε-collateralization safe.

For example, a malicious agent controlling the price oracle could act as follows. First, she would perform price updates to push any account collateralization below C_{min}, such that it becomes undercollateralized. The attacker can then perform liquidations on these accounts and benefit from the discount resulting from both the price update and r_{liq}. The attacker has maximized her profits by updating p such that $V_\Gamma^{l}(\mathsf{B})$ in (7) is zero, where B is an account under attack. In this case, $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$ can be performed with $v = 0$, and repeated liquidations can be executed to seize the full balance of B's LP-tokens.

As a matter of fact, a recent failure of the oracle price feed utilized by the Compound lending pool implementation lead to \$100M of collateral being (incorrectly) liquidated [19]: though it is unclear whether this was an intentional exploit, it illustrates the feasibility of such a price oracle attack.

5.2 Utilization bounds and risks

The notion of *utilization* plays a fundamental role in the incentive model of lending pools as explained in [47]. As a matter of fact, it is often used as a key parameter of interest rate models in implementations [11,22] and literature [47]. The utilization of a token type in a lending pool is the fraction of previously deposited funds currently lent to borrowing users. Formally:

$$U_\pi(\tau) \;=\; \frac{\sum_{\mathsf{A}} (\pi_l\, \mathsf{A})\, \tau}{\pi_f(\tau) + \sum_{\mathsf{A}} (\pi_l\, \mathsf{A})\, \tau} \tag{15}$$

Over- and under-utilization The value of $U_\pi(\tau)$ ranges between 0 and 1. We say that τ is *under-utilized* if its utilization is 0 and *over-utilized* when it is 1. We say that an LP state is under(over)-utilized if there is at least one under(over)-utilized token.

Under-utilization occurs when some units of τ have been deposited, but not lent to any user. This implies that action Int does not increase the loan value of any account, so that the exchange rate of τ in (4) remains constant, thereby not resulting in any gain for lenders.

On the other hand, over-utilization occurs when some users have borrowed τ, but the lending pool has no deposited funds of τ. In this case users can neither borrow or redeem.

Under- and over-utilization are not desirable and should be avoided. An optimal utilization rate eq. (15) of a free token type τ strikes a balance between the competing objectives of interest maximization and the ability for users to borrow or redeem tokens of type $\tau' = \mathit{fst}(\pi_m(\tau))$. In particular, the lending pool interest rate models described in [47] intend to incentivize actions of both borrowers and lenders to discover a utilization equilibrium between under- and over-utilization. Informally, this is achieved with interest rate models which rise and fall with utilization: increasing utilization and interest rates incentivize deposits and repayment of loans. Decreasing utilization and interest rates incentivize redeems and additional loan borrowing.

We proceed to discuss under- and overutilization attacks: here, we note that the former is weaker than the latter, as funds can still be safely recovered in a case of underutilization.

Under-utilization attacks Under-utilization can be achieved by a group of malicious users interested in reducing interest accrual for depositors or discouraging borrowing of a token τ. Here, the attacker can temporarily reduce utilization by repaying large amounts of loans, though the effectiveness of this approach will depend on the amounts of τ repaid by the attacker, as a lowered utilization can also reduce the interest rate (in certain models [47]), thereby incentivizing additional borrowing. An attacker which can update the price oracle can lower the collateralization of borrowers arbitrarily, thereby incentivizing repayments and liquidations to target lower utilization of specific tokens.

Over-utilization attacks Over-utilization could be achieved by a group of malicious users interested in preventing redeems or borrows of τ. The malicious users can do this by redeeming all units of τ while avoiding loans to be repaid or liquidated. We illustrate an over-utilization attack in Table 10. Here, users A and C initially hold the entire supply of τ_0 in their balances. A colludes with B

Actions	σ_{A}		σ_{A}			σ_{C}		π_f		π_l B	π_m		U_π	
	τ_0	τ_0'	τ_0	τ_1	τ_1'	τ_0	τ_0'	τ_0	τ_1	τ_0	τ_0	τ_1	τ_0	τ_1
0. Initial State	100	–	–	100	–	50	–	–	–	–	–	–	–	–
1. $\mathsf{Dep}_{\mathsf{A}}(100 : \tau_0)$	**0**	**100**	–	100	–	50		**100**	–	–	τ_0'**:100**	–	**0**	–
2. $\mathsf{Dep}_{\mathsf{B}}(100 : \tau_1)$	0	100	–	**0**	**100**	50		100	**100**	–	τ_0':100	τ_1'**:100**	0	**0**
3. $\mathsf{Bor}_{\mathsf{B}}(50 : \tau_0)$	0	100	**50**	0	100	50		50	100	**50**	τ_0':100	τ_1':100	**0.5**	0
4. $\mathsf{Dep}_{\mathsf{C}}(50 : \tau_0)$	0	100	50	0	100	**0**	**50**	100	100	50	τ_0'**:150**	τ_1':100	**0.3**	0
5. $\mathsf{Rdm}_{\mathsf{A}}(100 : \tau_0')$	**100**	**0**	50	0	100	0	50	**0**	100	50	τ_0'**:50**	τ_1':100	**1.0**	0

Table 10: Over-utilization attack.

to steal C's balance of τ_0: in actions 0-2, both A and B deposit units of $100 : \tau_0$ and $100 : \tau_1$ respectively. B utilizes her balance of $100 : \tau_1'$ as collateral to borrow $50 : \tau_0$ from the lending pool in action 3. At this point, A and B are acting as lender and borrower of τ_0, for which the utilization is 0.5. C, having observed an opportunity to earn interest on τ_0 decides to deposit $50 : \tau_0$ in action 4. However, user A still has a balance of redeemable $100 : \tau_0'$, which she redeems in action 5. Now, users A and B have removed all units of τ_0 from the lending pool, pushing the utilization of τ_0 to 1 and preventing C from redeeming her funds. Of course, user B cannot redeem his balance of τ_1' since her loan has not been repaid, but this can be considered the cost of the attack.

6 DeFi archetypes: lending pools and beyond

We now discuss the interplay between lending pools and other DeFi applications, like algorithmic stable coins, automatic market makers, margin trading and flash loans, which are all predominantly deployed on the Ethereum blockchain [38].

Lending pools The emergent behaviour of lending pools in times of high price volatility is examined in [46] by simulation of a lending pool liquidation model. Here, a large price drop can cause many accounts to become undercollateralized: assuming liquidators sell off collateral at an external market for units of the repaid token type, the authors suggest that limited market demand for collateral tokens may prevent liquidations from being executed, thereby posing a risk to ε*-collateralization safety* as we have defined in eqs. (12) and (14).

Lending pool behaviour at the user level is modelled in [48], which simulates agents interacting with the Compound implementation to examine the evolution of *liquidatable* and *undercollateralized* debt, notions similar to *(strong)* ε*-collateralization safety* (12) (14). [39, 40] examine the competition for user deposits between staking in proof-of-stake systems and lending pools: in the case where lending pools are believed to be more profitable, users may shift deposits away from the staking contract of the underlying consensus protocol towards lending pools, thereby endangering the security of the system.

Lending pool interest rate behaviour is examined in [47], where empirical behaviour of interest rate models in Compound [22], Aave [11] and dYdX [25] are analyzed. In particular, the authors observe a statistically significant coupling in interest rates between deployed lending pools, suggesting that the dynamic interest models are effective in discovering a global interest rate equilibrium for a given token. Our formal model is parameterized by the interest rate, that must always be positive (8): since this property holds for all interest rate functions in [47], our model can be instantiated with them.

Algorithmic stable coins MakerDAO [27] is the leading algorithmic stable coin and is credited with being one of the earliest DeFi projects. It incorporates several features found in lending pools, such as deposits, minting, and collateralization. Users are incentivized to interact with the smart contract to mint or redeem DAI tokens. This, in turn, adjusts the supply of DAI such that a stable value against the reference price (e.g USD) is maintained. Synthetic tokens are similar to algorithmic stable coins but may track an asset price such as gold or other real-world assets. Reference asset prices are determined by price oracles.

The authors of [49] introduce a taxonomy for various price stabilization mechanisms, providing insight into the functionality of such contracts. [46] uncovers a vulnerability in the governance design of MakerDAO, allowing an attacker to utilize flash loans to steal funds from the contract. The empirical performance of MakerDAO's oracles is studied in [45], which also proposes alternate price feed aggregation models to improve oracle accuracy. Finally, [42] investigates the optimal bidding strategy for collateral liquidators in MakerDAO, which is executed by through user auctions.

Stable coins which track prices of real-world currencies (e.g. USD) exhibit a price stability useful for lending pools: users with stable collateral or loan values have a lower likelihood of suddently becoming undercollateralized.

Automatic market makers Leading automatic market makers Uniswap [31] and Curve Finance [23] hold \$1.6B [30] and \$1.5B [23] worth of tokens and feature an estimated \$320M [30] and \$36M [23] worth of token exchange transactions every day. An automatic market maker (AMM) is organized in token pairs (τ, τ'), which users can interact with to exchange units of τ for τ' or vice-versa. AMM's do not match opposing actions of buyers and sellers: users simply exchange tokens with a AMM pair, where the exchange rate is determined algorithmically as a function of the AMM pair balance. Hence, the dynamic exchange rate of an AMM token pair is affected with each user interaction.

The work in [33] investigates alternative, algorithmic exchange rate models and defines the user arbitrage problem, where a profit-seeking agent must determine the optimal set of AMM pairs (with differing exchange rates) to interact with: given such arbitrage opportunities will be exploited by rational users, it is expected that exchange rates across AMM's remain consistent. AMM price models can fail: The *constant product* exchange rate model implemented by Uniswap [31] and Curve [24] is simple, but can theoretically reach a state where the the exchange rate is arbitrarily high. [54] proposes bounded exchange rate models to address this.

[32] suggests that AMM's track global average token prices effectively. As such, AMM's can inform price oracles: such oracles, however, only update price information with each new block [29] computed from time-weighted price averages of AMM pairs over the past block interval. This increases the cost of

manipulating prices of the oracle, as the manipulated price must be sustained over a period of time. We note that lending pool implementations do not rely on oracles which derive prices from AMM states.

AMM's suffer from front-running, where an attacking user observes the victim's announced, yet unconfirmed token exchange transaction, and sequences its own transaction prior to that of the victim. A front-running attack on an AMM user takes advantage of the change in exchange rate resulting from the victim's token exchange, who ends up paying a higher price, as illustrated in [55]. Front-running of smart contracts is investigated more generally in [44]: mitigations such as commit-and-reveal schemes are proposed, which come with an increased cost for user-contract interactions. In the context of AMM's, [41] introduces the notion of gas auctions, where adversarial users compete to front-run a given AMM exchange transaction by outbidding each others transaction fee.

We note that similar attacks can be modeled with an attacker that can drop or reorder transactions in our lending pool model. Such an attacker can trivially defer attempts of a borrower to repay a loan: subsequent interest accrual will eventually cause the user to become undercollateralized, so that the attacker can liquidate the victim. Such an attacker can also monopolize all liquidations for herself, preventing other users from executing such an action: [41] suggests that miners may be incentivized to perform such attacks due to gain resulting from liquidation discounts.

Margin trading An important use case of lending pools are *leveraged* long or short positions initiated by users, also referred to as margin trading. In a leveraged long position of τ against τ', the user speculates that the price of the former will increase against the price of the latter: a user borrows τ' at a lending pool against collateral deposited in τ, and then exchanges the borrowed units of τ' back to τ at a token exchange or an AMM. The user will now earn an amplified profit if the price of τ appreciates relative to τ', since both the borrowed balance and redeemable collateral in τ appreciates in value whilst only the loan repayable with τ' decreases in value. A leveraged short position simply reverses the token types. Margin trading contracts such as bZx Fulcrum [13] combine lending and AMM functionalities to offer margin trades through a single smart contract. However, since such margin trading contracts perform large token exchanges at external AMM's, attackers can use such actions to manipulate AMM prices, as shown in [51]. Furthermore, the scope of such attacks is magnified when performed with flash loans.

Flash loans Any smart contract holding balances of tokens can expose flash loan functionality to users: here, a user can borrow and return a loan within a single atomic transaction group. Informally, we describe an atomic transaction group as an a sequence of actions from a single user, which must execute to completion or not execute at all. Atomic transaction groups can be implemented in Ethereum by user-defined smart contracts [7], but can also be supported explicitly, such as in Algorand [36]. As such, flash loans are guaranteed to be repaid or not executed at all. The work in [53] introduces an initial framework to identify flash loan transactions on the Ethereum blockchain for an analysis of their intended use-cases, which include arbitrage transactions, account liquidations (in lending pools or stable coins) and attacks on smart contracts. We note that our model can be easily extended to encompass flash loan semantics.

Flash loans have been utilized in recent attacks in DeFi contracts [51] [14] [26] [28] [12]. The flashloan attack on bZx Fulcrum described in [51] involves sending the borrowed tokens to a margin trading contract, which, in turn, initiates a large token exchange at an external AMM: here, the large amount of exchanged tokens causes a significant shift in dynamic AMM exchange rate, which represents an arbitrage opportunity exploited by the attacker in several execution steps involving other contracts. Flash loans provide attackers with access to very large token values to initiate attacks.

7 Conclusions

We have provided a systematization of knowledge on lending pools and their role in DeFi, by leveraging a new model which enables formal definitions of lending pool properties, vulnerabilities, and attacks. This work represents a first step towards the rigorous analysis of DeFi contracts, improving existing literature with a precise executable semantics of interactions beween users and LPs.

Differences between our model and LP implementations We have synthesised our model from informal descriptions in the literature and actual implementation and documentation of lending pools Compound [22] and Aave [11]. To distill a usable, succinct model we have abstracted away some implementation details, that could be incorporated in the model at the cost of a more complex presentation. We discuss here some of the main abstractions we made.

The original implementations of Compound and Aave gave administrators control over the economic parameters of the LP, i.e. C_{min}, r_{liq}, and the interest rate function. This made administrators of such early versions privileged users, who could in principle prevent honest depositors, borrowers and liquidators from withdrawing funds. A Compound administrator, for example, can replace application logic which computes collateralization and authorizes supported tokens [15]. Later versions of these platforms have introduced *governance tokens* (respectively, COMP and AAVE), which are allocated to initial investors or to LP users, who earn units of such tokens upon each interaction. Governance tokens allow holders to propose, vote for, and apply changes in economic parameters, including interest rate functions. By contrast, our model assumes that economic parameters are fixed, and omits governance tokens.

In implementations, adding a new token type to the LP must be authorized by the governance mechanisms. By contrast, in our model any user can add a new token type to the LP by just performing the first deposit of tokens of that type. Implementations also allow administrators or governance to assign weights to each token type. This is intended to adjust collateralization and liquidation thresholds C_{min} and r_{liq} for the predicted price volatility of token types present in a user's loan and collateral. Further, implementations require users to pay *fees* upon actions. These fees are accumulated in a reserve controlled by the governance mechanisms of the LP, and intended to act as a buffer in case of unforeseen events. Our model does not feature token-specific weights and fees.

User liquidations in implementations are limited to repay a maximum fraction of the loan amount [6,18]. However, this implementation constraint can be bypassed by a user employing multiple accounts, so we omit it in our model.

Lending pool platforms implement the update of interest accrual in a *lazy* fashion: since smart contracts cannot trigger transactions, periodic interest accrual would rely on a trusted user to reliably perform such actions, introducing a source of corruption. Therefore, interest accrual is performed whenever a user performs an action which requires up-to-date loan amounts. Here, the interest rate in implementations is not recomputed for each time period. Instead, a single interest rate is applied to the period since the last interest accrual [9,20] in order to reduce the cost of execution, leading to inaccuracies in loan interest.

Comparison with other LP models There are few models of lending pools in the literature. The *liquidation model* of [46] is meant to simulate interactions between lending pool liquidations and token exchange markets in times of high price volatility. Unlike in our presented model, [46] performs liquidations in aggregate, and it omits individual user actions. The interest rate functions of [47] formalize various interest rate strategies used by LP implementations, and can be seen as complementary to our work. Indeed, even if we did not incorporate such functions directly in our model (for brevity), they could be easily included as instances of $I_\pi(\tau)$ in rule [Int]. The work [50] introduces a LP state model, which is instantiated with historical user transactions observable in the Compound implementation deployed on Ethereum. The model abstraction facilitates the observation of state effects of each interaction, and investigates the (historical) latency of user liquidations following the undercollateralization of borrowing accounts. Aforementioned work prioritizes high-level analysis over model fidelity: indeed, the lending pool properties and attacks we present are a direct consequence of the precision in our lending pool semantics.

Future work Our model already allows us to formally establish properties of LPs (Section 4), and to precisely describe potential attacks to LPs as sequences of user actions (Section 5). This paves the way for future automatic analyses of LPs, which could exploit e.g., model checking or automated theorem proving tools. Following the same approach we used in Section 3, we could devise formal models of other DeFi archetypes, like e.g. stable coins, automatic market makers, and flash loans. In this perspective, it would be possible to extend the scope of analysis techniques to attacks that exploit the interplay between different archetypes, which so far have been found manually by adversaries, as documented in [51]. A complementary line of research is the design of domain-specific languages for DeFi contracts, in the spirit of the works [34,37,43,52] on languages for financial derivatives. By leveraging primitives specifically tailored to DeFi, these languages could simplify the task of analysing DeFi contracts: actually, this task is overwhelmingly complex for current LP implementations, which amount to thousands of lines of Solidity code.

References

1. ERC-20 token standard (2015), https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
2. Understanding the DAO attack (June 2016), http://www.coindesk.com/understanding-dao-hack-journalists/
3. Parity Wallet security alert (July 2017), https://paritytech.io/blog/security-alert.html
4. A Postmortem on the Parity Multi-Sig library self-destruct (November 2017), https://goo.gl/Kw3gXi
5. Aave markets website (2020), https://app.aave.com/markets
6. Aave maximum liquidation amount (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolLiquidationManager.sol#L181
7. Aave v1 flashloan receiver interface (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/flashloan/interfaces/IFlashLoanReceiver.sol#L11
8. Aave v1 implementation (2020), https://github.com/aave/aave-protocol/tree/efaeed363da70c64b5272bd4b8f468063ca5c361
9. Aave v1 simplified interest (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/libraries/CoreLibrary.sol#L423
10. Aave valuation of atokens (2020), https://github.com/aave/aave-protocol/blob/efaeed363da70c64b5272bd4b8f468063ca5c361/contracts/lendingpool/LendingPoolDataProvider.sol#L114
11. Aave website (2020), https://www.aave.com
12. Akropolis Defi attack (2020), https://cryptonews.com/news/defi-akropolis-drops-20-following-a-usd-2m-heavy-hack-8299.htm
13. bzx fulcrum website (2020), https://fulcrum.trade
14. Coindesk: Value DeFi attack (2020), https://www.coindesk.com/value-defi-suffers-6m-flash-loan-attack
15. Compound comptroller setter (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L1152
16. Compound implementation (2020), https://github.com/compound-finance/compound-protocol/tree/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6
17. Compound markets website (2020), https://compound.finance/markets
18. Compound maximum liquidation amount (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L510
19. Compound oracle attack (2020), https://news.bitcoin.com/100-million-liquidated-on-defi-protocol-compound-following-oracle-exploit
20. Compound simplified interest (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/CToken.sol#L423
21. Compound valuation of ctokens (2020), https://github.com/compound-finance/compound-protocol/blob/a5591d5f9a7f6f7ad3601ec89b126a8c2af159f6/contracts/ComptrollerG5.sol#L753
22. Compound website (2020), https://www.compound.finance

23. Curve statistics (2020), https://www.curve.fi/dailystats
24. Curve website (2020), https://www.curve.fi
25. dydx website (2020), https://dydx.exchange
26. Harvest Finance flashloan attack post-mortem (2020), https://medium.com/harvest-finance/harvest-flashloan-economic-attack-post-mortem-3cf900d65217
27. Makerdao website (2020), https://https://makerdao.com
28. Origin Dollar attack (2020), https://cryptonews.com/news/4th-major-defi-hack-in-a-month-origin-dollar-loses-usd-7m-8331.htm
29. Uniswap oracle template (2020), https://github.com/Uniswap/uniswap-v2-periphery/blob/dda62473e2da448bc9cb8f4514dadda4aeede5f4/contracts/examples/ExampleOracleSimple.sol
30. Uniswap statistics (2020), https://info.uniswap.org
31. Uniswap website (2020), https://www.uniswap.org
32. Angeris, G., Chitra, T.: Improved price oracles: Constant function market makers. arXiv preprint arXiv:2003.10001 (2020), https://arxiv.org/pdf/2003.10001
33. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of uniswap markets. Cryptoeconomic Systems Journal (2019), https://ssrn.com/abstract=3602203
34. Arusoaie, A.: Certifying Findel derivatives for blockchain. CoRR **abs/2005.13602** (2020), https://arxiv.org/abs/2005.13602
35. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Principles of Security and Trust (POST). LNCS, vol. 10204, pp. 164–186. Springer (2017). https://doi.org/10.1007/978-3-662-54455-6_8
36. Bartoletti, M., Bracciali, A., Lepore, C., Scalas, A., Zunino, R.: A formal model of Algorand smart contracts. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2009.12140
37. Biryukov, A., Khovratovich, D., Tikhomirov, S.: Findel: Secure derivative contracts for Ethereum. In: Financial Cryptography Workshops. LNCS, vol. 10323, pp. 453–467. Springer (2017). https://doi.org/10.1007/978-3-319-70278-0_28
38. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
39. Chitra, T.: Competitive equilibria between staking and on-chain lending. arXiv preprint arXiv:2001.00919 (2019), https://arxiv.org/pdf/2001.00919
40. Chitra, T., Evans, A.: Why stake when you can borrow? Available at SSRN 3629988 (2020), https://arxiv.org/pdf/2006.11156
41. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symposium on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
42. Darlin, M., Papadis, N., Tassiulas, L.: Optimal Bidding Strategy for Maker Auctions. arXiv preprint arXiv:2009.07086 (2020), https://arxiv.org/pdf/2009.07086
43. Egelund-Müller, B., Elsman, M., Henglein, F., Ross, O.: Automated execution of financial contracts on blockchains. Business & Information Systems Engineering **59**(6), 457–467 (2017). https://doi.org/10.1007/s12599-017-0507-z
44. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13

45. Gu, W.C., Raghuvanshi, A., Boneh, D.: Empirical measurements on pricing oracles and decentralized governance for stablecoins. Available at SSRN 3611231 (2020), http://dx.doi.org/10.2139/ssrn.3611231
46. Gudgeon, L., Pérez, D., Harz, D., Livshits, B., Gervais, A.: The decentralized financial crisis. In: Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 1–15. IEEE (2020). https://doi.org/10.1109/CVCBT50464.2020.00005
47. Gudgeon, L., Werner, S., Perez, D., Knottenbelt, W.J.: Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In: ACM Conference on Advances in Financial Technologies. pp. 92–112 (2020). https://doi.org/10.1145/3419614.3423254
48. Kao, H.T., Chitra, T., Chiang, R., Morrow, J.: An Analysis of the Market Risk to Participants in the Compound Protocol https://scfab.github.io/2020/FAB2020_p5.pdf
49. Moin, A., Sekniqi, K., Sirer, E.G.: Sok: A classification framework for stablecoin designs. In: Financial Cryptography and Data Security. LNCS, vol. 12059, pp. 174–197. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_11
50. Perez, D., Werner, S.M., Xu, J., Livshits, B.: Liquidations: Defi on a knife-edge. In: Financial Cryptography (2021), (to appear) https://arxiv.org/abs/2009.13235
51. Qin, K., Zhou, L., Livshits, B., Gervais: Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In: Financial Cryptography (2021), (to appear) https://arxiv.org/pdf/2003.03810
52. Seijas, P.L., Thompson, S.J.: Marlowe: Financial contracts on blockchain. In: ISoLA. LNCS, vol. 11247, pp. 356–375. Springer (2018). https://doi.org/10.1007/978-3-030-03427-6_27
53. Wang, D., Wu, S., Lin, Z., Wu, L., Yuan, X., Zhou, Y., Wang, H., Ren, K.: Towards understanding flash loan and its applications in defi ecosystem. arXiv preprint arXiv:2010.12252 (2020), https://arxiv.org/pdf/2010.12252
54. Wang, Y.: Automated market makers for decentralized finance (defi). arXiv preprint arXiv:2009.01676 (2020), https://arxiv.org/pdf/2009.01676
55. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. arXiv preprint arXiv:2009.14021 (2020), https://arxiv.org/pdf/2009.14021

A Supplementary material

Proof of Lemma 1

The proof is by induction on the length of the trace from an initial state to the state $\sigma \mid \pi \mid p$. The base case is when $\sigma \mid \pi \mid p$ is an initial configuration. Then, (10) trivially holds since $\mathbb{T}_\pi$ is empty. Now assume as induction hypothesis that (10) holds for a reachable configuration Γ. We can show that the equation also holds for all configurations Γ' such that $\Gamma \xrightarrow{\ell} \Gamma'$ by considering all possible cases for ℓ. First note, that there are a number of cases where the state components involved in (10) are not affected at all. These are: $\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, Px, Int, $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$, and $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$. If ℓ is $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ or $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$ we note that the transition will increase and decrease both sides of (10) equally. Last, if ℓ is $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$ or $\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$ the sum in the lhs of (10) is kept constant (minted tokens are just transferred from one user to another one) and the rhs is not affected. $\square$

Proof of Lemma 2

We show that for a single transition $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$ the following holds:

(a) $ER_\pi(\tau) < ER_{\pi'}(\tau)$, if $\ell = \mathsf{Int}$ and $\exists \mathsf{A} : (\pi_l\ \mathsf{A})\ \tau > 0$
(b) $ER_\pi(\tau) = ER_{\pi'}(\tau)$, otherwise.

Part (a) is easy to see, since the execution of Int strictly increases the existing loans on τ which are used in the numerator of the first case in (4), without affecting the denominator.

For part (b) there are a number of cases where the state components involved in ER are not affected at all. These are: Int (if $\nexists \mathsf{A} : (\pi_l\ \mathsf{A})\ \tau > 0$), $\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, Px, and $\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$. If ℓ is $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$, $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$, or $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau_0, \tau_1)$ we note that the transition will increase and decrease the summands in the numerator of the first case in (4) equally. Last, If ℓ is $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ or $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$ we note that the transition will increase and decrease the numerator and denominator of the first case in (4) in quantities proportional to ER_π. $\square$

Proof of Lemma 3

The proof is by induction on the length of the trace $\sigma \mid \pi \mid p \rightarrow^* \sigma' \mid \pi' \mid p'$. The base case is when $\sigma \mid \pi \mid p = \sigma' \mid \pi' \mid p'$. Then the lemma trivially holds since $\mathit{sply}_{\sigma,\pi}(\tau) = \mathit{sply}_{\sigma',\pi'}(\tau)$. Now, assume as induction hypothesis that the lemma holds for all executions of length n. We show that it also holds for executions of length $n+1$. In particular we show that for a single transition $\sigma \mid \pi \mid p \xrightarrow{\ell} \sigma' \mid \pi' \mid p'$ token supplies remain constant by considering all possible cases for ℓ, where state components of Equation (11) are affected. These are: $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$, $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$, $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$. If ℓ is $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$, $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$ or $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$, changes applied to $\sigma_{\mathsf{A}}(\tau)$ and $\pi_f(\tau)$ in Equation (11) cancel out. If ℓ is $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$, the same is true for user balance and lending pool balance of token $u_\pi(\tau)$. $\square$

Proof of Lemma 4

By inspecting the formalization of the transitions it is easy to see which actions can increase or decrease in just one transition the net worth of a user on a specific token or in total, and to which extent. Indeed is is easy to see that the only actions that can modify the total net worth of a user A are Int, $\mathsf{Trf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, $\mathsf{Mtrf}_{\mathsf{A}}(\mathsf{B}, v : \tau)$, Px and $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$. Only the latter uses an action of the form $\_{\mathsf{A}}(\dots)$. The rest of the actions of the form $\_{\mathsf{A}}(\dots)$ are $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$, $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$, $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$, and $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$. Inspecting their effect on wallets and loans we can notice that they simply exchange tokens in a way that keeps the net worth constant: $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Rdm}_{\mathsf{A}}(v : \tau)$ simply swap amounts of free tokens and corresponding minted tokens proportionally to the exchange rate, while $\mathsf{Bor}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$ simply move exact amounts of free tokens between wallets and loans. $\square$

Proof of Lemma 5

The main idea is that interest accrual affects net worth by increasing the share of deposited tokens (on which there is at least one non-empty loan) and increasing loan amounts. The only actions that increase a user's deposits are $\mathsf{Dep}_{\mathsf{A}}(v : \tau)$ and $\mathsf{Liq}_{\mathsf{A}}(\mathsf{B}, v : \tau, v' : \tau')$, while the only action that decreases loans is $\mathsf{Rep}_{\mathsf{A}}(v : \tau)$. $\square$

Formal Analysis of Lending Pools in Decentralized Finance

Publication Information

Massimo Bartoletti, James Chiang, Tommi Junttila, Alberto Lluch-Lafuente, Massimiliano Mirelli, and Andrea Vandin. “Formal Analysis of Lending Pools in Decentralized Finance.” In *Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning: 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part III*, pp. 335-355. Cham: Springer Nature Switzerland, 2022.

Contribution

- Co-author.

Formal Analysis of Lending Pools in Decentralized Finance

Massimo Bartoletti[1] [0000−0003−3796−9774], James Chiang[2] [0000−0002−5126−9494], Tommi Junttila[3], Alberto Lluch Lafuente[2⋆] [0000−0001−7405−0818], Massimiliano Mirelli[2] [0000−0001−9441−173X], and Andrea Vandin[4,2] [0000−0002−2606−7241]

[1] Università degli Studi di Cagliari, Cagliari, Italy
bart@unica.it
[2] Technical University of Denmark, DTU Compute, Copenhagen, Denmark
{jchi,albl}@dtu.dk
massimilianomirelli.mm@gmail.com
[3] Aalto University, Espoo, Finland
tommi.junttila@aalto.fi
[4] Sant'Anna School of Advanced Studies, Pisa, Italy
andrea.vandin@santannapisa.it

Abstract. Decentralised Finance (DeFi) applications constitute an entire financial ecosystem deployed on blockchains. Such applications are based on complex protocols and incentive mechanisms whose financial safety is hard to determine. Besides, their adoption is rapidly growing, hence imperilling an increasingly higher amount of assets. Therefore, accurate formalisation and verification of DeFi applications is essential to assess their safety. We have developed a tool for the formal analysis of one of the most widespread DeFi applications: Lending Pools (LP). This was achieved by leveraging an existing formal model for LPs, the Maude verification environment and the MultiVeStA statistical analyser. The tool supports several analyses including reachability analysis, LTL model checking and statistical model checking. In this paper we show how the tool can be used to analyse several parameters of LPs that are fundamental to assess and predict their behaviour. In particular, we use statistical analysis to search for threshold and reward parameters that minimize the risk of unrecoverable loans.

1 Introduction

Financial trading has recently shifted to virtual markets, platforms entirely regulated and controlled by novel protocols. *Decentralised Finance* (DeFi) [34] applications are deployed on blockchains like Ethereum [34,12], which offer distributed infrastructures to execute *smart contracts* [18] without intermediaries. DeFi has recently been employed by a growing community of users. As of April 2022, the growth of the capital locked by DeFi applications has increased almost

⋆ Corresponding author.

10 times in the last two years: from approximately \$9.78bn, on 1 April 2020, to over \$83.51bn, on 1 April 2022 [29]. Even assuming the security guarantees ensured by the underlying blockchain, DeFi smart contracts have several vulnerabilities latent in their design [30,36]. Given the considerable amount of funds daily exchanged on DeFi platforms [1,16], even minor design flaws could determine massive and intolerable losses [21]. Notwithstanding the increasing interest of several research groups in this area [9,5,2,32,4,19], the complexity of DeFi protocols yields new interesting research problems. Formal verification of these systems is crucial, in order to ensure their correctness and security.

The verification tool proposed in this paper simulates and analyses *Lending Pools* (LPs), one of the most popular DeFi applications, whose two main features are lending and borrowing assets, to support various financial practices, including margin trading. Our verification tool is based on the formal model of LPs proposed in [6]. Such model encompasses the behaviour of the most widespread LPs, namely Aave [10] and Compound [24].

We craft an operational specification of the LP model of [6] in Maude [15], a specification language which is particularly suitable for highly concurrent systems such as LPs. Additionally, Maude provides a very extensive environment for both simulating and verifying the properties of the specified models. Given the complexity of the modelled systems, the analyses techniques offered by the Maude environment are not sufficient. Specifically, since the system may evolve by following an infinite number of execution paths, the traditional model checking methods result in being either ineffective or unviable. Therefore, the Maude-based LP simulator has been extended to support statistical analyses. This has been achieved by integrating the simulator with the MultiVeStA statistical analyser proposed in [31] and recently redesigned to focus on analyses of interest for of economical agent-based models [33]. The tool offers analysis techniques from the family of statistical model checking [3]. These statistical analyses, despite producing less accurate results, allow to observe the quantitative behaviour of large instances of the model, offering statistically-reliable results. In the case of lending pools, this approach allows to estimate parameters of the model so to increase its safety. Specifically, an essential safety property of the model is that the value of non-repayable loans is low.

This paper is based on the work done in [25] and proposes a Maude-based LP simulator (Section 3) capable of conducting several analyses of lending pools including LTL model checking and statistical analysis. The tool is open source and available at [26]. Additionally, the study showcases the usage of the tool by answering a still non-investigated research question, aiming at an enhancement of the analysed platforms' safety. In particular, the statistical analysis presented in Section 4 shows that a choice of the parameters used to instantiate the LP model reduces the amount of non-repayable loans.

2 Lending Pools and Price Models

Lending Pools. *Lending Pools* [35] are a class of DeFi applications which allow users to lend and borrow cryptoassets. At the time of writing, LPs are the most used DeFi applications, with the majority of them being deployed on Ethereum [29]. Deposited funds are pooled and lent on-demand to borrowers, only if they possess enough collateral (i.e. only if their account is overcollateralized). As blockchains typically do not provide strong identities, but pseudonyms [12], users' actions are difficult to be regulated under a jurisdiction, which makes collateralization the main protection mechanism against adversarial behaviours [27]: an agent can only borrow a quantity of tokens worth less than the amount of collateral they deposited. This mechanism and others (e.g. interest rates) is in place in order to incentivize borrowers to repay their loans.

We now recall details of the lending pools model in [6]. The basic components of the model are *agents* and *cryptoassets*. LP agents are the rational entities taking part in the protocol. Contrarily, LP cryptoassets are token types, each representing a different virtual currency. The model distinguishes two classes of token types: *free tokens* and *minted tokens*, denoted respectively by the sets $\mathcal{T}_f = \{\tau_i\}_{i\in[1..k]}$ and $\mathcal{T}_m = \{\tau_i'\}_{i\in[1..k]}$, where k is the number of cryptocurrencies available on the pool. The difference between these classes of token types is that free tokens have a value established by external markets, whereas minted tokens are assets coined by the protocol, hence holding value only in a specific LP environment. In other words, minted tokens are loyalty credits held by the agents actively joining the protocol. In fact, minted tokens are granted by the protocol to the agents in return for free tokens, hence each minted token τ' corresponds to a free token τ, also called its underlying token. We denote by $\mathcal{T}$ the set of all token types, i.e. $\mathcal{T} = \mathcal{T}_f \cup \mathcal{T}_m$.

Given agents and assets, the LP model yields as a transition system where each state Γ is of the form $\Gamma = \sigma \mid \pi \mid p$:

1. The *wallets* function $\sigma : \mathcal{A} \to (\mathcal{T} \to \mathbb{R}_0^+)$ stores each agent's balance of tokens. For instance, the wallet of an agent A is expressed by the partial function σ_A, and the balance of its τ-typed tokens by $\sigma_A(\tau)$.
2. The *pool* component π is a triple (π_f, π_l, π_m). It is composed by three partial functions: $\pi_f : \mathcal{T}_f \to \mathbb{R}_0^+$ storing the amount of free tokens deposited in the pool, $\pi_l : \mathcal{A} \to (\mathcal{T}_f \to \mathbb{R}_0^+)$ memorising the loans each agent owes to the pool and $\pi_m : \mathcal{T}_f \to (\mathcal{T}_m \times \mathbb{R}_0^+)$ keeping track of the minted tokens (also called the *collateral* or credits) purchased from the pool.
3. The price function $p : dom(\pi_f) \to \mathbb{R}_0^+$ stores the price of each free token available in the pool.

Given a partial map f, we denote by $f\{v/x\}$ the point-wise update of f at the point x to the value v. In order to add and remove tokens in the functions defined above, a partial binary operation $\circ : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+$, such as addition, is extended to them. Given a partial map $f : \mathcal{T} \to \mathbb{R}_0^+$, a token type $\tau \in \mathcal{T}$ and a value $v \in \mathbb{R}_0^+$, the partial map $f \circ v : \tau$ is defined as

$$f \circ v : \tau = \begin{cases} f\{f(\tau) \circ v/\tau\} \text{ if } \tau \in \mathit{dom}(f) \text{ and } f(\tau) \circ v \text{ is defined} \\ f\{v/\tau\} \text{ if } \tau \notin \mathrm{dom}(f) \end{cases}$$

In order to describe the model evolution, some additional definitions shall be given. The following LP components may rely on the whole state Γ, or some of its components. This dependency is indicated by the means of subscripts. For instance, writing F_X means that F depends on the X component of the state.

The functions V_Γ^l and V_Γ^m define, respectively, value of tokens lent to a given agent, and the value of minted tokens owned by a given agent:

$$V_\Gamma^l(A) = \sum_{\tau \in \mathcal{T}_f} (\pi_l(A))(\tau) \cdot p(\tau) \qquad V_\Gamma^m(A) = \sum_{\tau' \in \mathcal{T}_m} \sigma_A(\tau') \cdot \mathit{ER}_\pi(\tau', \tau) \cdot p(\tau)$$

where $\mathit{ER}_\pi(\tau', \tau)$ is the exchange rate of minted tokens (see Section 3.1 of [6]).

The collateralization of an agent A is defined as $C_\Gamma(A) = {V_\Gamma^m(A)}/{V_\Gamma^l(A)}$. This is an essential indicator of agents' lending safety: in fact, a collateralization below a given threshold ($C_{\min}$) entails an agent to be liquidated and hence to incur in a financial loss, as detailed later.

The behaviour of agents interacting with a lending pool is formalized as a set of rewriting rules, which define transitions between states. Such transitions are written as $\Gamma \xrightarrow{\mathrm{r}_A(z^n)} \Gamma'$, where Γ is the starting state, Γ' is the target state, and $\mathrm{r}_A(z^n)$ is the action (fired by A) which triggers the state transition. Actions have the form $\mathrm{r}_A(z^n)$, where r is the action name, and z^n is an n-tuple of parameters.

The main actions of lending pools are informally summarised in Table 1. Since the focus in this paper is on liquidations as one of the key incentive mechanisms, we will provide the details for such action only. Figure 1 provides a formal description of the rule. The essential preconditions to understand the rule are ④, ⑧, ⑨, ⑩ and ⑪.

$$\frac{\begin{array}{lll} \textcircled{1}\ \tau' \in \mathcal{T}_m & \textcircled{2}\ \sigma_A(\hat\tau) \geq v & \textcircled{3}\ \pi_l(B)(\hat\tau) \cdot \mathit{Maxliq} \geq v \\ \textcircled{4}\ v' = v \cdot \frac{p(\hat\tau)}{p(\tau)} \cdot r_{\text{liq}} & \textcircled{5}\ \sigma_B(\tau') \geq v' & \textcircled{6}\ \pi'_f = \pi_f + v : \hat\tau \\ \textcircled{7}\ \pi'_l = \pi_l\{\pi_l(B) - v : \hat\tau / B\} & \textcircled{8}\ \sigma'_A = \sigma_A - v : \hat\tau + v' : \tau' & \textcircled{9}\ \sigma'_B = \sigma_B - v' : \tau' \\ \textcircled{10}\ C_{\sigma|\pi|p}(B) < C_{\min} & \textcircled{11}\ C_{\sigma'|\pi'|p}(B) \leq C_{\min} & \end{array}}{\sigma \mid \pi \mid p \xrightarrow{\mathsf{Liq}_A(B, v:\hat\tau, \tau')} \sigma\{\sigma'_A/A\}\{\sigma'_B/B\} \mid (\pi'_f, \pi'_l, \pi_m) \mid p}$$

Fig. 1: The rule for liquidation.

③ The amount of repayable loan is limited by a percentage factor *Maxliq*, as done in Aave [11] and Compound [28].

$\text{Dep}_A(v : \tau)$	A deposits v free-tokens of type τ from its wallet to the pool. Subsequently, the pool coins v' units of τ', with v' computed so to incentivize deposits only if the LP is lacking free tokens.
$\text{Rdm}_A(v : \tau')$	A redeems v units of the minted token τ', as long as A's collateralization is greater than a threshold (C_{min}) and LP holds enough tokens of type τ'.
$\text{Bor}_A(v : \tau)$	A borrows v units of a free token τ, assuming it has enough collateral.
$\text{Rep}_A(v : \tau)$	A repays v units of its loan in the free token τ to the LP.
$\text{Liq}_A(B, v : \hat{\tau}, \tau')$	A (liquidator) liquidates a variable amount of B's (borrower's) minted tokens τ', by paying v units of free tokens $\hat{\tau}$. Notably, $\hat{\tau} \in \mathcal{T}_f$ is in general different from τ, the underlying token of $\tau' \in \mathcal{T}_m$. This action can be executed only if the B's collateralization is below C_{min}, meaning B is undercollaterized.
Int	The LP contract accrues interest on the existing loans. This disincentivizes borrowers from postponing their loans repayment.
Price	Token prices are updated according to a given price evolution model.

Table 1: Summary of some of the lending pools actions from [6].

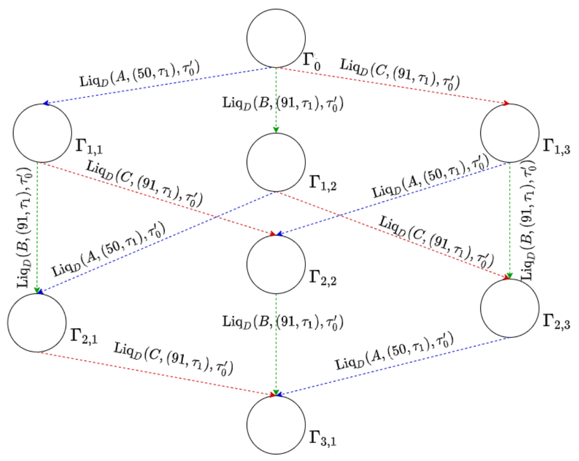
(4) computes the reward for the liquidating agent. This is based on the liquidated amount v and the reward factor r_{liq}. The idea is that A, by repaying part of B's loan, is reducing the likelihood of the protocol to become illiquid. This behaviour is incentivized by the platform by setting the aforementioned reward to a value strictly higher than 1. A common value for r_{liq} is 1.1.

(8), (9) update the involved agents' wallets, A repays v units of B's loan in $\hat{\tau}$ and is compensated with v' units of τ'

(10) ensures that the rule is executable only if B's collateralization is less than C_{min}, which is often set to 1.5. This rule is the reason why agents' collateralization should be at least C_{min}, so to avert the risk of being liquidated and incurring in the loss of the liquidation reward r_{liq}.

(11) prevents A from seizing a higher collateral amount than the one required for B to be considered safe (i.e. $C_{\Gamma}(B) \geq C_{\text{min}}$).

Figure 2 illustrates the transition system for a simple running example, where three liquidate actions are executed. The figure shows six possible traces all originating from Γ_0 and having $\Gamma_{3,1}$ as final state. Each state in the figure is defined by a row in Table 2. Additionally, transitions, namely Liq actions performed by D, are indicated by different colours depending on the liquidated borrower in both the transition system and the table. Notably, assuming $C_{\text{min}} = 1.5$ and $r_{\text{liq}} = 1.1$, all borrowers in Γ_0, A, B and C, are undercollaterized. Specifically, A is marginally undercollaterized since $C_{\Gamma_0}(A) = 1.25 > 1.1 = r_{\text{liq}}$, while B and C are strongly undercollaterized, being both $C_{\Gamma_0}(B)$ and $C_{\Gamma_0}(C)$ below 1.1. This allows D to seize the entire B and $C's$ collateral, as evident from $\Gamma_{3,1}$ in Table 2. Contrarily A's collateralization is restored to C_{min}.


Γ_0
$\mathrm{Liq}_D(A,(50,\tau_1),\tau_0')$
$\mathrm{Liq}_D(B,(91,\tau_1),\tau_0')$
$\mathrm{Liq}_D(C,(91,\tau_1),\tau_0')$
$\Gamma_{1,1}$
$\Gamma_{1,2}$
$\Gamma_{1,3}$
$\Gamma_{2,1}$
$\Gamma_{2,2}$
$\Gamma_{2,3}$
$\Gamma_{3,1}$
Fig. 2: Example transition system.

As an example, consider transition $\Gamma_0 \xrightarrow{\mathrm{Liq}_D(B,91:\tau_1,\tau_0')} \Gamma_{1,2}$. Agent D repays 91 units of τ_1, seizing $91 \cdot r_{\mathrm{liq}} \approx 100$ units of τ_0' from agent B. This also affects π, in a way that the funds in τ_1 are incremented by 91 units, as illustrated by $\pi_f(\tau_1)$, while B's loan is decremented by 91 units, as shown by $\pi_l(B)(\tau_1)$. Contrarily, π_m is not modified by the transaction, as the 100 units of minted tokens τ_0' are simply transferred from B's wallet to D's one.

Stock Market Price Modelling We use the *geometric Brownian motion* (GBM) to define a predictive model for price evolution based on past stock market trends. A GBM is a continuous-time stochastic process $P_t = P_0 \cdot exp\left[\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right]$. The two constants μ and σ are respectively called *drift* and *volatility*, whereas W_t is a random variable following a *Weiner process*, i.e. a process $W_t = \epsilon\sqrt{dt}$ satisfying the following properties: (i) $\epsilon \sim N(0,1)$ and (ii) for any given pair (t_0, t_0'), W_{t_0} and $W_{t_0'}$ are independent. In other words, a W_t is the component yielding the stochastic behaviour of a GBM. The geometric Brownian motion as a whole can be viewed as the harmonic result of its two components [20]: (i) the drift $\left(\mu - \frac{\sigma^2}{2}\right)t$ and (ii) the volatility σW_t. The effects of the two components on the resulting

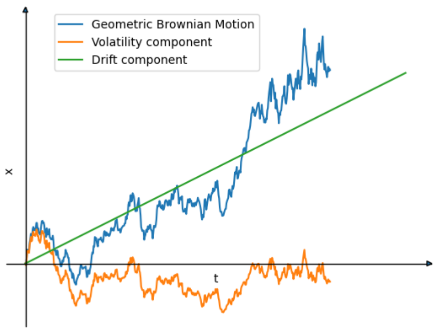


Fig. 3: GBM components.

Γ	π_f	π_l			σ_A		σ_B		σ_C		σ_D			C_Γ		
		A	B	C												
	τ_1	τ_1	τ_1	τ_1	τ_1	τ_0'	τ_1	τ_0'	τ_1	τ_0'	τ_1	τ_0'	τ_1'	A	B	C
Γ_0^i	195	80	100	125	80	100	100	100	125	100	500	0	500	1.25	1	0.8
$\Gamma_{1,1}$	**245**	**30**	100	125	80	**45**	100	100	125	100	**450**	**55**	500	**1.5**	1	0.8
$\Gamma_{1,2}$	**286**	80	**9**	125	80	100	100	**0**	125	100	**410**	**100**	500	1.25	**0**	0.8
$\Gamma_{1,3}$	**286**	80	100	**34**	80	100	100	100	125	**0**	**410**	**100**	500	1.25	1	**0**
$\Gamma_{2,1}$	**336**	**30**	**9**	125	80	**45**	100	**0**	125	100	**359**	**155**	500	**1.5**	**0**	0.8
$\Gamma_{2,2}$	**336**	**30**	100	**34**	80	**45**	100	100	125	**0**	**359**	**155**	500	**1.5**	1	**0**
$\Gamma_{2,3}$	**377**	80	**9**	**34**	80	100	100	**0**	125	**0**	**318**	**200**	500	1.25	**0**	**0**
$\Gamma_{3,1}$	**427**	**30**	**9**	**34**	80	**45**	100	**0**	125	**0**	**268**	**255**	500	**1.5**	**0**	**0**

Table 2: States of the transition system in Figure 2. For simplicity, the price function p is assumed to be constant such that $p(\tau_0) = p(\tau_1) = 1$ in every state. The values of the LP parameters are $C_{\min} = 1.5$, $r_{\text{liq}} = 1.1$ and $\mathit{Maxliq} = 1$.

process is shown in Figure 3. The drift component defines the trend of the resulting process, whereas the volatility component is a measure of the randomly sampled shocks. Intuitively, this signifies that negative values for μ yield to a downward prediction trend, whereas positive ones to a growth. Oppositely, the higher the σ is, the more significantly the prices predictions change. Ususall, μ and σ are estimated based on the daily log returns of the targeted stock market [17,20]. Given the closing prices of two consecutive trading days C_1 and C_2, the log return w.r.t. the second trading day is defined as $ln(C_2) - ln(C_1)$.

3 An LP Simulator for Liquidating Agents

We now lay the foundations for tackling a significant research problem for LPs: finding optimal $C_{\min}$ and r_{liq} parameters. This is achieved by instantiating the LP simulator to conduct statistical analyses of the model. The simulator comprises: the Maude specification of LPs [26]; a strategy for automating the behaviour of rational liquidators (Section 3.1); and a price evolution model for the three most widely employed cryptocurrencies (Section 3.2).

3.1 A Fully-automated Liquidating Strategy

This section introduces a liquidating strategy causing the LP protocol to possibly reach unsafe states, where loans are not guaranteed to be repaid. We first give an intuitive understanding of aggressive liquidating behaviours, and then describe the proposed liquidating strategy.

The impact of liquidations on collateralization Liquidate actions involve two agents: a liquidator, i.e. as an agent with enough tokens to fire liquidate actions, and a borrower with a collateralization below the threshold $C_{\min}$.

Liquidators have a fundamental role in the financial safety of LPs, as they supply free tokens whenever the pool is lacking them. On the other hand, excessively zealous liquidators could be harmful to the system, since they could disincentivize undercollaterized borrowers to repay their loans. This is exemplified in Figure 4, where all the liquidating scenarios are outlined. The figure illustrates the agents' collateralization, detailing the outcomes of liquidate actions in every possible (non-trivial) state. The scenarios are also captured by the running example in Figure 2.

Firstly, the three dashed lines in the figure correspond to the liquidation parameters specific to the instantiated pool. Their labels represent the respective line slopes. The line labelled 1 depicts the scenarios where the collateral value equals the loan value. Consequently, it can be intended as the loan repayment incentivizing threshold, i.e. the collateralization value below which borrowing agents should be considered to be disincentivized in repaying their loans as their outstanding loan debt exceeds their collateral in value. These residual loans are also called *non-recoverable*.

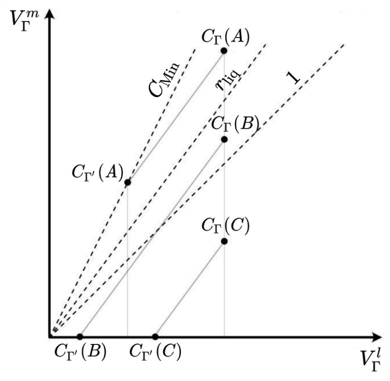


Fig. 4: Liquidation scenarios

Additionally, the three points indicate the initial collateralization of three liquidated borrowers. Each liquidation action is illustrated by a solid line drawn from $C_\Gamma(I)$ to $C_{\Gamma'}(I)$ for $I \in \{A, B, C\}$. Liquidations entail a decrease in the liquidated user's collateralization by a linear factor proportional to r_{liq} and ultimately determined by the liquidator. Note that the liquidation actions described in the figure follow the semantics of the liquidate action, as the resulting loan value must be greater than zero and the final collateralization must be at most C_{min}.

It is worth observing that the liquidations in the figure can be achieved by applying only one action if and only if two conditions hold. Firstly, the liquidator invests enough liquidity to seize the entire seizable collateral. Secondly, the liquidated borrower does not diversify the type of the loan. If either the first condition or the second does not hold, then the liquidations illustrated in the figure can be achieved uniquely by performing several liquidate actions on the borrower. This is frequently the case in the major LP implementations (Compound and Aave). In fact, these prevent the whole seizable collateral amount to be atomically liquidated, by setting *Maxliq* which is variable in Compound [28] and constant (equals to 0.5) in Aave [11]. Our model includes the parameter *Maxliq* as a constant following Aave, but it could be extended to a variable one.

The proposed liquidating strategy As shown in Figure 4, the collateralization of A is re-established, whereas liquidations cause B and C to lose their entire collateral, disincentivizing them from repaying the loans. In light of this fact, a relevant research question is whether there exists an optimal pair $(C_{\min}, r_{\text{liq}})$ such that the number of non-recoverable loans is minimal.

It is worth to observe that, ideally, the closer r_{liq} is to 1, the more the collateralization of a loan can drop and still be recoverable by liquidation. Thus a r_{liq} marginally greater than 1 is optimal in our model, since it would lead to the strongest recovery of user collateralization during liquidation. However, actual platforms deviate from such ideally optimal r_{liq} as the costs incurred by liquidators to execute actions have to be compensated by a suitable discount r_{liq} on the purchase of minted tokens from the liquidated borrowers. In order to investigate the effects of choices r_{liq} and $C_{\min}$, we propose a strategy attempting to reproduce a rational behaviour for liquidators. The employed strategy simulates a *rational* behaviour where liquidators repay the entire borrowers' loan. The rationality of the behaviour we are going to study is based on the following key observations:

1. Fast liquidations have the advantage of restoring liquidity whenever the borrowers have collateralization slightly above r_{liq} (see agent A in Figure 4).
2. On the other hand, fast liquidations may generate non-recoverable loans whenever the borrowers have collateralization slightly below r_{liq} (as for agents B and C in Figure 4).
3. Price fluctuations can change the scenario between (1) and (2). For example, it could raise the collateralization of borrowers to r_{liq} allowing the liquidators to effectively restore the agents' collateralization to $C_{\min}$, so that it would be convenient to delay liquidations.

The strategy used to implement the liquidator behaviour selects the liquidate input parameters, so to maximise the value of seized collateral. Specifically, given a liquidator L, the strategy computes the remaining four parameters of Liq: the borrower's agent identifier (B_r), the amount of loan to be repaid (v_r), the type of the asset to be repaid $(\hat{\tau}_r)$ and the one of the asset to be seized (τ'_r). Because of space constraints, we refer to [25] for a detailed account of the strategy.

3.2 Price Modelling

This section describes the price model employed to predict cryptocurrencies prices, based on historical data. We start with an overview of the price model to motivate its adoption. Afterwards, we present the three model instantiation scenarios used in the subsequent statistical analysis.

Predicting cryptocurrency prices The cryptoassets prices are derived from a statistical model representative of the past price behaviour based on the GBM. A GBM is instantiated by two parameters drift and volatility which can be estimated from the currency historical data. This makes the GBM the ideal stochastic process for modelling stock prices based on their past evolution [17].

Aiming at stress-testing the LP protocol and inspired by [19], we have designed three different scenarios, each comprising a pair of price trends. In practice, each scenario simulates the evolution of prices of a given collateral and loan assets, in a way that respectively when the former declines, the latter increases. In fact, assuming that each borrower B_0 owes a loan in only one asset type τ_l and similarly holds collateral of only one asset type τ_m, such a model for prices necessarily causes some borrowers to become undercollaterized, as shown in (1).

$$C_\Gamma(B_0) = \frac{V^m_\Gamma(B_0)}{V^l_\Gamma(B_0)} \xrightarrow{p(\tau_m)\to 0 \;\; p(\tau_l)\to V} 0, \text{with } V \gg 0 \tag{1}$$

More precisely, prices modelling is achieved by opportunely gathering the data used to estimate the parameters (drift and volatility) for generating a growing, decreasing or relatively constant GBM process. In the literature, daily closing prices of stock markets are used since their samples generally tend to be normal, which allows to employ the GBM generic formula. Ultimately, since prices' predictions pairs should variate in a way that they simultaneously display an opposite behaviour, it is necessary to correlate them, as shown in [20].

Prices model instantiation Given a collateral asset τ_m and a loan asset τ_l, the three prices evolution pairs are shown in Table 3.

Scenario	τ_m	τ_l	$p(\tau_m)$	$p(\tau_l)$
ETH-WBTC	ETH	WBTC	Declining	Increasing
ETH-USDC	ETH	USDC	Declining	Constant
USDC-WBTC	USDC	WBTC	Constant	Increasing

Table 3: The three implemented prices evolution scenarios

The choice of the cryptocurrencies in the table is motivated by their closing price historical evolution in three different trimesters (shown in the Appendix, Figure 9). By using those samples, it is possible to simulate the desired trends indicated in the columns named $p(\tau_m)$ and $p(\tau_l)$. This is achieved by estimating the expected price returns (μ) and the price volatility (σ), which are utilised as the drift and volatility instantiating the resulting GBM. The two parameters are estimated according to [20]. The drift μ is simply obtained by computing the mean over the closing prices. Contrarily, σ is calculated as $\sigma = \frac{s}{\sqrt{T}}$, where $T = \frac{91}{365}$, s indicates the standard deviation of the log returns and $\sqrt{T}$ is the annualisation constant.

The selected sampling time span (91 days, i.e. a trimester) is motivated by the fact that cryptoassets are subject to sudden fluctuations and, even though short samples might not be representative of the entire population, this is a consolidated practice [20]. Besides, the resulting price predictions span over the same time frames, as each price model instantiation produces 91 prices predictions, as illustrated in Section 3.2. Notably, the selected cryptocurrencies (ETH, USDC and WBTC) were among the four-most-utilised assets on the Compound market [16] at the moment of writing. Lastly, the selected closing price samples are suitable, since the derived log returns distributions tend to be normal.

Figure 5 shows an estimation of the GBM parameters (obtained from the close prices in the Appendix, Figure 9), by the previously discussed methodology. The parameters are then utilised to instantiate the six GBM processes (each for price evolution), simulating the scenarios in Table 3. Finally, the asset initial price P_0 is a constant set to the actual price in USD of each asset on May 5th, 2021.

Currency	μ	σ	P_0 (usd)
ETH	-0.012	0.12	3269.08
USDC	-7.84E-5	0.005	0.99
WBTC	0.012	0.094	57260.0

Fig. 5: GBM parameters

Expected price predictions We have used the MultiVeStA statistical analyzer to examine the prices predictions generated by the GBM in each of the scenarios explained in Section 3.2. The details are provided in the appendix (Figure 10), and show the normalised trend of the price scenarios, discussed in Section 3.2. The figures in the appendix show that the expected behaviour, expressed in Table 3 is obtained in all the considered scenarios. Additionally, in Figures 10a and 10c prices predictions are strongly correlated as it is expected. In fact, the GBMs pairs were instantiated as negatively correlated processes accordingly to [20], Section 14.5. Contrarily, Figure 10b shows less correlated prices predictions. This is probably due to the fact that the computation was bounded to execute maximum $5,010$ simulations. In fact, from experimental evidence, the approximation seem to converge at a very slow speed.

4 Statistical Analysis of Liquidation Scenarios

We have experimented with the LP model simulator described in Section 3 in order to answer the question: *given a specific scenario, what is the impact of the pair of LP parameters $C_{\min}$ and r_{liq}?*

We have considered scenarios generated by four factors. First, the liquidator logic defined in Section 3.1, determines immediate and quick liquidations, causing a significant financial loss to the liquidated party. Secondly, the agent to be liquidated is selected so to maximise the value of seized collateral, which is the most beneficial and rational option for liquidators. Thirdly, liquidators are assumed to hold an *unbounded* amount of resources, which allows them to repeat liquidations as long as there exists an undercollaterized agent. Finally, cryptoasset prices evolve following a trend aimed at causing borrowers to suddenly become undercollaterized.

We recall that the effect of the pair $C_{\min}$ and r_{liq} we are looking for is one that minimises the number of undercollaterized borrowers. We have explored the space of choices for the pair by executing MultiVeStA experiments for all $C_{\min}$ ranging, with step 0.1, from 1.2 to 1.5 and r_{liq} ranging from

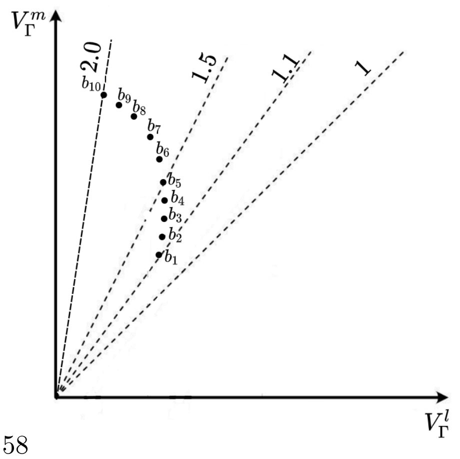


Fig. 6: Distribution of collateralization in initial configurations.

1.1 to $C_{\text{min}} - 0.1$. These ranges were selected based on the values typically assigned to these parameters in the actual implementations: $C_{\text{min}} = 1.5$ and $r_{\text{liq}} = 1.1$ [6].

On these premises, we first illustrate the LP model initial configurations used for the subsequent experimentation. Next, we present the results of the performed experiments.

Initial configurations The initial configurations were designed to test the resistance of different borrowers' collateralization to becoming unrecoverable, when subject to repeated liquidations. Since the intention is to observe the model behaviour under three price models (Section 3.2), three different initial configurations are produced, each having a different price for collateral and loan assets. All the configurations share the same amount and types of agents. Specifically, a generic initial configuration comprises ten borrowers having collateralization ranging from 1.0 to 2.0, with step 0.1. This is depicted in Figure 6, where b_i represents the generic borrower B_i's collateralization ($C_{\Gamma^i}(B_i)$), for Γ^i initial configuration. Additionally, an arbitrary number of liquidators (three) are added to each configuration.

Experimental results The results discussed here were obtained by performing MultiVeStA experiments of the LP simulator. Specifically, the inputs to the tool are: the LP simulator discussed in Section 3, a MultiQuaTEx property to express the desired measure to be estimated (the expected collateralization value at each liquidation round for each borrower), and a pair of statistical parameters defining the confidence interval (CI) of interest: the maximum confidence interval width δ and the confidence level $\alpha = 0.05$ which provides 95% statistical confidence that the estimated value is in the confidence interval. For each property, MultiVeStA will generate enough simulation to meet the required CI.

Figure 7 shows the per-borrower collateralization for varying liquidation rounds and choices of C_{min} and r_{liq} in the eth-wbtc prices scenario, with a fixed $r_{\text{liq}} = 1.1$. In this scenario, one can see that undercollaterized agents have a very different behaviour than overcollaterized ones. Specifically, the undercollaterized agents undergo very serious liquidations, which often lead them to unrecoverability, as their collateralization converges to a constant below C_{min}. Contrarily, overcollaterized agents do not incur in severe financial losses.

Additionally, our experiments (presented in detail in the Appendix, Figures 11a to 11c) show that the C_{min} and r_{liq} having the least negative effects on undercollaterized balances is $C_{\text{min}} = 1.5$, $r_{\text{liq}} = 1.1$. This is also quantitatively confirmed by the numbers in Figure 8. Intuitively, this is a consequence of the fact that when $C_{\text{min}} = 1.5$ and $r_{\text{liq}} = 1.1$ the collateralization of each agent b_1 to b_5 is higher on average than for any other C_{min} and r_{liq} pairs. As a result, the

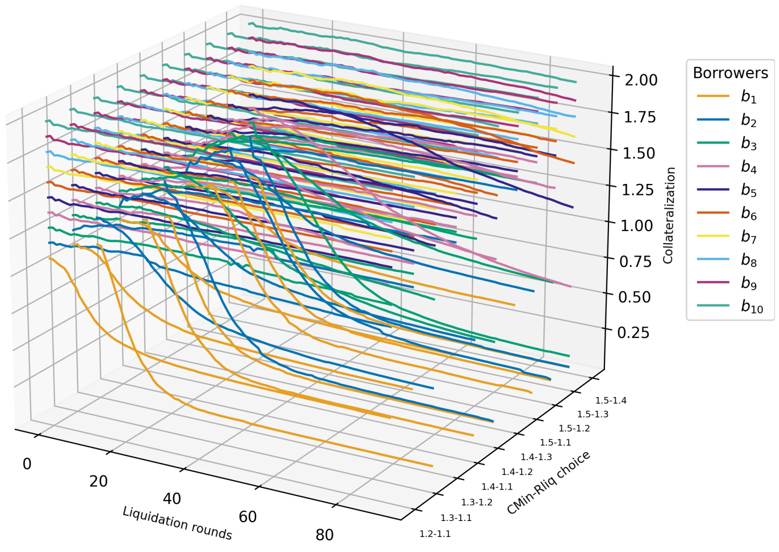


Fig. 7: Per-borrower collateralization (b_1 to b_{10}) in the ETH-WBTC prices scenario, for varying liquidation rounds and `CMin-Rliq` choices.

number of unrecoverable loans, the ones held by agents whose collateralization is below 1, is minimised.

Finally, our experiments (presented in detail in the Appendix, Figures 12a to 12c) show that overcollaterized borrowers could still incur in liquidations, in case the prices abruptly change as in the prices scenario ETH-WBTC. Differently, in the other scenarios, employing the stable coin USDC, overcollaterized agents are, on average, rarely liquidated.

Price scenario	(`CMin-Rliq`)		
	(1.5-1.1)	(1.4-1.1)	(1.3-1.1)
ETH-WBTC	0.7115	0.6518	0.6137
ETH-USDC	0.7106	0.6583	0.6231
USDC-WBTC	0.8381	0.7739	0.7299

Fig. 8: Minimum average $C_{\Gamma}(B_1)$

5 Related Works

Verification of DeFi applications is a fairly recent research area where several techniques have been applied. We focus our discussion on works devoted to formal modelling and reasoning of DeFi applications, which typically follow two parallel directions: verification of the model properties [9,5,2,32], and statistical analysis of the model variables [4,14,22,13,19].

The work in [9] is one of the first addressing formal verification of smart contract properties. Their study combines a game-theoretic approach with probabilistic model checking, ultimately validating their results with the model checker PRISM [23]. Another example of research in this direction is Tolmach et al. [32]

which developed the first multi-pools model and verified invariant properties initially formulated by [8]. Finally, [2] proposed a very relevant study on smart contracts, by modelling not only the contracts and the agents' behaviour but also the underlying blockchain using the BIP framework [7] and statistical model checking (as we do). The work in [2] employs statistical methods too. However, in their case, statistics is useful to estimate unknown variables of the analysed model, hence deriving desirable properties. The quantitative variables estimation is also achieved by performing Monte-Carlo simulations, with a more closely look at the behaviours displayed by agents [14]. In fact, most of this research in this line [22,13,4] bases its results on Agent-Based Simulations, which is employed to stress test the actual smart contracts implementations being executed on a *"custom-built Ethereum virtual machine that is written in C++"* [22]. This research direction, although suggesting promising results, is not ultimately supported by strong statistical guarantees, since the number of Monte Carlo simulations performed to run their analyses is arbitrarily chosen and not backed by a formal justification [22,19]. Nonetheless, a work relevant to ours is the analysis conducted in [22] on the Compound protocol scalability in face of high stock market prices volatility. Similarly to our work, their analysis models the prices by the use of the GBM. However, their data collection and analysis methodologies are very different. In fact, they do not sample entire historical periods as illustrated in Section 3.2 for estimating prices volatility. Contrarily, they simply evaluate the minimum and maximum volatility values ever observed and instantiate the GBM for different prices volatilities so to simulate several market environments. Finally, the prices model in Section 3.2 has been mostly inspired by [19]. Similarly to [22], they stress-test an LP model, not a specific implementation, by using the same price model explained in Section 3.2. Nonetheless, a remarkable difference is that they instantiate the predictions of the collateral and loan assets pairs with three different correlation parameters. We assume instead predictions of prices pairs to be strongly negatively correlated ($\rho = -1$), in order to simulate the worst-case scenario. Additionally, we reproduced [19]'s environment by using historical data of three different real cryptoassets on the market.

6 Conclusions

We have presented a tool for the analysis of lending pools, an archetypal DeFi application. Overall the tool consists of (i) an accurate LP simulator based on the model of [6] which can support both the study of vulnerabilities and attacks of LPs; (ii) a model checker capable of doing simple reachability analysis and verifying whether LTL properties hold of specific configurations; and (iii) a tool for statistical analysis backed by the statistical model checker MultiVeStA. In this paper, we have focused on (iii) and we have shown how to use it to optimize the LP parameters under specific scenarios. Details on (i) and (ii) as well as further examples, including reproduction of price oracle attacks using reachability analysis and LTL model checking are available in [25].

Future research supported by the developed tool could include the formalization of further attacks and properties of the model. Specifically, one could study resistance to illiquidity, as suggested by [22], or the behaviour of multi-pools configurations, each offering different market opportunities to agents, as proposed by [35] and partially developed in [32].

Acknowledgements Massimo Bartoletti is partially supported by Conv. Fondazione di Sardegna & Atenei Sardi project F75F21001220007 *ASTRID*. James Hsin-yu Chiang is supported by the PhD School of DTU Compute. Alberto Lluch Lafuente is partially supported by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu). Andrea Vandin is partially supported by the DFF project REDUCTO 9040-00224B.

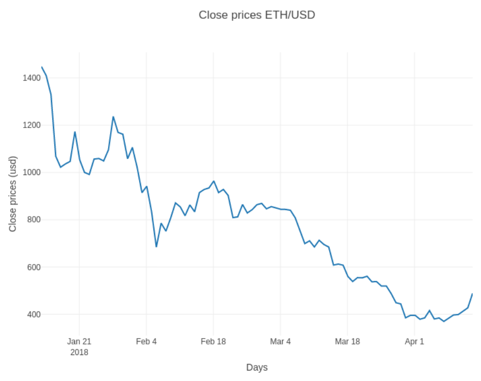
References

1. Aave, S.: Aave markets - webpage. `https://aave.com/` (2021)
2. Abdellatif, T., Brousmiche, K.L.: Formal verification of smart contracts based on users and blockchain behaviors models. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). pp. 1–5. IEEE (2018)
3. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Transactions on Modeling and Computer Simulation (TOMACS) **28**(1), 1–39 (2018)
4. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of Uniswap markets. Cryptoeconomic Systems (1) (2021). https://doi.org/10.21428/58320208.c9738e64
5. Bai, X., Cheng, Z., Duan, Z., Hu, K.: Formal modeling and verification of smart contracts. In: Proceedings of the 2018 7th International Conference on Software and Computer Applications. pp. 322–326 (2018)
6. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: SoK: Lending pools in decentralized finance. In: Financial Cryptography Workshops. LNCS, vol. 12676, pp. 553–578. Springer (2021). https://doi.org/10.1007/978-3-662-63958-0_40, the Lending Pools model used in this paper is taken from a preliminary version of the paper, published as arXiv preprint 2012.13230.
7. Basu, A., Bensalem, B., Bozga, M., Combaz, J., Jaber, M., Nguyen, T.H., Sifakis, J.: Rigorous component-based system design using the BIP framework. IEEE software **28**(3), 41–48 (2011)
8. Bernardi, T., Dor, N., Fedotov, A., Grossman, S., Immerman, N., Jackson, D., Nutz, A., Oppenheim, L., Pistiner, O., Rinetzky, N., et al.: Wip: Finding bugs automatically in smart contracts with parameterized invariants. `https://groups.csail.mit.edu/sdg/pubs/2020/sbc2020.pdf` (2020)
9. Bigi, G., Bracciali, A., Meacci, G., Tuosto, E.: Validation of decentralised smart contracts through game theory and formal methods. In: Programming Languages with Applications to Biology and Security, pp. 142–161. Springer (2015)
10. Boado, E.: Aave whitepaper. `https://github.com/aave/protocol-v2/blob/master/aave-v2-whitepaper.pdf` (2020), accessed on 26.02.2021 - commit aeded1520c667e59a564cf69f33a6e489b2fe489
11. Boado, E., Aave, S.: Aave protocol maximum liquidate threshold. `https://github.com/aave/aave-protocol/blob/1ff8418eb5c73ce233ac44bfb7541d07828b273f/contracts/lendingpool/LendingPoolLiquidationManager.sol#L181` (2021)

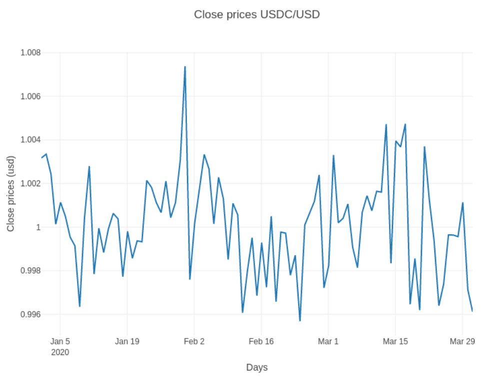
12. Buterin, V.: Ethereum whitepaper. https://ethereum.org/en/whitepaper/ (2013), accessed on 24.02.2021
13. Chitra, T., Evans, A.: Why stake when you can borrow? CoRR **abs/2006.11156** (2020), https://arxiv.org/abs/2006.11156
14. Chitra, T., Quaintance, M., Haber, S., Martino, W.: Agent-based simulations of blockchain protocols illustrated via Kadena's chainweb. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 386–395. IEEE (2019)
15. Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., Rubio, R., Talcott, C.: Maude manual (version 3.0). Tech. rep., Technical report, SRI International Computer Science Laboratory (2020)
16. Compound Labs, I.: Compound markets - webpage. https://compound.finance/markets (2021)
17. Dmouj, A.: Stock price modelling: Theory and practice. Masters Degree Thesis, Vrije Universiteit (2006)
18. Entriken, W.: Introduction to smart contracts. https://ethereum.org/en/developers/docs/smart-contracts/ (2020), accessed on 27.02.2021
19. Gudgeon, L., Perez, D., Harz, D., Livshits, B., Gervais, A.: The decentralized financial crisis. In: 2020 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 1–15. IEEE (2020)
20. Hull, J.C.: Options futures and other derivatives. Pearson Education India (2003)
21. Jeffrey, G.: Compound price oracle attack. https://news.bitcoin.com/100-million-liquidated-on-defi-protocol-compound-following-oracle-exploit/ (2020)
22. Kao, H.T., Chitra, T., Chiang, R., Morrow, J.: An analysis of the market risk to participants in the Compound protocol. In: Third International Symposium on Foundations and Applications of Blockchains (2020)
23. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
24. Leshner, R., Hayes, G.: Compound: The money market protocol. https://compound.finance/documents/Compound.Whitepaper.v04.pdf (2019)
25. Mirelli, M.: A formal verification tool for Lending Pools. Master's thesis, Aalto University. School of Science (2021), http://urn.fi/URN:NBN:fi:aalto-202108298504
26. Mirelli, M.: A Maude simulator for Lending Pools. https://github.com/MMirelli/maude-lp (2021), accessed on 22.06.2022 - commit 2dae39b035938f5f9791040c53121fb473b4b7dd
27. Perez, D., Werner, S.M., Xu, J., Livshits, B.: Liquidations: DeFi on a knife-edge. In: Financial Cryptography and Data Security. LNCS, vol. 12675, pp. 457–476. Springer (2021). https://doi.org/10.1007/978-3-662-64331-0_24
28. Peterins, E., Flatow, J., Hayes, G., Wolff, M., Greenberg, A.: Compound protocol maximum liquidate threshold. https://github.com/compound-finance/compound-protocol/blob/4e99ea3a64ab4f1bdf9c07c7a1bf325db09ab809/scenario/src/Event/ComptrollerEvent.ts#L170 (2021)
29. Pulse: Defi pulse - webpage. https://defipulse.com (2021), accessed on 07.06.2021
30. Qin, K., Zhou, L., Livshits, B., Gervais, A.: Attacking the DeFi ecosystem with flash loans for fun and profit. In: Financial Cryptography. LNCS, vol. 12674, pp. 3–32. Springer (2021). https://doi.org/10.1007/978-3-662-64322-8_1

31. Sebastio, S., Vandin, A.: Multivesta: Statistical model checking for discrete event simulators. Tech. rep., IMT Institute for Advanced Studies Lucca (2013)
32. Tolmach, P., Li, Y., Lin, S.W., Liu, Y.: Formal analysis of composable defi protocols. arXiv preprint arXiv:2103.00540 (2021)
33. Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F.: Automated and distributed statistical analysis of economic agent-based models. arXiv preprint arXiv:2102.05405 (2021)
34. Wackerow, P., Rhechler: Decentralized finance (defi) - webpage. `https://ethereum.org/en/defi/` (2021), accessed on 02.06.2021
35. Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: SoK: Decentralized Finance (DeFi). CoRR **abs/2101.08778** (2021), `https://arxiv.org/abs/2101.08778`
36. Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in DeFi protocols. In: IEEE Symposium on Security and Privacy. pp. 919–936. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00113

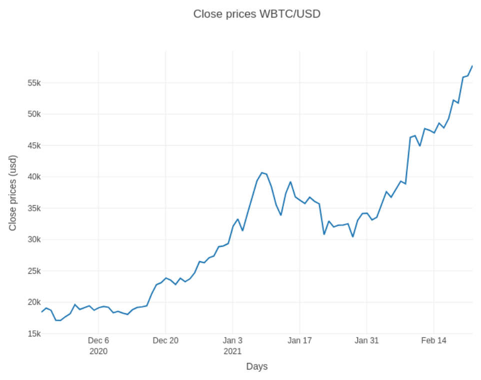
A Figures



(a) 13/01/2018-14/04/2018

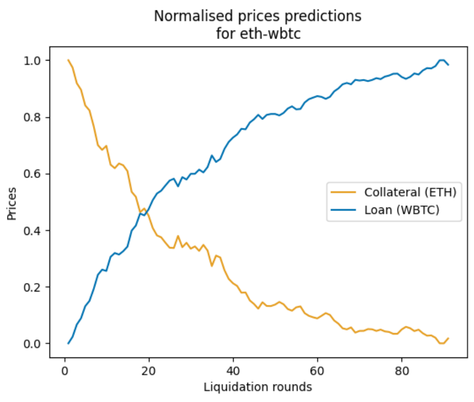


(b) 01/01/2020-01/04/2020

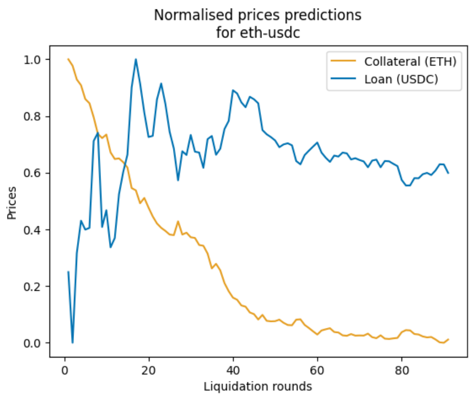


(c) 24/11/2020-23/02/2021

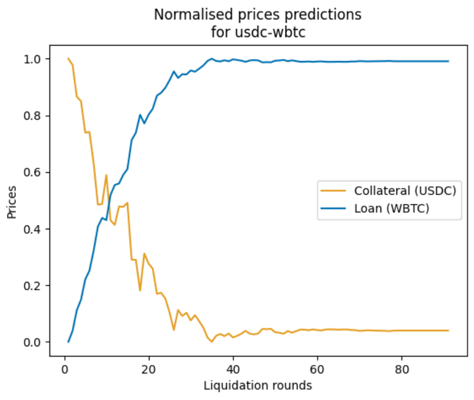
Fig. 9: Trimester closing prices, collected from CoinGecko APIs



(a)

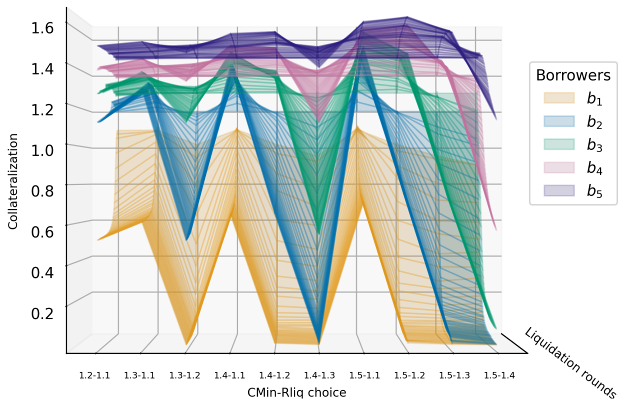


(b)

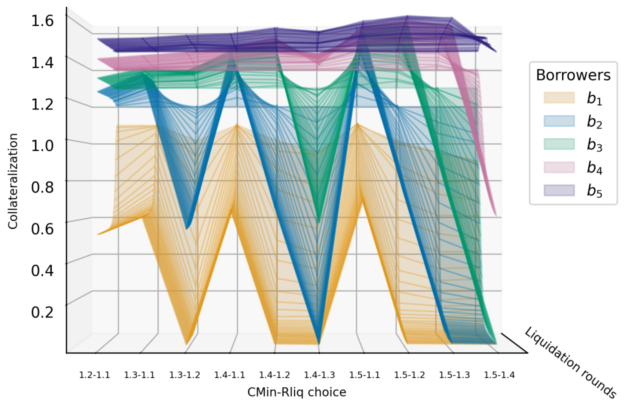


(c)

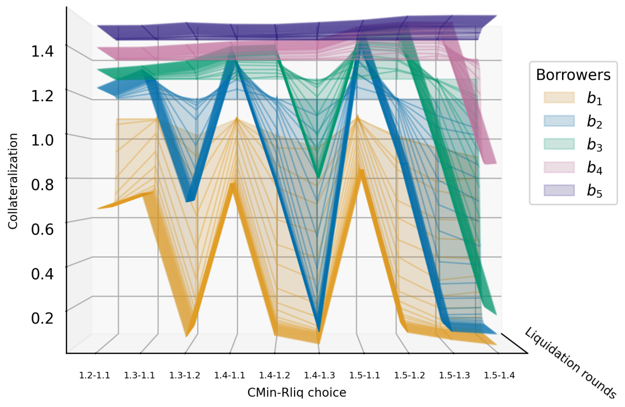
Fig. 10: Prices predictions produced, for each scenario in Table 3, by GBMs instantiated with the parameters in Figure 5.



(a) Scenario: eth-wbtc.

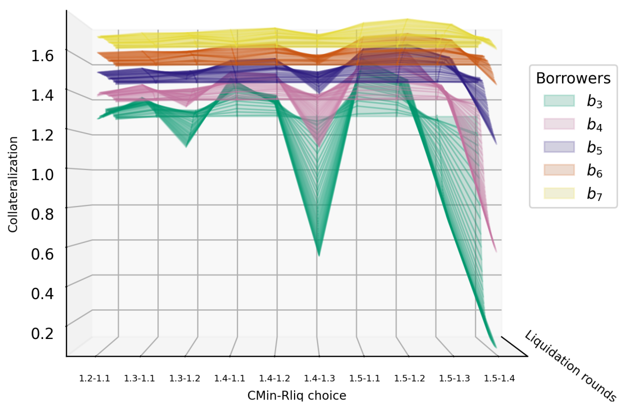


(b) Scenario: eth-usdc.

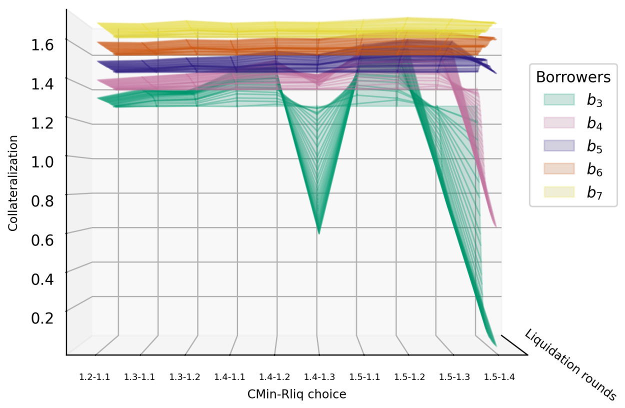


(c) Scenario: usdc-wbtc.

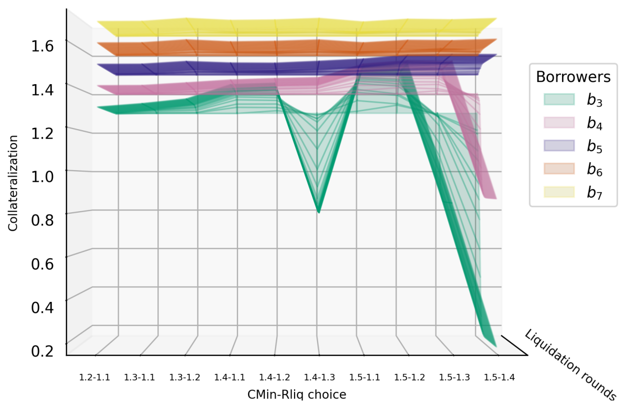
Fig. 11: Per-borrower collateralization (b_1 to b_5) in the three prices scenarios, for varying `CMin-rliq` choices.



(a) Scenario: eth-wbtc.



(b) Scenario: eth-usdc.



(c) Scenario: usdc-wbtc.

Fig. 12: Per-borrower collateralization (b_3 to b_7) in the three prices scenarios, for varying `CMin-rliq` choices.

Maximizing Extractable Value from Automated Market Makers

Publication Information

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. "Maximizing extractable value from automated market makers." In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pp. 3-19. Cham: Springer International Publishing, 2022.

Contribution

- Co-author.

Maximizing Extractable Value from Automated Market Makers

Massimo Bartoletti[1], James Hsin-yu Chiang[2], Alberto Lluch Lafuente[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
[2] Technical University of Denmark, DTU Compute, Copenhagen, Denmark

Abstract. Automated Market Makers (AMMs) are decentralized applications that allow users to exchange crypto-tokens without the need for a matching exchange order. AMMs are one of the most successful DeFi use cases: indeed, major AMM platforms process a daily volume of transactions worth USD billions. Despite their popularity, AMMs are well-known to suffer from transaction-ordering issues: adversaries can influence the ordering of user transactions, and possibly front-run them with their own, to extract value from AMMs, to the detriment of users. We devise an effective procedure to construct a strategy through which an adversary can *maximize* the value extracted from user transactions.

Keywords: miner extractable value, front-running, decentralized finance

1 Introduction

Decentralized finance (DeFi) is emerging as an alternative to traditional finance, boosted by blockchains, crypto-tokens and smart contracts [17]. *Automated Market Makers (AMMs)* — one of the main DeFi applications — allow users to exchange crypto-tokens without the need to find another party wanting to participate in the exchange. Major AMM platforms like e.g. Uniswap, Curve Finance, and SushiSwap, hold dozens of billions of USD and process hundreds of millions worth of transactions daily [8,1,5].

AMMs are sensitive to *transaction-ordering attacks*, where adversaries who can influence the ordering of transactions in the blockchain exploit this power to *extract value* from user transactions [13,15,16,20]. We illustrate this kind of attacks through a minimal example. Assume a Uniswap-like AMM holding 100 units of a crypto-token τ_0 and 100 units of another token τ_1, and assume that both tokens have the same price in the reference currency (say, USD 1,000). Now, suppose that user A wants to swap 20 units of τ_0 in her wallet for at least 15 units of τ_1. This requires to append to the blockchain a transaction of the form $\mathsf{A} : \mathsf{swap}^0(20 : \tau_0, 15 : \tau_1)$, where the prefix A indicates the wallet involved in the transaction, swap is the called AMM function, and the superscript 0 indicates the swap direction, i.e. deposit $20 : \tau_0$ to receive back at least $15 : \tau_1$ (a superscript 1 would indicate the opposite direction). In a *constant-product* AMM platform like Uniswap, the actual amount of τ_1 transferred to A must be such that the product between the AMM reserves remains constant before and after a swap.

Now, suppose that an adversary M (possibly a miner) observes A's transaction in the txpool, and appends to the blockchain the following *sandwich*:

$$\mathsf{M} : \mathsf{swap}^0(5.9 : \tau_0, 5.5 : \tau_1) \;\; \mathsf{A} : \mathsf{swap}^0(20 : \tau_0, 15 : \tau_1) \;\; \mathsf{M} : \mathsf{swap}^1(25.9 : \tau_0, 20.6 : \tau_1)$$

where the last transaction is in the opposite direction, i.e. M sends $20.6 : \tau_1$ to receive at least $25.9 : \tau_0$. As a result, A only yields the *minimum* amount of $15 : \tau_1$ in return for $20 : \tau_0$. This implies that USD 5,000 have been gained by M and lost by A. This has been called *Miner Extractable Value* (MEV) [13].

Recent works study this and other kinds of attacks to AMMs [13,16,19,20]: however, all these approaches are preeminently *empirical*, as they focus on the definition of heuristics to extract value from AMMs, and on their evaluation in the wild. To the best of our knowledge, a general solution to obtain *optimal* MEV is still missing, even in the special case of constant-product AMMs.

To exemplify a case where prior approaches fail to extract optimal MEV, consider the following set of user transactions, containing a swap of τ_0 for τ_1, a deposit of units of τ_0 and τ_1, and a redeem of units of minted (liquidity) tokens:

$$\{ \quad \mathsf{A} : \mathsf{swap}^0(40 : \tau_0, 35 : \tau_1), \; \mathsf{A} : \mathsf{dep}(30 : \tau_0, 40 : \tau_1), \; \mathsf{A} : \mathsf{rdm}(10 : (\tau_0, \tau_1)) \quad \}$$

Here, both the swap and the dep transactions would be rejected. For instance, the constant-product invariant dictates that $40 : \tau_0$ sent by the user swap in the initial AMM state $(100 : \tau_0, 100 : \tau_1)$ will return exactly $28.6 : \tau_1$; since the swap transaction requires $35 : \tau_1$, it would be discarded. The known heuristics here fail to extract any value. Even considering only the swap, the sandwich would not be profitable for M, since it requires the *same* direction for M's and A's swap (offer τ_0 to obtain τ_1), making A's swap not enabled. Further, the known heuristics only operate on swap actions, neglecting user deposits and redeems. This paper proposes a layered construction to extract the *maximum value* from all user transactions, through a multi-layer sandwich that we call *Dagwood sandwich.* In our example, M's strategy would be to fire the following three-layer sandwich:

$$\begin{array}{ll} \mathsf{M} : \mathsf{swap}^1(11 : \tau_0, 13 : \tau_1) & \mathsf{A} : \mathsf{swap}^0(40 : \tau_0, 35 : \tau_1) \\ \mathsf{M} : \mathsf{swap}^1(42 : \tau_0, 38 : \tau_1) & \mathsf{A} : \mathsf{dep}(30 : \tau_0, 40 : \tau_1) \\ \mathsf{M} : \mathsf{swap}^0(18 : \tau_0, 21 : \tau_1) & \end{array}$$

The first transaction is a swap in the opposite direction (i.e., pay τ_1 to get τ_0) w.r.t. the subsequent user swap, unlike in the classical sandwich heuristic. M's second swap enables A's deposit; the final swap is an arbitrage move [9]. The user redeem is dropped, since it would negatively contribute to M's profit. By firing the transaction sequence above, M can extract approx. USD 5,700 from A, improving over swap-only attacks, that would only extract USD 5,000.

Contributions To the best of our knowledge, this work is the first to formalise the *MEV game* for AMMs (Section 3), and the first to effectively construct optimal solutions which attack all types of transactions supported by constant-product AMMs (Section 4). We discuss in Section 6 the applicability of our technique in the wild. The proofs of our statements are in Appendix A.

2 Automated Market Makers

We assume a set $\mathbb{T}_0$ of ***atomic token types*** (ranged over by $\tau, \tau', \ldots$), representing native cryptocurrencies and application-specific tokens. We denote by $\mathbb{T}_1 = \mathbb{T}_0 \times \mathbb{T}_0$ the set of ***minted token types***, representing shares in AMMs. In our model, tokens are *fungible*, i.e. individual units of the same type are interchangeable. In particular, amounts of tokens of the same type can be split into smaller parts, and two amounts of tokens of the same type can be joined. We use v, v', r, r' to range over nonnegative real numbers ($\mathbb{R}_0^+$), and we write $r : \tau$ to denote r units of token type $\tau \in \mathbb{T} = \mathbb{T}_0 \cup \mathbb{T}_1$.

We model the ***wallet*** of a user A as a term $\mathsf{A}[\sigma]$, where the partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_0^+$ represents A's token holdings, and write $\mathsf{A}[\_]$ if the wallet balance is clear from context. We denote with $\operatorname{dom}(\sigma)$ the domain of σ. An ***AMM*** is a pair of the form $(r_0 : \tau_0, r_1 : \tau_1)$, representing the fact that the AMM is holding r_0 units of τ_0 and r_1 units of τ_1. We denote by $\mathit{res}_{\tau_0,\tau_1}(\Gamma)$ the *reserves* of τ_0 and τ_1 in Γ, i.e. $\mathit{res}_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1)$ if $(r_0 : \tau_0, r_1 : \tau_1)$ is in Γ.

A ***state*** is a composition of wallets and AMMs, represented as a term:

$$\mathsf{A}_1[\sigma_1] \mid \cdots \mid \mathsf{A}_n[\sigma_n] \mid (r_1 : \tau_1, r_1' : \tau_1') \mid \cdots \mid (r_k : \tau_k, r_k' : \tau_k')$$

where: (i) all A_i are distinct, (ii) the token types in an AMM are *distinct*, and (iii) distinct AMMs cannot hold exactly the same token types. Note that two AMMs can have a common token type τ, as in $(r_1 : \tau_1, r : \tau) \mid (r' : \tau, r_2 : \tau_2)$, thus enabling indirect trades between token pairs not directly provided by any AMM. We use $\Gamma, \Gamma', \ldots$ to range over states. For a base term Q (either wallet or AMM), we write $Q \in \Gamma$ when $\Gamma = Q \mid \Gamma'$, for some Γ', where we assume that two states are equivalent when they contain the same base terms.

We define the ***supply*** of a token type τ in a state Γ as the sum of the balances of τ in all the wallets and the AMMs occurring in Γ. Formally:

$$\mathit{sply}_\tau(\mathsf{A}[\sigma]) = \begin{cases} \sigma(\tau) & \text{if } \tau \in \operatorname{dom}(\sigma) \\ 0 & \text{otherwise} \end{cases} \qquad \mathit{sply}_\tau(r_0 : \tau_0, r_1 : \tau_1) = \begin{cases} r_i & \text{if } \tau = \tau_i \\ 0 & \text{otherwise} \end{cases}$$

and the supply of τ in $\Gamma \mid \Gamma'$ is the summation $\mathit{sply}_\tau(\Gamma) + \mathit{sply}_\tau(\Gamma')$.

We model the interaction between users and AMMs as a transition system between states. A transition $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ represents the evolution of the state Γ into Γ' upon the execution of the ***transaction*** T. The possible transactions are:

- $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, which allows A to ***deposit*** $v_0 : \tau_0$ and $v_1 : \tau_1$ to an AMM, receiving in return units of the minted token (τ_0, τ_1).
- $\mathsf{A} : \mathsf{swap}^d(v_0 : \tau_0, v_1 : \tau_1)$ with $d \in \{0, 1\}$, which allows A to ***swap*** tokens, i.e. transfer $v_d : \tau_d$ to an AMM, and receive in return *at least* $v_{1-d} : \tau_{1-d}$.

- $\mathsf{A} : \mathsf{rdm}(v : \tau)$, which allows to A ***redeem*** v units of minted token $\tau = (\tau_0, \tau_1)$ from an AMM, receiving in return units of the atomic tokens τ_0 and τ_1.

We now formalise the one-step relation $\xrightarrow{\mathsf{T}}$ through rewriting rules, inspired by [9]. We use the standard notation $\sigma\{{}^{v}\!/\!{}_{x}\}$ to update a partial map σ at point x: namely, $\sigma\{{}^{v}\!/\!{}_{x}\}(x) = v$, while $\sigma\{{}^{v}\!/\!{}_{x}\}(y) = \sigma(y)$ for $y \neq x$. For a partial map $\sigma \in \mathbb{T} \rightharpoonup \mathbb{R}_0^+$, a token type $\tau \in \mathbb{T}$ and a partial operation $\circ \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \rightharpoonup \mathbb{R}_0^+$, we define the partial map $\sigma \circ v : \tau$ (updating τ's balance in σ by v) as follows:

$$\sigma \circ v : \tau = \begin{cases} \sigma\{{}^{\sigma(\tau)\,\circ\, v}\!/\!{}_{\tau}\} & \text{if } \tau \in \operatorname{dom}\sigma \text{ and } \sigma(\tau) \circ v \in \mathbb{R}_0^+ \\ \sigma\{{}^{v}\!/\!{}_{\tau}\} & \text{if } \tau \notin \operatorname{dom}\sigma \end{cases}$$

Deposit Any user can create an AMM for a token pair (τ_0, τ_1), provided that such an AMM is not already present in the state. This is achieved by the transaction $\mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$, through which A transfers $v_0 : \tau_0$ and $v_1 : \tau_1$ to the new AMM. In return, A receives an amount of units of a new token type (τ_0, τ_1), which is minted by the AMM. We formalise this behaviour by the rule:

$$\frac{\sigma(\tau_i) \geq v_i > 0 \;\; (i \in \{0,1\}) \qquad \tau_0 \neq \tau_1 \qquad \tau_0, \tau_1 \in \mathbb{T}_0 \qquad (\_ : \tau_0, \_ : \tau_1), (\_ : \tau_1, \_ : \tau_0) \notin \Gamma}{\mathsf{A}[\sigma] \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v_0 : (\tau_0, \tau_1)] \mid (v_0 : \tau_0, v_1 : \tau_1) \mid \Gamma} \;[\text{Dep0}]$$

Once an AMM is created, any user can deposit tokens into it, as long as doing so preserves the ratio of the token holdings in the AMM. When a user deposits $v_0 : \tau_0$ and $v_1 : \tau_1$ to an existing AMM, it receives in return an amount of minted tokens of type (τ_0, τ_1). This amount is the ratio between the deposited amount v_0 and the redeem rate of (τ_0, τ_1) in the current state Γ. This redeem rate is the ratio between the amount r_0 of τ_0 stored in the AMM, and the total supply $\mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)$ of the minted token in the state.

$$\frac{\sigma(\tau_i) \geq v_i > 0 \;\; (i \in \{0,1\}) \qquad r_1 v_0 = r_0 v_1 \qquad v = \frac{v_0}{r_0} \cdot \mathit{sply}_{(\tau_0,\tau_1)}(\Gamma)}{\begin{array}{l}\Gamma \;=\; \mathsf{A}[\sigma] \;\mid\; (r_0 : \tau_0, r_1 : \tau_1) \;\mid\; \Gamma' \xrightarrow{\mathsf{A}:\mathsf{dep}(v_0:\tau_0, v_1:\tau_1)} \\ \qquad \mathsf{A}[\sigma - v_0 : \tau_0 - v_1 : \tau_1 + v : (\tau_0, \tau_1)] \;\mid\; (r_0 + v_0 : \tau_0, r_1 + v_1 : \tau_1) \;\mid\; \Gamma'\end{array}} \;[\text{Dep}]$$

The premise $r_1 v_0 = r_0 v_1$ ensures that the ratio between the reserves of τ_0 and τ_1 in the AMM is preserved, i.e. ${}^{r_1 + v_1}\!/\!{}_{r_0 + v_0} \;=\; {}^{r_1}\!/\!{}_{r_0}$.

Swap Any user A can swap units of τ_0 in her wallet for units of τ_1 in an AMM $(r_0 : \tau_0, r_1 : \tau_1)$, or *vice versa* swap units of τ_1 in the wallet for units of τ_0 in the AMM. This is achieved by the transaction $\mathsf{A} : \mathsf{swap}^d(v_0 : \tau_0, v_1 : \tau_1)$, where $d \in \{0, 1\}$ is the *swap direction*. If $d = 0$ ("left" swap), then v_0 is the amount of τ_0 transferred from A's wallet to the AMM, while v_1 is a *lower bound* on the amount of τ_1 that A will receive in return. Conversely, if $d = 1$ ("right" swap), then v_1 is the amount of τ_1 transferred from A's wallet, and v_0 is a lower bound on the received amount of τ_0. The actual amount v of received units of τ_{1-d} must satisfy the ***constant-product invariant*** [18], as in Uniswap [7], SushiSwap [6] and other common AMMs implementations:

$$r_0 \cdot r_1 \;=\; (r_d + v_d) \cdot (r_{1-d} - v)$$

Formally, for $d \in \{0,1\}$ we define:

$$\frac{\sigma(\tau_d) \geq v_d > 0 \qquad v = \frac{r_{1-d} \cdot v_d}{r_d + v_d} \qquad 0 < v_{1-d} \leq v}{\begin{array}{l}\mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}^d(v_0:\tau_0, v_1:\tau_1)} \\ \mathsf{A}[\sigma - v_d : \tau_d + v : \tau_{1-d}] \mid (r_0 : \tau_0, r_1 : \tau_1) + v_d : \tau_d - v : \tau_{1-d} \mid \Gamma\end{array}} \text{[Swap]}$$

where we define the update of the units of τ in an AMM, for $\circ \in \{+, -\}$, as:

$$(r_0 : \tau_0, r_1 : \tau_1) \circ v : \tau = \begin{cases} (r_0 \circ v : \tau_0, r_1 : \tau_1) & \text{if } \tau = \tau_0 \text{ and } r_0 \circ v \in \mathbb{R}_0^+ \\ (r_0 : \tau_0, r_1 \circ v : \tau_1) & \text{if } \tau = \tau_1 \text{ and } r_1 \circ v \in \mathbb{R}_0^+ \end{cases}$$

Redeem Users can redeem units of a minted token (τ_0, τ_1) for units of the underlying atomic tokens τ_0 and τ_1. Each unit of (τ_0, τ_1) can be redeemed for equal fractions of τ_0 and τ_1 remaining in the AMM:

$$\frac{\sigma(\tau_0, \tau_1) \geq v > 0 \qquad v_0 = v \frac{r_0}{sply_{(\tau_0,\tau_1)}(\Gamma)} \qquad v_1 = v \frac{r_1}{sply_{(\tau_0,\tau_1)}(\Gamma)}}{\begin{array}{l}\Gamma = \mathsf{A}[\sigma] \mid (r_0 : \tau_0, r_1 : \tau_1) \mid \Gamma' \xrightarrow{\mathsf{A}:\mathsf{rdm}(v:(\tau_0,\tau_1))} \\ \qquad \mathsf{A}[\sigma + v_0 : \tau_0 + v_1 : \tau_1 - v : (\tau_0, \tau_1)] \mid (r_0 - v_0 : \tau_0, r_1 - v_1 : \tau_1) \mid \Gamma'\end{array}} \text{[Rdm]}$$

A key property of the transition system is *determinism*, i.e. if $\Gamma \xrightarrow{\mathsf{T}} \Gamma'$ and $\Gamma \xrightarrow{\mathsf{T}} \Gamma''$, then the states Γ' and Γ'' are equivalent. We denote with *type*(T) the type of T (i.e., dep, swap, rdm), and with *usr*(T) the user issuing T. For a sequence of transactions $\lambda = \mathsf{T}_1 \cdots \mathsf{T}_n$, we write $\Gamma \xrightarrow{\lambda} \Gamma'$ whenever there exist intermediate states $\Gamma_1, \dots \Gamma_{n-1}$ such that $\Gamma \xrightarrow{\mathsf{T}_1} \Gamma_1 \xrightarrow{\mathsf{T}_2} \cdots \xrightarrow{\mathsf{T}_{n-1}} \Gamma_{n-1} \xrightarrow{\mathsf{T}_n} \Gamma'$. When this happens, we say that λ is *enabled* in Γ, or just $\Gamma \xrightarrow{\lambda}$. A state Γ is *reachable* if there exist some Γ_0 only containing wallets with atomic tokens and some λ such that $\Gamma_0 \xrightarrow{\lambda} \Gamma$.

3 The MEV game

The model in the previous section defines how the state of AMMs and wallets evolves upon a sequence of transactions, but it does not specify how this sequence is formed. We specify this as a single-player, single-round game where the only player is an adversary M who attempts to maximize its MEV. Accordingly, we call this the ***MEV game***. The *initial state* of the game is given by a reachable state Γ (not including M's wallet) and by a finite multiset $\mathfrak{X}$ of user transactions, representing the pool of pending transactions (also called ***txpool***). The *moves* of M are pairs (σ, λ), where σ is M's initial balance, and λ is a sequence formed by (part of) the transactions in $\mathfrak{X}$, and by any number of M's transactions. We require that the sequence λ in a move is enabled in Γ. The MEV game assumes the following (see Section 6 for a discussion thereof):

1. Users balances in Γ are sufficiently high to not interfere with the validity of any specific ordering of actions in $\mathfrak{X}$.
2. The balance σ of M does not include minted tokens.

3. The length of the sequence λ is unbounded.
4. Prices of atomic tokens are fixed throughout the game execution.

Besides the above, some further assumptions are implied by our AMM model:

5. AMMs only hold atomic tokens (this is a consequence of [Dep0]).
6. Swap actions do not require fees (this is a consequence of [Swap]).
7. There are no transaction fees.
8. Interval constraints on received token amounts are modelled in swaps only.

A *solution* to the game is a move that maximizes M's *gain*, i.e. the change in M's net worth after performing the sequence λ from Γ. Intuitively, the net worth of a user is the overall *value* of tokens in her wallet. To define it, we need to associate a ***price*** to each token. We assume that the prices of atomic tokens are given by an oracle $P \in \mathbb{T}_0 \to \mathbb{R}_0^+$: naturally, the MEV game solution will need to be recomputed should the price of atomic tokens be updated. The price $P_\Gamma(\tau_0, \tau_1)$ of a minted token (τ_0, τ_1) in a state Γ is defined as follows:

$$P_\Gamma(\tau_0, \tau_1) = \frac{r_0 \cdot P(\tau_0) + r_1 \cdot P(\tau_1)}{sply_{(\tau_0,\tau_1)}(\Gamma)} \quad \text{if } res_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1) \tag{1}$$

Minted tokens are priced such that the net worth of a user is preserved when she deposits or redeems minted tokens in her wallet. We assume that the reserves in an AMM are never reduced to zero in an execution, in order to preserve equality of minted token prices between two states with equal reserves, thereby facilitating proofs and analysis. While our semantics of AMMs allows reserves to be emptied, we note that this does not occur in practice, as it would halt the operation of the respective AMM pair. We define the ***net worth*** of a user A in a state Γ such that $\mathsf{A}[\sigma] \in \Gamma$ as follows:

$$W_{\mathsf{A}}(\Gamma) \;=\; \textstyle\sum_{\tau \in \mathrm{dom}\,(\sigma)} \sigma(\tau) \cdot P_\Gamma(\tau) \tag{2}$$

and we denote by $G_{\mathsf{A}}(\Gamma, \lambda)$ the ***gain*** of user A upon performing a sequence of transactions λ enabled in state Γ (if λ is not enabled, the gain is zero):

$$G_{\mathsf{A}}(\Gamma, \lambda) \;=\; W_{\mathsf{A}}(\Gamma') - W_{\mathsf{A}}(\Gamma) \qquad \text{if } \Gamma \xrightarrow{\lambda} \Gamma' \tag{3}$$

A ***rational player*** is a player which, for all initial states $(\Gamma, \mathcal{X})$ of the game, always chooses a move (σ, λ) that maximizes the function $G_{\mathsf{M}}(\mathsf{M}[x] \mid \Gamma,\ y)$ on variables x and y. We define the ***miner extractable value*** in $(\Gamma, \mathcal{X})$ as the gain obtained by a rational player by applying such a solution (σ, λ), i.e.:

$$MEV(\Gamma, \mathcal{X}) \;=\; G_{\mathsf{M}}(\mathsf{M}[\sigma] \mid \Gamma, \lambda)$$

Lemma 1 states that firing transactions preserves the *global* net worth, i.e. the gains of some users are balanced by equal overall losses of other users.

Lemma 1. $\sum_{\mathsf{A}} G_{\mathsf{A}}(\Gamma, \mathsf{T}) = 0$.

By using a simple inductive argument, we can extend Lemma 1 to sequences of transactions: if $\Gamma \xrightarrow{\lambda} \Gamma'$, then the summation of the gains $G_{\mathsf{A}}(\Gamma, \lambda)$ over all users (including M) is 0. Hence, the MEV game is zero-sum. The following lemma ensures that deposit and redeem actions do not directly affect the net worth of the user who performs them.

Lemma 2. *If type*$(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$*, then* $G_{usr(\mathsf{T})}(\Gamma, \mathsf{T}) = 0$.

Finally, we note that prices of a minted token in two states are equal if the reserve ratio in the two states are as well.

Lemma 3. *Let* $\Gamma \xrightarrow{\lambda} \Gamma'$*, and let* $res_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1)$*,* $res_{\tau_0,\tau_1}(\Gamma') = (r_0', r_1')$*. Then,* $P_{\Gamma}(\tau_0, \tau_1) = P_{\Gamma'}(\tau_0, \tau_1)$ *if and only if* $r_0/r_1 = r_0'/r_1'$*.*

4 Solving the MEV game

By Lemma 1, a move which minimizes the gain of all users but M must maximize M's gain, and therefore is a solution to the MEV game. More formally, we have:

Corollary 1. $G_{\mathsf{M}}(\Gamma, \lambda)$ *is maximized iff* $G_{\mathsf{A}}(\Gamma, \lambda)$ *is minimized for all* $\mathsf{A} \neq \mathsf{M}$*.*

The net worth W_{A} of a user A can be decomposed in two parts: W^0_{A}, which accounts for the atomic tokens, and W^1_{A}, which accounts for the minted tokens:

$$W^0_{\mathsf{A}}(\Gamma) = \textstyle\sum_{\tau \in \mathbb{T}_0} \sigma_{\mathsf{A}}(\tau) \cdot P(\tau) \qquad W^1_{\mathsf{A}}(\Gamma) = \textstyle\sum_{\tau \in \mathbb{T}_1} \sigma_{\mathsf{A}}(\tau) \cdot P_{\Gamma}(\tau) \tag{4}$$

This provides M with two levers to reduce the users' gain: token balances, and the price of minted tokens. To use the first lever, M needs to exploit user actions in the txpool $\mathcal{X}$ of the MEV game. For the second lever, since the prices of atomic tokens ($\tau \in \mathbb{T}_0$) are fixed, M can only influence the price of minted tokens ($\tau \in \mathbb{T}_1$). This can be achieved by performing actions on the respective AMMs.

In the rest of the section we devise an *optimal* strategy to exploit these two levers. Intuitively, our strategy constructs a multi-layer ***Dagwood Sandwich***[3], containing an ***inner layer*** for each exploitable user action in $\mathcal{X}$, which M *front-runs* by a swap transaction to enable it (if necessary), and a ***final layer*** of swaps by M to minimize the prices of all minted tokens.

The construction of the final layer of the Dagwood sandwich is shown in §4.1, while the construction of the inner layers is presented in §4.2.

4.1 Price minimization

Lemma 4 below states that, in any state, M can minimize the price of a minted token by using a single swap, at most. In particular, this minimization can always be performed in the final layer of the Dagwood sandwich.

[3] We name it after Dagwood Bumstead, a comic strip character who is often illustrated while producing enormous multi-layer sandwiches.

Lemma 4. *There exists a function P^{min} such that if $\mathsf{M}[\sigma] \mid \Gamma \rightarrow^* \mathsf{M}[\sigma'] \mid \Gamma'$ then: (i) $P_{\Gamma'}(\tau_0, \tau_1) \geq P^{min}_{\Gamma}(\tau_0, \tau_1)$; (ii) there exist σ'' and λ consisting at most of a swap by M such that $\mathsf{M}[\sigma''] \mid \Gamma' \xrightarrow{\lambda} \mathsf{M}[\_] \mid \Gamma''$ and $P_{\Gamma''}(\tau_0, \tau_1) = P^{min}_{\Gamma}(\tau_0, \tau_1)$.*

In order to construct the swap transaction which minimizes the price of a minted token (τ_0, τ_1) in Γ, we need some auxiliary definitions. For each swap direction $d \in \{0, 1\}$, we define the ***canonical swap values*** as:

$$w^d_d(\tau_0, \tau_1, \Gamma) = \sqrt{\frac{P(\tau_{1-d})}{P(\tau_d)} \cdot r_0 \cdot r_1} - r_d \qquad w^d_{1-d}(\tau_0, \tau_1, \Gamma) = \frac{r_{1-d} \cdot w^d_d(\tau_0, \tau_1, \Gamma)}{r_d + w^d_d(\tau_0, \tau_1, \Gamma)}$$

Intuitively, w^d_d is the amount of tokens *deposited* in a swap of direction d: it is defined such that, after the swap, the AMM reaches an equilibrium, where the ratio of the AMM reserves is equal to the (inverse) ratio of the token prices. Instead, w^d_{1-d} is the amount of tokens *received* after the swap, i.e. it is the unique value for which the swap invariant is satisfied.

If both $w^0_0(\tau_0, \tau_1, \Gamma) \leq 0$ and $w^1_1(\tau_0, \tau_1, \Gamma) \leq 0$, then the price of the minted token (τ_0, τ_1) is already minimized. Otherwise, if $w^d_d(\tau_0, \tau_1, \Gamma) > 0$ for some d (and there may exist at most one d for which this holds), then we define the ***price minimization transaction*** $\mathsf{X}^d(\tau_0, \tau_1, \Gamma)$ as:

$$\mathsf{M} : \mathsf{swap}^d(\, w^d_0(\tau_0, \tau_1, \Gamma) : \tau_0,\ w^d_1(\tau_0, \tau_1, \Gamma) : \tau_1 \,) \tag{5}$$

Theorem 1 constructs the final layer of the Dagwood sandwich. We show that this layer is the solution of the MEV game on an empty txpool. This is because if M cannot leverage user transactions, the solution is just to minimize the price of all minted tokens. The solution is obtained by sequencing price minimization transactions on all AMMs. Since the price of a minted token is a function of the reserves of the corresponding AMM, this can be done in any order.

Theorem 1. *Let $\Gamma = \|_{i \in I}(r_{i,0} : \tau_{i,0}, r_{i,1} : \tau_{i,1}) \mid \Gamma_w$, where Γ_w only contains wallets. For all $j \in I$ and $d \in \{0, 1\}$, let $v^d_j = w^d_d(\tau_{j,0}, \tau_{j,1}, \Gamma)$, and let:*

$$\sigma_j = \begin{cases} v^d_j : \tau_{j,d} & \text{if } v^d_j > 0 \\ 0 & \text{if } v^0_j, v^1_j \leq 0 \end{cases} \qquad \lambda_j = \begin{cases} \mathsf{X}^d(\tau_{j,0}, \tau_{j,1}, \Gamma) & \text{if } v^d_j > 0 \\ \varepsilon & \text{if } v^0_j, v^1_j \leq 0 \end{cases}$$

Then, $(\sigma_1 \cdots \sigma_n, \lambda_1 \cdots \lambda_n)$ is a solution to the game $(\Gamma, \mathcal{X})$ for an empty $\mathcal{X}$.

4.2 Constructing the inner layers

Consider a solution (σ, λ) to the game $(\mathsf{A}[\sigma_\mathsf{A}] \mid \Gamma, \mathcal{X})$, and let:

$$\mathsf{M}[\sigma] \mid \mathsf{A}[\sigma_\mathsf{A}] \mid \Gamma \xrightarrow{\lambda} \mathsf{M}[\sigma'] \mid \mathsf{A}[\sigma'_\mathsf{A}] \mid \Gamma'$$

By decomposing the net worth as in (4), we find that A's gain for λ is:

$$\begin{aligned} G_\mathsf{A}(\mathsf{M}[\sigma] \mid \mathsf{A}[\sigma_\mathsf{A}] \mid \Gamma, \lambda) &= W^0_\mathsf{A}(\Gamma') - W^0_\mathsf{A}(\Gamma) + W^1_\mathsf{A}(\Gamma') - W^1_\mathsf{A}(\Gamma) \\ &= \sum_{\tau \in \mathbb{T}_0} (\sigma'_\mathsf{A}(\tau) - \sigma_\mathsf{A}(\tau)) \cdot P(\tau) + \sum_{\tau \in \mathbb{T}_1} (\sigma'_\mathsf{A}(\tau) \cdot P_{\Gamma'}(\tau) - \sigma_\mathsf{A}(\tau) \cdot P_\Gamma(\tau)) \end{aligned}$$

Since λ is a solution, by Lemma 4 we can replace $P_{\Gamma'}(\tau)$ with $P_{\Gamma}^{min}(\tau)$:

$$= \sum_{\tau \in \mathbb{T}_0} \big(\sigma'_{\mathsf{A}}(\tau) - \sigma_{\mathsf{A}}(\tau)\big) \cdot P(\tau) + \sum_{\tau \in \mathbb{T}_1} \big(\sigma'_{\mathsf{A}}(\tau) \cdot P_{\Gamma}^{min}(\tau) - \sigma_{\mathsf{A}}(\tau) \cdot P_{\Gamma}(\tau)\big) \qquad (6)$$

Note that all token prices in (6) are already defined in state Γ. Thus, A's gain can be minimized by considering only the effect on the token balance σ'_{A}, which we can rewrite as $\sigma_{\mathsf{A}} + \Delta_0 + \Delta_1 + \cdots$ where Δ_i is the effect on user A's balance induced by the i'th transaction in λ: this transaction is necessarily one initially authorized by A. We will show that Δ_i is *fixed* for any user transaction when executed in an inner solution layer: the position of an inner layer in solution λ does not affect its optimality.

The following theorem states that solutions to the MEV game can be constructed incrementally, by layering the local solutions for each individual transaction in the txpool. Intuitively, we choose a transaction T from $\mathcal{X}$, we solve the game for $(\Gamma, [\mathsf{T}])$, we compute the state Γ' obtained by executing this solution, and we inductively solve the game in the $(\Gamma', \mathcal{X}')$, where $\mathcal{X}'$ is $\mathcal{X}$ minus T.

Theorem 2. *With respect to the MEV game in $(\Gamma, \mathcal{X})$:*

1. *If $\mathcal{X}$ is empty, the solution is the final layer constructed for $(\Gamma, [])$ in §4.1.*
2. *Otherwise, if $\mathcal{X} = [\mathsf{T}] + \mathcal{X}'$, let (σ, λ) be the inner layer constructed for $(\Gamma, [\mathsf{T}])$, let $\mathsf{M}[\sigma] \mid \Gamma \xrightarrow{\lambda} \mathsf{M}[\_] \mid \Gamma'$, and let (σ', λ') be the solution for $(\Gamma', \mathcal{X}')$. Then, the solution to $(\Gamma, \mathcal{X})$ is $(\sigma + \sigma', \lambda\lambda')$.*

We now describe how to define the inner layers of the Dagwood sandwich, i.e. the base case of the inductive construction given by Theorem 2. Each inner layer includes a user transaction from the txpool, possibly front-run by M such that executing the layer leads the user's net worth to a local minimum. We define below the construction of these inner layers for each transaction type.

Swap inner layer Swap actions only affect the balance of *atomic* tokens. To minimize the gain of A after a swap, M must make A receive exactly the *minimum* amount of requested tokens. The effect of the swap on A's *atomic net worth* is:

$$W^0_{\mathsf{A}}(\Gamma') - W^0_{\mathsf{A}}(\Gamma) = -v_d \cdot P(\tau_d) + v_{1-d} \cdot P(\tau_{1-d}) \qquad \text{if } \Gamma \xrightarrow{\mathsf{A}:\mathsf{swap}^d(v_0:\tau_0, v_1:\tau_1)} \Gamma'$$

If the change in A's atomic net worth is negative, A's transaction is included in the layer. Although this transaction minimizes A's atomic net worth, it simultaneously affects the price of the minted token (τ_0, τ_1). This is not an issue, since the final layer of the Dagwood sandwich minimizes the prices of *all* minted tokens. Thus, the change of minted token prices due to the swap inner layer will *not* affect the user gain in the full Dagwood sandwich, as evident from (6). Note that the amount of tokens exchanged in a swap is chosen by the user, so the actual position of the layer in the Dagwood sandwich is immaterial (Theorem 2).

We now define the transaction used by M to front-run A's swap, ensuring that A receives the least amount of tokens from the swap. For $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$ and $\mathsf{T} = \mathsf{A} : \mathsf{swap}^{d_\mathsf{A}}(v_0 : \tau_0, v_1 : \tau_1)$, let the ***swap front-run reserves*** be:

$$\mathsf{SF}r_{d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \frac{\left| \sqrt{v_0^2 \cdot v_1^2 + 4 \cdot v_0 \cdot v_1 \cdot r_0 \cdot r_1} \right| - v_0 \cdot v_1}{2 \cdot v_{1-d_\mathsf{A}}}$$
$$\mathsf{SF}r_{1-d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \frac{r_0 \cdot r_1}{\mathsf{SF}r_{d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T})}$$

These values define the reserves of (τ_0, τ_1) in the state Γ' reached from $\mathsf{M}[\sigma] \mid \Gamma$ with M's transaction. Intuitively, if the swap front-run reserves do *not* coincide with the reserves r_0, r_1 in Γ, then M's transaction is needed to enable A's swap. We define the ***swap front-run direction*** d_M as:

$$d_\mathsf{M} = \begin{cases} d_\mathsf{A} & \text{if } \mathsf{SF}r_{d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) > r_{d_\mathsf{A}} \\ 1 - d_\mathsf{A} & \text{if } \mathsf{SF}r_{1-d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) > r_{1-d_\mathsf{A}} \end{cases}$$

We define the ***swap front-run values*** (i.e., the parameters of M's swap) as:

$$\begin{aligned} \mathsf{SF}w_{d_\mathsf{M}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) &= \begin{cases} \mathsf{SF}r_{d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) - r_{d_\mathsf{A}} & \text{if } d_\mathsf{M} = d_\mathsf{A} \\ r_{d_\mathsf{A}} - \mathsf{SF}r_{d_\mathsf{A}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) & \text{if } d_\mathsf{M} = 1 - d_\mathsf{A} \end{cases} \\ \mathsf{SF}w_{1-d_\mathsf{M}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) &= \begin{cases} r_{1-d_\mathsf{M}} - \mathsf{SF}r_{1-d_\mathsf{M}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) & \text{if } d_\mathsf{M} = d_\mathsf{A} \\ \mathsf{SF}r_{1-d_\mathsf{M}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) - r_{1-d_\mathsf{M}} & \text{if } d_\mathsf{M} = 1 - d_\mathsf{M} \end{cases} \end{aligned} \tag{7}$$

We combine these values to craft the ***swap front-run transaction***:

$$\mathsf{SFX}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \mathsf{M} : \mathsf{swap}^{d_\mathsf{M}}(\mathsf{SF}w_0(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_0, \mathsf{SF}w_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_1)$$

The inner layer is included in the Dagwood sandwich if it reduces A's net worth, i.e. if $-v_d \cdot P(\tau_d) + v_{1-d} \cdot P(\tau_{1-d}) < 0$. The swap front-run transaction is omitted if the reserves in Γ coincide with the swap front-run reserves. The balance of M in the (local) game solution is $\mathsf{SF}w_{d_\mathsf{M}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_{d_\mathsf{M}}$. Note that, the amount of tokens exchanged by the swapping user in (6) is fixed by $(-v_d, +v_{1-d})$, and the effect of a swap inner layer does not depend on its position along the Dagwood sandwich (Theorem 2).

Example 1. We recast our first example in §1 as a MEV game, assuming a txpool $\mathcal{X} = \{\mathsf{A} : \mathsf{swap}^0(40 : \tau_0, 35 : \tau_1)\}$. The initial state is $\Gamma = (100 : \tau_0, 100 : \tau_1) \mid \Gamma_w$, where Γ_w is made of user wallets, among which $\mathsf{A}[40 : \tau_0]$, and $P(\tau_0) = P(\tau_1) = 1,000$. We construct the Dagwood sandwich. Since A's swap yields a reduction in A's atomic net worth, $35 \cdot P(\tau_1) - 40 \cdot P(\tau_0) = -5,000$, then A's transaction is included in the inner layer. To check if A's swap must be front-run by M, we first compute the swap front-run reserves:

$$\mathsf{SF}r_0(\tau_0, \tau_1, \mathsf{T}, \Gamma) = \frac{\sqrt{40^2 \cdot 35^2 + 4 \cdot 40 \cdot 35 \cdot 100^2} - 40 \cdot 35}{2 \cdot 35} \approx 88.8$$
$$\mathsf{SF}r_1(\tau_0, \tau_1, \mathsf{T}, \Gamma) = \frac{100^2}{89} \approx 112.7$$

Since these values differ from the reserves in the initial game state, M must front-run A's transaction. The direction d_{M} of M's swap is 1, as $\mathsf{SF}r_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) > r_1$. The swap front-run values (7) are given by:

$$\mathsf{SF}w_0(\tau_0, \tau_1, \Gamma, \mathsf{T}) = 100 - 88.8 \approx 11.2 \quad \mathsf{SF}w_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) = 112.7 - 100 \approx 12.7$$

Therefore, the swap inner layer is made of two transactions:

$$\mathsf{M} : \mathsf{swap}^1(11.2 : \tau_0, 12.7 : \tau_1) \quad \mathsf{A} : \mathsf{swap}^0(40 : \tau_0, 35 : \tau_1)$$

and M's balance of the (local) game solution is $12.7 : \tau_1$. To construct the final layer, we consider the state $\Gamma'' = (128.8 : \tau_0, 77.7 : \tau_1) \mid \cdots$, shown in Figure 1. In Γ'', the canonical swap values are given by:

$$w_0^1(\tau_0, \tau_1, \Gamma'') = \frac{128.8 \cdot 22.3}{77.7 + 22.3} \approx 28.7$$
$$w_1^1(\tau_0, \tau_1, \Gamma'') = \sqrt{\tfrac{1}{1} \cdot 128.8 \cdot 77.7} - 77.7 \approx 22.3$$

Since $w_1^1(\tau_0, \tau_1, \Gamma'') > 1$, the direction d of the price minimization swap is 1. Therefore, the final layer is made of a single swap on the pair (τ_0, τ_1):

$$\mathsf{M} : \mathsf{swap}^1(28.7 : \tau_0, 22.3 : \tau_1))$$

where M's required balance is $22.3 : \tau_1$. Summing up, the Dagwood sandwich is constructed by appending the final layer to the inner layer, and M's required balance is $\sigma = 12.7 : \tau_1 + 22.3 : \tau_1 = 35 : \tau_1$. The MEV obtained by M through the Dagwood sandwich is $(11.2 - 12.7) \cdot 1,000 + (28.7 - 22.3) \cdot 1,000 \approx 5,000$. $\square$

Deposit inner layer By Lemma 2, deposits preserve the user's net worth. Thus, executing $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ in Γ does not bring any gain to A:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}) = -v_0 \cdot P(\tau_0) - v_1 \cdot P(\tau_1) + v \cdot P_\Gamma(\tau_0, \tau_1) = 0 \tag{8}$$

where v is the amount of minted tokens (τ_0, τ_1) given to A upon the deposit. By Lemma 4, $P_\Gamma(\tau_0, \tau_1) \geq P_\Gamma^{min}(\tau_0, \tau_1)$. By using this inequality in (8), we have:

$$\begin{aligned} & - v_0 \cdot P(\tau_0) - v_1 \cdot P(\tau_1) + v \cdot P_\Gamma^{min}(\tau_0, \tau_1) \leq 0 \\ \iff & \ v \cdot P_\Gamma^{min}(\tau_0, \tau_1) \leq v_0 \cdot P(\tau_0) + v_1 \cdot P(\tau_1) \end{aligned}$$

$$\begin{aligned} & \mathsf{M}[35 : \tau_1] \mid \Gamma = (100 : \tau_0, 100 : \tau_1) \mid \cdots \\ \xrightarrow{\mathsf{SFX}(\tau_0, \tau_1, \Gamma, \mathsf{T})} \ & \mathsf{M}[11.2 : \tau_0, 22.3 : \tau_1] \mid \Gamma' = (88.8 : \tau_0, 112.7 : \tau_1) \mid \cdots \\ \xrightarrow{\mathsf{T} = \mathsf{A}:\mathsf{swap}^0(40:\tau_0, 35:\tau_1)} \ & \mathsf{M}[11.2 : \tau_0, 22.3 : \tau_1] \mid \Gamma'' = (128.8 : \tau_0, 77.7 : \tau_1) \mid \cdots \\ \xrightarrow{\mathsf{X}(\tau_0, \tau_1, \Gamma'')} \ & \mathsf{M}[40 : \tau_0, 0 : \tau_1] \mid \Gamma''' = (100 : \tau_0, 100 : \tau_1) \mid \cdots \end{aligned}$$

Fig. 1. A Dagwood sandwich exploiting a single user swap.

By (6) it follows that including T in a game solution λ reduces A's net worth, since the decrease of A's net worth in atomic tokens is *not* always offset by the increase of net worth in minted tokens. Additionally, the minted token price $P_{\Gamma}(\tau_0, \tau_1)$ in (8) when the user deposit occurs is determined by deposit parameters v_0, v_1 alone: let $\Gamma \rightarrow^* \Gamma'$ be such that the given user deposit T is *enabled* in both Γ and Γ'. By [Dep], this implies $v_0/v_1 = r_0/r_1 = r'_0/r'_1$ where $(r_0, r_1) = res_{\tau_0,\tau_1}(\Gamma)$ and $(r'_0, r'_1) = res_{\tau_0,\tau_1}(\Gamma')$. Then, by Lemma 3, $P_{\Gamma}(\tau_0, \tau_1) = P_{\Gamma'}(\tau_0, \tau_1)$, as the reserve ratios in Γ and Γ' are equal. Thus, the amount of minted tokens v received by the depositing user in (6) is fixed by (v_0, v_1), and the effect of a deposit inner layer does not depend on its position along the Dagwood sandwich (Theorem 2).

Similarly to the construction of the swap inner layer, M may need to front-run transaction $\mathsf{T} = \mathsf{A} : \mathsf{dep}(v_0 : \tau_0, v_1 : \tau_1)$ to enable it. For $\Gamma = (r_0 : \tau_0, r_1 : \tau_1) \mid \cdots$, we define the ***deposit front-run reserves*** as:

$$\mathsf{DF}r_0(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \left| \sqrt{{}^{v_0}/_{v_1} \cdot r_0 \cdot r_1} \right| \qquad \mathsf{DF}r_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \left| \sqrt{{}^{v_1}/_{v_0} \cdot r_0 \cdot r_1} \right|$$

which satisfy $\mathsf{DF}r_0(\tau_0, \tau_1, \Gamma, \mathsf{T}) \cdot v_1 = \mathsf{DF}r_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) \cdot v_0$, as required by [Dep]. Given a swap direction d_{M}, we define the ***deposit front-run values*** as:

$$\mathsf{DF}w_{d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \mathsf{DF}r_{d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) - r_{d_{\mathsf{M}}}$$
$$\mathsf{DF}w_{1-d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = r_{1-d_{\mathsf{M}}} - \mathsf{DF}r_{1-d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T})$$

If $\mathsf{DF}w_{d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) > 0$ and $\mathsf{DF}w_{1-d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) > 0$ holds for a swap direction d_{M}, then we define the ***deposit front-run transaction*** as:

$$\mathsf{DFX}(\tau_0, \tau_1, \Gamma, \mathsf{T}) = \mathsf{M} : \mathsf{swap}^{d_{\mathsf{M}}}(\mathsf{DF}w_0(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_0, \mathsf{DF}w_1(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_1)$$

If the reserve ratio in the initial state does not coincide with the ratio of deposited funds, i.e. $v_0/v_1 \neq r_0/r_1$, then the deposit inner layer is $\mathsf{DFX}(\tau_0, \tau_1, \Gamma, \mathsf{T})\ \mathsf{T}$, and the balance required by M is $\mathsf{DF}w_{d_{\mathsf{M}}}(\tau_0, \tau_1, \Gamma, \mathsf{T}) : \tau_{d_{\mathsf{M}}}$. Otherwise, the deposit inner layer is made just by T, and the required balance is zero.

Redeem inner layer By Lemma 2, redeem actions preserve the user's net worth, i.e. A's gain is zero when firing $\mathsf{T} = \mathsf{A} : \mathsf{rdm}(v : (\tau_0, \tau_1))$ in Γ:

$$G_{\mathsf{A}}(\Gamma, \mathsf{T}) \;=\; -v \cdot P_{\Gamma}(\tau_0, \tau_1) + v_0 \cdot P(\tau_0) + v_1 \cdot P(\tau_1) \;=\; 0$$

Unlike for the deposit inner layer, redeem transactions *increase* the users' gain when executed in the game solution. This is apparent when substituting in the above equation $P_{\Gamma}(\tau_0, \tau_1) = P_{\Gamma}^{min}(\tau_0, \tau_1)$ (as per Lemma 4) to express the user gain *contribution* (6) of the redeem action.

$$-v \cdot P_{\Gamma}^{min}(\tau_0, \tau_1) + v_0 \cdot P(\tau_0) + v_1 \cdot P(\tau_1) \geq 0$$

Therefore, user redeem actions always *reduce* M's gain, and so they are *not* included in the solution. Therefore, the redeem inner layer is always empty.

$$
\begin{aligned}
& \mathsf{M}[18 : \tau_0, 50.5 : \tau_1] \mid \Gamma = (100 : \tau_0, 100 : \tau_1) \mid \cdots \\
\xrightarrow{\mathsf{SFX}(\tau_0,\tau_1,\Gamma,\mathsf{T})} & \mathsf{M}[29.3 : \tau_0, 37.8 : \tau_1] \mid \Gamma' = (88.8 : \tau_0, 112.7 : \tau_1) \mid \cdots \\
\xrightarrow{\mathsf{T}=\mathsf{A}:\mathsf{swap}^0(40:\tau_0,35:\tau_1)} & \mathsf{M}[29.3 : \tau_0, 37.8 : \tau_1] \mid \Gamma'' = (128.8 : \tau_0, 77.7 : \tau_1) \mid \cdots \\
\xrightarrow{\mathsf{DFX}(\tau_0,\tau_1,\Gamma'',\mathsf{T}')} & \mathsf{M}[71.4 : \tau_0, 0 : \tau_1] \mid \Gamma''' = (86.6 : \tau_0, 115.5 : \tau_1) \mid \cdots \\
\xrightarrow{\mathsf{T}'=\mathsf{A}:\mathsf{dep}(30:\tau_0,40:\tau_1)} & \mathsf{M}[71.4 : \tau_0, 0 : \tau_1] \mid \Gamma'''' = (116.6 : \tau_0, 155.5 : \tau_1) \mid \cdots \\
\xrightarrow{\mathsf{X}(\tau_0,\tau_1,\Gamma'''')} & \mathsf{M}[53.4 : \tau_0, 20.8 : \tau_1] \mid (134.6 : \tau_0, 134.6 : \tau_1) \mid \cdots
\end{aligned}
$$

Fig. 2. A Dagwood sandwich exploiting a user swap, deposit and redeem (dropped).

Example 2. We now recast the full example in Section 1 as a MEV game, considering all three user transactions in the txpool:

$$\mathcal{X} = \{\, \mathsf{A} : \mathsf{swap}^0(40 : \tau_0, 35 : \tau_1)\,,\ \mathsf{A} : \mathsf{dep}(30 : \tau_1, 40 : \tau_1)\,,\ \mathsf{A} : \mathsf{rdm}(10 : (\tau_0, \tau_1))\,\}$$

The game solution is shown in Figure 2: note that we can reuse the swap inner layer from Example 1, since the initial state and user swap action are identical. Thus, we continue by constructing the deposit inner layer for user deposit T' in state $\Gamma'' = (128.8 : \tau_0, 77.7 : \tau_1)$. Here, the deposit front-run reserves are:

$$
\begin{aligned}
\mathsf{DF}r_0(\tau_0, \tau_1, \Gamma'', \mathsf{T}') &= \left| \sqrt{{}^{30}\!/_{40} \cdot 128.8 \cdot 77.7} \right| = 86.6 \\
\mathsf{DF}r_1(\tau_0, \tau_1, \Gamma'', \mathsf{T}') &= \left| \sqrt{{}^{40}\!/_{30} \cdot 128.8 \cdot 77.7} \right| = 115.5
\end{aligned}
$$

Since the ratio of the deposit front-run reserves does not coincide with the reserve ratio in Γ'' ($86.6/115.5 \neq 128.8/77.7$), the deposit front-run by M is necessary to enable the user deposit action. By choosing a swap direction $d_{\mathsf{M}} = 1$, we obtain the positive deposit front-run values, which confirm the choice of the direction:

$$\mathsf{DF}w_0(\tau_0, \tau_1, \Gamma'', \mathsf{T}') = 128.8 - 86.6 \approx 42.2 \quad \mathsf{DF}w_1(\tau_0, \tau_1, \Gamma'', \mathsf{T}') = 115.5 - 77.7 \approx 37.8$$

Therefore, M's deposit front-run transaction is:

$$\mathsf{DFX}(\tau_0, \tau_1, \Gamma'', \mathsf{T}') \;=\; \mathsf{M} : \mathsf{swap}^1(42.2 : \tau_0, 37.8 : \tau_1)$$

which requires a balance $\sigma(\tau_1) \geq 37.8$. The deposit inner layer is obtained by prepending this transaction to A's deposit. The redeem inner layer is empty, as shown before. By (5), the final layer to minimize the price of minted tokens is:

$$\mathsf{M} : \mathsf{swap}^1(18.0 : \tau_0, 20.8 : \tau_1)$$

Summing up, the full Dagwood sandwich (see also Figure 2) is:

$$\mathsf{SFX}(\tau_0, \tau_1, \Gamma, \mathsf{T}) \;\; \mathsf{T} \;\; \mathsf{DFX}(\tau_0, \tau_1, \Gamma'', \mathsf{T}') \;\; \mathsf{T}' \;\; \mathsf{X}(\tau_0, \tau_1, \Gamma'''')$$

which requires an initial balance $\sigma = \{18.0 : \tau_0, 12.7 + 37.8 : \tau_1\}$ by M. By inspection of the Dagwood sandwich execution in Figure 2, it can be seen that the miner has obtained a gain of approximately 5,700. $\square$

5 Related work

Daian *et al.* [13] study the effect of transaction reordering obtained through *priority gas auctions*. These are games between users who compete to include a bundle of transactions in the next block, bidding on transaction fees to incentivize miners to include their own bundle. Notably, [13] finds empirical evidence of the fact that the gain derived from transaction reorderings in decentralized exchanges (DEX) exceeds the gain given by block rewards and transaction fees in Ethereum. The same work also proposes a game model of priority gas auctions, showing a Nash equilibrium for players to take turns bidding, compatibly with behavior observed in the wild on Ethereum. Our mining game differs from that in [13], since we assume a greedy adversary wanting to maximize its gain at the expense of all the other users, exploiting arbitrages on AMMs.

Zhou *et al.* [20] provide a theoretical framework to study the front-running on AMMs. Two sandwich heuristics are studied: the *front-run & back-run swap* sandwich, and the novel *front-run redeem & back-run swap and deposit*. The swap semantics used in [20] is simplified, compared to ours, since no minimum amount of received tokens is enforced by the AMM, users only perform swaps and hold no minted tokens (depositing and swapping agents are decoupled). Further, extractable value from arbitrage is considered separately. In comparison, we emphasize that we propose a solution to attack all main user action types offered by leading AMMs, thereby extracting value from user submitted swaps and deposits. Our model also accurately model minted tokens: their value is dynamically affected by miner and user swaps during the execution of the attack. Thus, our game solution extracts the maximum value in a more concrete setting, considering the victim transactions of both aforementioned attacks in [20], and leaving no arbitrage opportunities unexploited.

More general ordering and injection of transactions by a rational agent is generally referred to as *front-running*. Eskandari *et al.* [15] provide a taxonomy for various front-running attacks in blockchain applications and networks. This taxonomy is expanded in [16] with liquidations, sandwich attacks and arbitrage actions between DEX.

Some works investigate the problem of detecting front-running attacks on public blockchains. For example, in [16], Qin *et al.* introduce front-running detection heuristics which are deployed to empirically study the presence of such attacks on public DeFi applications. On the other hand, various fair ordering schemes have been proposed to mitigate front-running or exploitation of miner-extractable value. However, simple commit-and-reveal schemes still leak information such as account balances. Breidenbach *et al.* [11] propose "submarine commitments", which rely on k-anonymity to prevent any leaks from user commitments. Baum *et al.* [10] introduce a order-book based DEX which delegates the matching of orders to an out-sourced, off-chain multi-party computation committee. Private user orders are not revealed to other participants, such that no front-running can occur in each privately-computed order matching round. Ciampi *et al.* [12] introduce a market maker protocol in which the strictly sequential trade history between an off-chain market maker and traders are verifiable as a hash-chain. Any subsequent reordering by the AMM is publicly provable: collateral from the market maker incentivizes honest, fair-ordering behaviour. Such work aims to provide alternative, front-running resistant designs with AMM-like functionality. In contrast, our work is intended to formalize the behaviour of current, mainstream AMMs in the presence of a rational adversary.

The DeFi community is developing tools to enable agents to extract value from smart contracts: e.g., flashbots [2] is a project aiming to develop Ethereum implementations which support transaction bundles: Rather than front-running individual transactions by adjusting their fees, an agent can communicate a sequence or *bundle* of transactions to the miner, asking its inclusion in the next block. Our game solutions could be implemented to solve for such bundles.

6 Conclusions

We have addressed the problem of adversaries extracting value from AMMs interactions to the detriment of users. We have constructed an *optimal* strategy that adversaries can use to extract value from AMMs, focussing on the widespread class of constant-product AMMs. Our results apply to any adversary with the power to reorder, drop or insert transactions: besides miners, this includes *roll-up aggregators*, like e.g. Optimism and StarkWare [3,4]. Notably, our work shows that it is possible to extract value from *all* types of AMM transactions, while previous works focus on extracting value from token swaps, only.

In practice, value is also extracted from AMMs by colluding mining and non-mining agents: for the Ethereum blockchain, agents can send *transaction bundles* [2] to mining pools for block inclusion, in return for a fee. Our technique naturally applies to this setting, where the actions of the miner are simply replaced by actions by the agent submitting the transaction bundle.

We now discuss the simplifying assumptions (1-8) listed in Section 3. (1) User balances do not limit the order in which transactions in the *txpool* can be executed. In practice, in some cases it would be possible to perform a sequence of actions by exploiting the funds received from previous actions. We leave ordering constraints imposed by limited wallet balances for future work. (2) The adversary holds no minted tokens prior to executing the game solution. Yet, the adversary can exploit an (unbounded) initial balance of atomic tokens to acquire minted tokens as part of the game solution by performing deposits. The optimality of the Dagwood sandwich illustrates that this is not necessary. (3) The size of the *Dagwood sandwich* is unbounded. In practice, a typical block of transactions will include other transactions besides those directed to AMMs, and so the adversary can find enough space for its sandwiches by dropping non-AMM transactions. During times of block-congestion, a constraint on the length of the Dagwood sandwich will apply: we conjecture that solving such an optimization is NP-hard, and regard this as an relevant question for future work. (4) Prices of atomic tokens are fixed for the duration of the game: the Dagwood sandwich will need to be recomputed should prices change. (5) AMMs only hold atomic tokens. This is common in practice, but we note that extending the mining game to account for arbitrary nesting of minted tokens by AMM pairs is an interesting direction of future research. (6) No AMM swap fees and (7) no transaction fees are modelled: the adversary's gain resulting from the Dagwood sandwich is an upper bound to profitability as fees tend to zero. Yet, fees affect this gain, so they should be taken into account to construct an optimal strategy. Furthermore, transaction fees may make it convenient for a miner to include user redeem transactions in the sandwich, while these are never exploited by our strategy. (8) Besides fees, we abstract from the intervals that users can express to constrain the amount of tokens received upon deposits and redeems (we only model these constraints for swaps). This is left for future work.

In this paper we have considered AMMs which implement the constant-product swap invariant, like e.g. Uniswap and SushiSwap. A relevant research question is how to solve the MEV game under different swap invariants, e.g. those used by Curve Finance and SushiSwap. Uniform frameworks which address this problem have been proposed in [14,9] where swap invariants are abstracted as functions subject to a given set of constraints.

Acknowledgements Massimo Bartoletti is partially supported by Conv. Fondazione di Sardegna & Atenei Sardi project F75F21001220007 *ASTRID*. James Hsin-yu Chiang is supported by the PhD School of DTU Compute.

References

1. Curve statistics (2020), https://www.curve.fi/dailystats
2. Flashbots (2021), https://github.com/flashbots/pm
3. Optimism website (2021), https://optimism.io/
4. Starkware website (2021), https://starkware.co/
5. SushiSwap statistics (2021), https://analytics.sushi.com/
6. SushiSwap token pair implementation (2021), https://github.com/sushiswap/sushiswap/blob/94ea7712daaa13155dfab9786aacf69e24390147/contracts/uniswapv2/UniswapV2Pair.sol
7. Uniswap token pair implementation (2021), https://github.com/Uniswap/uniswap-v2-core/blob/4dd59067c76dea4a0e8e4bfdda41877a6b16dedc/contracts/UniswapV2Pair.sol
8. Uniswap V2 statistics (2021), https://v2.info.uniswap.org/
9. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: A theory of Automated Market Makers in DeFi. In: Coordination Models and Languages (COORDINATION). LNCS, vol. 12717, pp. 168–187. Springer (2021). https://doi.org/10.1007/978-3-030-78142-2_11
10. Baum, C., David, B., Frederiksen, T.K.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. In: Applied Cryptography and Network Security (ACNS). LNCS, vol. 12726, pp. 163–194. Springer (2021). https://doi.org/10.1007/978-3-030-78372-3_7

11. Breidenbach, L., Daian, P., Tramèr, F., Juels, A.: Enter the Hydra: Towards principled bug bounties and exploit-resistant smart contracts. In: USENIX Security Symposium. pp. 1335–1352. USENIX Association (2019)
12. Ciampi, M., Ishaq, M., Magdon-Ismail, M., Ostrovsky, R., Zikas, V.: FairMM: A fast and frontrunning-resistant crypto market-maker. Cryptology ePrint Archive, Report 2021/609 (2021), https://eprint.iacr.org/2021/609
13. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symp. on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
14. Engel, D., Herlihy, M.: Composing Networks of Automated Market Makers. In: Advances in Financial Technologies (AFT). p. 15–28. ACM (2021). https://doi.org/10.1145/3479722.3480987
15. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer (2020). https://doi.org/10.1007/978-3-030-43725-1_13
16. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? (2021), https://arxiv.org/abs/2101.05511
17. Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: SoK: Decentralized finance (DeFi). CoRR **abs/2101.08778** (2021)
18. Zhang, Y., Chen, X., Park, D.: Formal specification of constant product market maker model & implementation (2018), https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf
19. Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in DeFi protocols. In: IEEE Symp. on Security and Privacy. pp. 919–936. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00113
20. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. In: IEEE Symp. on Security and Privacy. pp. 428–445. IEEE (2021). https://doi.org/10.1109/SP40001.2021.00027

A Proofs

Lemma 1. $\sum_{\mathsf{A}} G_{\mathsf{A}}(\Gamma, \mathsf{T}) = 0$.

Proof. Follows from Lemma 3 (preservation of net worth) in [9].

Lemma 2. *If* $type(\mathsf{T}) \in \{\mathsf{dep}, \mathsf{rdm}\}$*, then* $G_{usr(\mathsf{T})}(\Gamma, \mathsf{T}) = 0$.

Proof. Follows from Lemma 3 (preservation of net worth) in [9].

Lemma 3. *Let* $\Gamma \xrightarrow{\lambda} \Gamma'$*, and let* $res_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1)$*,* $res_{\tau_0,\tau_1}(\Gamma') = (r'_0, r'_1)$*. Then,* $P_\Gamma(\tau_0, \tau_1) = P_{\Gamma'}(\tau_0, \tau_1)$ *if and only if* $r_0/r_1 = r'_0/r'_1$*.*

Proof. Let the *projected minted token price* of (τ_0, τ_1) at reserve ratio $R > 0$ in state Γ be defined as:

$$P^R_\Gamma(\tau_0, \tau_1) = \frac{r'_0}{sply_\Gamma(\tau_0, \tau_1)} \cdot P(\tau_0) + \frac{r'_1}{sply_\Gamma(\tau_0, \tau_1)} \cdot P(\tau_1)$$

where for the *projected reserves* (r'_0, r'_1), both $r'_0 \cdot r'_1 = r_0 \cdot r_1$ and $R = r'_0/r'_1$ hold. Thus, the projected minted token price can be rewritten entirely in terms of token reserves and supply in Γ and *projected ratio* R:

$$P^R_\Gamma(\tau_0, \tau_1) = \frac{\sqrt{r_0 \cdot r_1 \cdot R}}{sply_\Gamma(\tau_0, \tau_1)} \cdot P(\tau_0) + \frac{\sqrt{r_0 \cdot r_1 / R}}{sply_\Gamma(\tau_0, \tau_1)} \cdot P(\tau_1) \tag{9}$$

We note that from (9) and (1) it follows that

$$P^R_\Gamma(\tau_0, \tau_1) = P_\Gamma(\tau_0, \tau_1) \quad \text{if} \quad \begin{array}{l} res_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1) \\ R = r_0/r_1 \end{array} \tag{10}$$

Alternatively, the *projected minted token price* in a given state Γ can be interpreted as the minted token price in Γ' of execution $\mathsf{M}[\sigma] \mid \Gamma \to^{\mathsf{T}} \mathsf{M}[\_] \mid \Gamma'$ where T is a miner swap action and the reserve ratio $r'_0/r'_1 = R$ holds in Γ' but not in Γ. By definition then, there exists σ and swap T for any reachable state Γ and $R > 0$, such that $\mathsf{M}[\sigma] \mid \Gamma \to^{\mathsf{T}} \mathsf{M}[\_] \mid \Gamma'$ and $P^R_\Gamma(\tau_0, \tau_1) = P_{\Gamma'}(\tau_0, \tau_1)$ if $R \neq r_0/r_1$.

We prove Lemma 3 by showing that for any R, the *projected minted token price* of a pair remains *constant* for any execution. Thus, if in two states Γ, Γ' along an execution the AMM pair reserve ratios both equal $R = r_0/r_1 = r'_0/r'_1$, prices must also be equal, thereby proving the lemma.

$$P^R_\Gamma(\tau_0, \tau_1) = P^R_{\Gamma'}(\tau_0, \tau_1) = P_\Gamma(\tau_0, \tau_1) = P_{\Gamma'}(\tau_0, \tau_1) \tag{11}$$

We prove that the *projected minted token price* remains constant for any execution by induction.

Base case: empty For an empty step, the projected minted token price remains constant (trivially).

Induction step: deposit/redeem For a deposit or redeem execution $\Gamma_n \rightarrow^{\mathsf{T}} \Gamma_{n+1}$ the following must hold for $c > 0$ by definition of [Dep] and [Rdm]

$$(c \cdot r_0^n, c \cdot r_1^n) = (r_0^{n+1}, r_1^{n+1}) \qquad c \cdot \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1) = \mathit{sply}_{\Gamma_{n+1}}(\tau_0, \tau_1)$$

Thus, we can write the *projected minted token price* in Γ_{n+1} in terms of reserves and token supply in Γ_n, such that the equality is apparent.

$$\begin{aligned} P^R_{\Gamma_{n+1}}(\tau_0, \tau_1) &= \frac{\sqrt{c^2 \cdot r_0^n \cdot r_1^n \cdot R}}{c \cdot \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_0) + \frac{\sqrt{c^2 \cdot r_0^n \cdot r_1^n / R}}{c \cdot \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_1) \\ &= \frac{\sqrt{r_0^n \cdot r_1^n \cdot R}}{\mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_0) + \frac{\sqrt{r_0^n \cdot r_1^n / R}}{\mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_1) = P^R_{\Gamma_n}(\tau_0, \tau_1) \end{aligned}$$

Induction step: swap For a swap execution $\Gamma_n \rightarrow^{\mathsf{T}} \Gamma_{n+1}$ both the supply of minted tokens and the reserve product is maintained by definition of Swap

$$r_0^n \cdot r_1^n = r_0^{n+1} \cdot r_1^{n+1} \qquad \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1) = \mathit{sply}_{\Gamma_{n+1}}(\tau_0, \tau_1)$$

Again, we can express the *projected minted token price* in Γ_{n+1} in terms of reserves and token supply in Γ_n to illustrate the equality.

$$P^R_{\Gamma_{n+1}}(\tau_0, \tau_1) = \frac{\sqrt{r_0^n \cdot r_1^n \cdot R}}{\mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_0) + \frac{\sqrt{r_0^n \cdot r_1^n / R}}{\mathit{sply}_{\Gamma_n}(\tau_0, \tau_1)} \cdot P(\tau_1) = P^R_{\Gamma_n}(\tau_0, \tau_1)$$

Thus, we have shown that the projected minted token price remains constant for all executions. Therefore, (11) holds, proving the lemma. □

Lemma 4. *There exists a function P^{min} such that if $\mathsf{M}[\sigma] \mid \Gamma \rightarrow^* \mathsf{M}[\sigma'] \mid \Gamma'$ then: (i) $P_{\Gamma'}(\tau_0, \tau_1) \geq P^{min}_{\Gamma}(\tau_0, \tau_1)$; (ii) there exist σ'' and λ consisting at most of a swap by M such that $\mathsf{M}[\sigma''] \mid \Gamma' \xrightarrow{\lambda} \mathsf{M}[\_] \mid \Gamma''$ and $P_{\Gamma''}(\tau_0, \tau_1) = P^{min}_{\Gamma}(\tau_0, \tau_1)$.*

Proof. lma:price-minimum The proof reuses the definition of the *projected minted token price* (9) defined in the proof of Lemma 3: there, we showed that the *projected minted token price* for any given reserve ratio $R > 0$ remains constant for all executions. Thus, by definition (9), the projected minted token price in Γ for all $R > 0$ is the minted token price *range* which can be achieved by executing a swap in any reachable state Γ.

To find $P^{min}_{\Gamma}(\tau_0, \tau_1)$, we first determine the R for which $P^R_{\Gamma}(\tau_0, \tau_1)$ is minimized in any reachable state Γ.

$$\frac{\partial}{\partial R} P^R_{\Gamma}(\tau_0, \tau_1) = \frac{\sqrt{r_0^n \cdot r_1^n}}{2 \cdot \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{R}} \cdot P(\tau_0) - \frac{\sqrt{r_0^n \cdot r_1^n}}{2 \cdot \mathit{sply}_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{R} \cdot R} \cdot P(\tau_1)$$

Setting the expression above as equal to zero and then solving for $R = R^{min}$ we obtain

$$R^{min} = \frac{r_0}{r_1} = \frac{P(\tau_1)}{P(\tau_0)}$$

Further, we have determined the projected minted token price *minimum* since the second derivative is positive

$$\frac{\partial^2}{\partial R^2} P_\Gamma^R(\tau_0, \tau_1) = -\frac{\sqrt{r_0^n \cdot r_1^n}}{4 \cdot sply_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{R} \cdot R} + \frac{3 \cdot \sqrt{r_0^n \cdot r_1^n}}{4 \cdot sply_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{R} \cdot R^2}$$

$$= -\frac{\sqrt{r_0^n \cdot r_1^n}}{4 \cdot sply_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{\frac{P(\tau_1)}{P(\tau_0)}} \cdot \frac{P(\tau_1)}{P(\tau_0)^2}} + \frac{3 \cdot \sqrt{r_0^n \cdot r_1^n}}{4 \cdot sply_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{\frac{P(\tau_1)}{P(\tau_0)}} \cdot \frac{P(\tau_1)}{P(\tau_0)^2}}$$

$$= \frac{2 \cdot \sqrt{r_0^n \cdot r_1^n}}{4 \cdot sply_{\Gamma_n}(\tau_0, \tau_1) \cdot \sqrt{\frac{P(\tau_1)}{P(\tau_0)}} \cdot \frac{P(\tau_1)}{P(\tau_0)^2}} > 0$$

Thus, the function $P_\Gamma^{min}(\tau_0, \tau_1)$ is given as

$$P_\Gamma^{min}(\tau_0, \tau_1) = P_\Gamma^{P(\tau_1)/P(\tau_0)}(\tau_0, \tau_1)$$

By definition of the *project minted token price*, a swap exists such that the projected price for reserve ratio R is achieved in the resulting state if the reserve ratio in Γ is not equal to R. Otherwise the reserve ratio must equal R, and thus the empty step achieves the projected price trivially. We have shown that $P_\Gamma^R(\tau_0, \tau_1) \geq P_\Gamma^{min}(\tau_0, \tau_1)$ for any $R > 0$, thereby proving the lemma. □

Theorem 1. *Let* $\Gamma = \parallel_{i \in I}(r_{i,0} : \tau_{i,0}, r_{i,1} : \tau_{i,1}) \mid \Gamma_w$*, where* Γ_w *only contains wallets. For all* $j \in I$ *and* $d \in \{0, 1\}$*, let* $v_j^d = w_d^d(\tau_{j,0}, \tau_{j,1}, \Gamma)$*, and let:*

$$\sigma_j = \begin{cases} v_j^d : \tau_{j,d} & \text{if } v_j^d > 0 \\ 0 & \text{if } v_j^0, v_j^1 \leq 0 \end{cases} \qquad \lambda_j = \begin{cases} \mathsf{X}^d(\tau_{j,0}, \tau_{j,1}, \Gamma) & \text{if } v_j^d > 0 \\ \varepsilon & \text{if } v_j^0, v_j^1 \leq 0 \end{cases}$$

Then, $(\sigma_1 \cdots \sigma_n, \lambda_1 \cdots \lambda_n)$ *is a solution to the game* $(\Gamma, \mathcal{X})$ *for an empty* $\mathcal{X}$*.*

Proof. Theorem 1 states that the solution to $(\Gamma, [])$ can be greedily constructed from canonical swaps for each AMM pair in Γ, thereby minimizing the prices of all minted tokens and net worth of users whilst maximizing the gain for the miner.

We prove the lemma by showing that the *price minimization swap* (5) for a pair (τ_0, τ_1) minimizes the respective minted token price. Since all AMM actions affect single pair reserves only, the miner can minimize the minted token price in any order, thereby proving the lemma.

To prove that the *price minimization swap* minimizes the minted token price of a pair, we show that it updates the pair reserve ratio to $r_0/r_1 = P(\tau_1)/P(\tau_0)$, which, as shown in the proof of Lemma 4, minimizes the price for all executions.

Case: $d = 0$ We assume the canonical swap direction to be $d = 0$. By definition of the canonical swap values at page 8, we have:

$$w_0^0(\tau_0, \tau_1, \Gamma) = \sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1} - r_0$$

$$w_1^0(\tau_0,\tau_1,\Gamma) = \frac{r_1 \cdot w_0^0(\tau_0,\tau_1,\Gamma)}{r_0 + w_0^0(\tau_0,\tau_1,\Gamma)} = \frac{r_1 \cdot \sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1} - r_0 \cdot r_1}{\sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1}}$$

Further, the reserve product invariant must hold before and after the price minimization swap in direction $d = 0$. We show that this holds:

$$(r_0 + w_0^0(\tau_0,\tau_1,\Gamma)) \cdot (r_1 - w_1^0(\tau_0,\tau_1,\Gamma)) = \sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1} \cdot \frac{r_0 \cdot r_1}{\sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1}} = r_0 \cdot r_1$$

Finally, we can show that the resulting reserve ratio following the price minimization swap is indeed $P(\tau_1)/P(\tau_0)$, thereby minimizing the minted token price (see proof of Lemma 4).

$$\frac{r_0 + w_0^0(\tau_0,\tau_1,\Gamma)}{r_1 - w_1^0(\tau_0,\tau_1,\Gamma)} = \frac{\sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1}}{\frac{r_0 \cdot r_1}{\sqrt{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1}}} = \frac{\frac{P(\tau_1)}{P(\tau_0)} \cdot r_0 \cdot r_1}{r_0 \cdot r_1} = \frac{P(\tau_1)}{P(\tau_0)}$$

Case: $d = 1$ Follows similarly and is omitted for brevity. $\square$

Theorem 2. *With respect to the MEV game in* $(\Gamma, \mathcal{X})$*:*

1. *If* $\mathcal{X}$ *is empty, the solution is the final layer constructed for* $(\Gamma, [])$ *in §4.1.*
2. *Otherwise, if* $\mathcal{X} = [\mathsf{T}] + \mathcal{X}'$*, let* (σ, λ) *be the inner layer constructed for* $(\Gamma, [\mathsf{T}])$*, let* $\mathsf{M}[\sigma] \mid \Gamma \xrightarrow{\lambda} \mathsf{M}[\_] \mid \Gamma'$*, and let* (σ', λ') *be the solution for* $(\Gamma', \mathcal{X}')$*. Then, the solution to* $(\Gamma, \mathcal{X})$ *is* $(\sigma + \sigma', \lambda\lambda')$*.*

Proof. We restate the user gain (6) from the execution of a game solution following Lemma 4.

$$G_{\mathsf{A}}(\mathsf{M}[\sigma] \mid \mathsf{A}[\sigma_{\mathsf{A}}] \mid \Gamma, \lambda)$$
$$= \sum_{\tau \in \mathbb{T}_0} \sigma'_{\mathsf{A}}(\tau) \cdot P(\tau) - \sigma_{\mathsf{A}}(\tau) \cdot P(\tau) + \sum_{\tau \in \mathbb{T}_1} \sigma'_{\mathsf{A}}(\tau) \cdot P_{\Gamma}^{min}(\tau) - \sigma_{\mathsf{A}}(\tau) \cdot P_{\Gamma}(\tau)$$

Here, the prices are either of atomic ($P_{\Gamma}(\tau)$), or minted tokens ($P_{\Gamma}(\tau)$ and $P_{\Gamma}^{min}(\tau)$), all determined in the initial state Γ. Thus, the exploitation of individual user actions by the miner is decided on the action's effect the user *token balance* only.

We prove Theorem 2 by showing that the "inner layer" for each user action type are *optimal* when constructed in any order from the submitted user actions in $\mathcal{X}$.

Swap-inner-layer Firsty, we show that the *swap front-run* by the miner will always minimize the amount of tokens *received* by the user. Let $\mathsf{T} = \mathsf{A} : \mathsf{swap}^0(v_0 : \tau_0, v_1 : \tau_1)$ where $d_{\mathsf{A}} = 0$ and

$$\mathsf{M}[\_] \mid \Gamma \xrightarrow{\mathsf{SFX}(\tau_0,\tau_0,\Gamma,\mathsf{T})} \mathsf{M}[\_] \mid \Gamma' \xrightarrow{\mathsf{T}} \mathsf{M}[\_] \mid \Gamma''$$

If the execution of user swap T results in the *minimal* received output amount v_1 for A, then for $res_{\tau_0,\tau_1}(\Gamma) = (r_0, r_1)$, $res_{\tau_0,\tau_1}(\Gamma') = (r'_0, r'_1)$ and $res_{\tau_0,\tau_1}(\Gamma'') = (r'_0 + v_0, r'_1 - v_1)$ the reserve product invariant must hold by definition of [SWAP].

$$(r'_0 + v_0) \cdot (r'_1 - v_1) = r'_0 \cdot r'_1 = r_0 \cdot r_1$$

Solving for r'_0, we can rewrite as:

$$\begin{aligned} (r'_0 + v_0) \cdot (\tfrac{r_0 \cdot r_1}{r'_0} - v_1) &= r_0 \cdot r_1 \\ r_0 \cdot r_1 - v_0 \cdot r'_0 + \tfrac{v_0 \cdot r_0 \cdot r_1}{r'_0} - v_0 \cdot v_1 &= r_0 \cdot r_1 \\ v_1 \cdot {r'_0}^2 + v_0 \cdot v_1 \cdot r'_0 - v_0 \cdot r_0 \cdot r_1 &= 0 \end{aligned}$$

The determinant to the quadratic equation is

$$D = v_0^2 \cdot v_1^2 + 4 \cdot v_0 \cdot v_1 \cdot r_0 \cdot r_1$$

Thus we can solve for *positive* reserves r'_0, r'_1 in state Γ' expressed in terms of the swap parameters (v_0, v_1) and reserves r_0, r_1 in initial state Γ, which coincide with definitions of the *swap front-run reserves* for $d_{\mathsf{A}} = 0$ (the case $d_{\mathsf{A}} = 1$ is omitted for brevity).

$$r'_0 = \frac{-v_0 \cdot v_1 + \sqrt{v_0^2 \cdot v_1^2 + 4 \cdot v_0 \cdot v_1 \cdot r_0 \cdot r_1}}{2 \cdot v_1} \qquad r'_1 = \frac{r_0 \cdot r_1}{r'_0}$$

If $(r_0, r_1) = (r'_0, r'_1)$, then clearly no swap front-run is required. Otherwise, the direction of the swap front-run depends on the value of r'_0, r'_1. For $r'_0 > r_0$ and $r'_1 < r_0$, the swap-front run direction $d_{\mathsf{M}} = 0$ is implied. For $r'_0 > r_0$ and $r'_1 < r_0$, $d_{\mathsf{M}} = 1$. The *swap front-run values* (7) follow from the difference between initial and *swap front-run* reserves.

Since the swap front-run always enables the user swap such that the the minimum output amount is returned, this implies that the *effect* on the user token balance when executing the solution (6) is solely determined by user swap parameters (v_0, v_1): it is not affected by its position in the full solution, enabling the greedy construction of the swap-inner-layer in Theorem 2.

The optimality of the swap-inner-layer can be easily shown: For our assumed user swap direction $d_{\mathsf{A}} = 0$, if $-v_0 \cdot P(\tau_0) + v_1 \cdot P(\tau_1) < 0$ holds, then the contribution to the user gain (6) must be negative, and furthermore, since by definition of [SWAP], v_1 is the minimum amount the user can receive, the swap-inner-layer must be optimal.

If $-v_0 \cdot P(\tau_0) + v_1 \cdot P(\tau_1) \geq 0$, then the swap-inner-layer will be $(0, \varepsilon)$, since there the user swap can never reduce the user gain in any game solution. We omit the case $d_{\mathsf{A}} = 1$ for brevity.

Deposit-inner-layer The optimality of the deposit-inner-layer follows from Section 4.2.

Redeem-inner-layer The optimality of the redeem-inner-layer $(0, \varepsilon)$ follows from Section 4.2. □

SoK: Mitigation of Front-running in Decentralized Finance

Publication Information

Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. "SoK: Mitigation of Front-running in Decentralized Finance." To appear in *Financial Cryptography and Data Security. FC 2022 International Workshops: AMHIS, CoDecFin, DeFi, VOTING, and WTSC, Grenada, May 6, 2022.*

Contribution

- Co-author.

Remarks

Accepted and presented at conference workshop. Proceedings by publisher are work-in-progress.

SoK: Mitigation of Front-running in Decentralized Finance

Carsten Baum[1], James Hsin-yu Chiang[2⋆], Bernardo David[3⋆⋆], Tore Kasper Frederiksen[4⋆⋆⋆], Lorenzo Gentile[3†],

[1] Aarhus University, Denmark
cbaum@cs.au.dk
[2] Technical University of Denmark, Denmark
jchi@dtu.dk
[3] IT University of Copenhagen, Denmark
bernardo@bmdavid.com, lorg@itu.dk
[4] Alexandria Institute, Denmark
tore.frederiksen@alexandra.dk

Abstract. Front-running is the malicious, and often illegal, act of both manipulating the order of pending trades and injecting additional trades to make a profit at the cost of other users. In decentralized finance (DeFi), front-running strategies exploit both public knowledge of user trades from transactions pending on the network and the miner's ability to determine the final transaction order. Given the financial loss and increased transaction load resulting from adversarial front-running in decentralized finance, novel cryptographic protocols have been proposed to mitigate such attacks in the permission-less blockchain setting. We systematize and discuss the state-of-the-art of front-running mitigation in decentralized finance, and illustrate remaining attacks and open challenges.

1 Introduction

labelsec:introduction
Specific instances of front-running in decentralized finance (DeFi) were first quantified by Daian et al. [18] and systematized by Eskandari et al. [23]. Besides imposing a financial penalty on honest users, front-running can also degrade the performance of blockchain networks, as recently observed on the Avalanche

⋆ This work was supported by the PhD School of DTU Compute.
⋆⋆ This work was supported by the Concordium Foundation and by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.
⋆⋆⋆ This work was supported by "Sikker brug af følsomme data", Performance Contract 2020 and "Digital sikkerhed, tillid og dataetik", Performance Contract 2021-2024, Ministry of Higher Education and Science, Denmark".
† This work was supported by the Concordium Foundation.

blockchain [3]. In order to evaluate the efficacy of front-running mitigation techniques, we first formulate the set of adversarial powers which permit front-running strategies to be exploited: concretely, if users submit their intended interaction to a pool of pending transactions, the front-running adversary has the ability to:

1. Append pending transactions to the blockchain.
2. Infer user intentions from pending transactions and blockchain state.

In this work, we describe common **front-running attacks** (§2) and assess three front-running **mitigation categories** (§3) for their isolated and combined efficacy in neutralizing front-running (Figure 1). We introduce a speculative sandwich attack on input batching techniques (§3.2), which can be mitigated with private user balances and secret input stores (§3.3).

Adversarial power	§3 **Mitigation**	
1. Transaction sequencing	§3.1 Fair ordering	
1. Transaction sequencing	§3.2 Batching of blinded inputs	Commit & reveal
2. Inference of user intent	§3.2 Batching of blinded inputs	Input aggregation
2. Inference of user intent	§3.3 Private user balances & secret input store	

Fig. 1. Overview of mitigation techniques

Fair ordering (§3.1), implemented at the consensus protocol layer, ensures that the local receipt-order of gossiped transactions seen by a node is consistent with the final transaction ordering in the blockchain. We observe that *fair ordering* effectively mitigates the miner's ability to freely sequence transactions, but introduces a front-running adversary which rushes the network.

		User balance & input store	
		Public	**Private, secret**
Batching of blinded inputs	Commit & reveal	Speculative Sandwich Attacks	Taint of user balances
	Input aggregation	Speculative Sandwich Attacks	-

Fig. 2. Efficacy: batching of blinded inputs.

Batching of blinded inputs (§3.2) replaces the *sequential* model of DeFi interaction with a *round-based* one, where user inputs are blinded in each round to ensure input independence, thereby thwarting front-running strategies that rely on prior knowledge of other users' intentions. However, if user balances are public, the input may still be partially inferred when the valid user's input space is constrained by its balance: here, we contribute a novel, *speculative* front-running attack that exploits the *direction* of an automated market maker (AMM) swap, leaked from the victim's public balance. Furthermore, we highlight differences between *commit & reveal* and *input aggregation* approaches to batching of blinded inputs (Figure 2). In commit & reveal schemes, user inputs are revealed *individually*: Although front-running in the specific round is no longer possible, they necessarily leak information about the subsequent balance-update for each participating user, even if the user balances are private. If the taint of private balances is sufficiently strong, this can allow the front-running adversary to infer the users future inputs (e.g. the intended AMM swap direction).

Private user balances (§3.3) are thus necessary to prevent the leakage of the valid user input space from balances and application state. Although DeFi state must generally remain public to retain its utility [2], we show that it is necessary to shield certain fragments thereof which explicitly reveal future user intent. **Secret input stores** (§3.3) protect inputs that are evaluated by the application after a time delay [46] or, in the case of order books, whenever a match with other user inputs [24,7] can be found.

2 Front-running attacks

AMM sandwich: We briefly summarize the functionality of constant product AMM's, namely, a liquidity pool holding token balances, r_0 and r_1, of two different token types, τ_0 and τ_1 respectively, s.t. $r_0 \cdot r_1$ is *always* constant when swaps are being carried out between τ_0 and τ_1. A user swaps units of τ_0 for units of τ_1 by authorizing a *left swap* action $\mathsf{SL}(v : \tau_0, w : \tau_1)$. Here, the user is sending $v : \tau_0$ to the AMM in return for at least $w : \tau_1$ (swap limit). For this left swap to be valid, the product of the reserves must be maintained. Thus, the following relation between initial and updated reserves must hold: $r_0 \cdot r_1 = (r_0 + v) \cdot (r_1 - w')$, where $w' \geq w$ and w' represents the units of τ_1 that the user actually gets. We refer w as the swap *limit*. A *right swap* of $\mathsf{SR}(v : \tau_0, w : \tau_1)$ follows similarly: the user sends $w : \tau_1$ for at least $v : \tau_0$ in return such that $r_0 \cdot r_1 = (r_0 - v') \cdot (r_1 + w)$ and $v' \geq v$ where v' represents the units of τ_0 received. Constant product AMM's exhibit *slippage*: subsequent swaps in the same direction exhibit decreasing exchange rates.

User swaps can be "sandwiched", exploiting slippage for the gain of the attacker. Consider a left swap $\mathsf{A} : \mathsf{SL}(\mathsf{v_A} : \tau_0, \mathsf{w_A} : \tau_1)$ submitted by user A. A front-run swap by attacker M in the same direction reduces the exchange rate for the subsequent victim swap: a final back-run swap by M in the opposing direction then profits from an improved exchange rate.

$$\mathsf{M} : \mathsf{SL}(\mathsf{v}^\mathsf{f}_\mathsf{M} : \tau_0, \mathsf{w}^\mathsf{f}_\mathsf{M} : \tau_1) \quad \mathsf{A} : \mathsf{SL}(\mathsf{v_A} : \tau_0, \mathsf{w_A} : \tau_1) \quad \mathsf{M} : \mathsf{SR}(\mathsf{v}^\mathsf{b}_\mathsf{M} : \tau_0, \mathsf{w}^\mathsf{b}_\mathsf{M} : \tau_1)$$

Optimal front-run $(\mathsf{v}^\mathsf{f}_\mathsf{M}, \mathsf{w}^\mathsf{f}_\mathsf{M})$ and back-run $(\mathsf{v}^\mathsf{b}_\mathsf{M}, \mathsf{w}^\mathsf{b}_\mathsf{M})$ parameters are a function of the victim's swap, inferred from the pending victim transaction gossiped across the network [5].

We illustrate a step-wise execution of a sandwich in Figure 3 and introduce notation for user and AMM state proposed in [4] for this purpose. The wallet of A is modelled as the term $\mathsf{A}[v_i : \tau_0, ..., v_n : \tau_n]$, where $v_0, ..., v_n$ are the respective balances of token types $\tau_0, ..., \tau_n$. The state of an AMM holding token types τ_0 and τ_1 is given by its reserve balances $(r_0 : \tau_0, r_1 : \tau_1)$. Thus, we express the system state as a composition of wallets and reserve balances.

$$\mathsf{A}[v : \tau] \mid (r_0 : \tau_0, r_1 : \tau_1)$$

Let the initial AMM balance be $(100 : \tau_0, 100 : \tau_1)$. User A wishes to perform the swap $\mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1)$. For simplicity, we assume unit values of τ_0 and τ_1 to be equal: given the ratio of AMM reserves is 1, there is no arbitrage opportunity to be exploited [4]. If A's order is executed immediately, A receives $13 : \tau_1$ for the $15 : \tau_0$ it sends to the AMM. Instead, however, if the user swap is sandwiched by attacker M (Figure 3), A only obtains the minimum amount $10 : \tau_1$, implying a reduction of $3 : \tau_1$. Note that the reserve product

$$\begin{aligned}
& \mathsf{A}[15 : \tau_0] \mid \mathsf{M}[15 : \tau_0, 10 : \tau_1] \mid (100 : \tau_0, 100 : \tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(15:\tau_0,13:\tau_1)} \; & \mathsf{A}[15 : \tau_0] \mid \mathsf{M}[23 : \tau_1] \mid (115 : \tau_0, 87 : \tau_1) \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} \; & \mathsf{A}[10 : \tau_1] \mid \mathsf{M}[23 : \tau_1] \mid (130 : \tau_0, 77 : \tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(30:\tau_0,23:\tau_1)} \; & \mathsf{A}[10 : \tau_1] \mid \mathsf{M}[30 : \tau_0] \mid (100 : \tau_0, 100 : \tau_1)
\end{aligned}$$

Fig. 3. Sandwich attack

is maintained at each execution step and that the sandwich execution preserves the initial reserve ratio: the attack leaves no arbitrage opportunity unexploited. The attacker M's profit of 5 units of τ_0 (or τ_1) is optimal [5]: A receives the minimum amount possible, namely its swap limit.

Scheduled AMM sandwich: For certain AMM variants, the knowledge of the user's intent to perform a swap can be directly inferred from the blockchain state. Paradigm [46] propose scheduled AMM swaps, or more generally, *scheduled inputs*. Let $\mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1, \mathsf{r})$ be a swap that is not executed immediately, but scheduled for evaluation together with the first user-AMM interaction *following* blockchain round r, thus requiring no further interaction from A. Since *scheduled* orders are stored in the AMM smart contract and evaluated at the beginning of a known round, the sandwich attack strategy can be exploited, albeit over two block rounds [46]: the front-run is sequenced at the end of round r and the back-run as the first newly submitted swap of round $\mathsf{r} + 1$.

Generalized front-run attacks: In decentralized finance, actions exist which are *profitable* for the authorizing user, but which can also be performed by any other agent with a sufficient balance. In the permissionless blockchain setting, *generalized front-runners*, a term coined by Daian [38], are automated agents that identify profitable, pending transactions, which can be authorized by *any* user, and simply replicate these with their own account, thereby depriving the original transaction submitter of it's profit. Since the security of DeFi applications rely on rational agents to solve for profitable arbitrage [48,45,21] and liquidation [41] strategies, the presence of generalized front-running threatens to restrict such opportunities to agents colluding with miners.

3 Mitigation categories

3.1 Fair ordering

A recent line of research [34,30,31] has formalized an intuitive notion of γ-*receipt-order-fairness*: given two distinct transactions $\texttt{tx}$ and $\texttt{tx}'$ broadcast by users, receipt-order-fairness of a consensus protocol ensures that $\texttt{tx}$ will be finalized prior to $\texttt{tx}'$ if a γ fraction of network nodes receives $\texttt{tx}$ prior to $\texttt{tx}'$. However, Kelkar et al. [30] show that even if all nodes agree on the relative order in which any *pair* of transactions were first observed at the gossip stage, a global transaction ordering of all transactions consistent with the local view of pair-wise orderings is not always possible (Condorcet Paradox). Instead, a weaker notion of γ-*batch-order-fairness* is realized in [31], where $\texttt{tx}$ will be sequenced prior to or in the same block as $\texttt{tx}'$ if a γ node fraction receives $\texttt{tx}$ first.

Front-running despite fair ordering: Although order fairness removes the miner or round leader's privilege to sequence transactions, it assumes that users have secure channels to servers participating in consensus: in practice, however, public blockchains rely on gossip networks to propagate pending transactions. Here, the *rushing* network adversary can control the receipt-order of transactions for each consensus node, thereby rendering the notion of γ-*batch-order-fairness* meaningless. In practice, such a network adversary model may be excessively strong: whereas in the standard setting the miner or round leader incurs no additional cost for front-running victims, a non-trivial communication cost is now imposed on the rushing adversary. Still, since order-fairness clearly cannot *eliminate* front-running attacks in the (realistic) gossip-network setting, the motivation for stronger front-running mitigation properties remains.

3.2 Batching of blinded inputs

Batching of blinded inputs is a technique to ensure 1) the independence between user inputs and 2) the prevention of any adversarial sequencing of inputs. Interactions occur in rounds: in each, inputs are committed during the *input-phase*, followed by an *output phase* where the application state is updated after evaluating user inputs with valid parameters. The *collection* of inputs can occur in a smart contract or by a committee executing a cryptographic protocol which authorizes the distribution of funds from a smart contract in the output phase. The update of the application state following each round can result from the evaluation of valid inputs in *randomized* order or an application-specific *aggregation* thereof: for example, a subset of submitted AMM swaps can be aggregated into a single resulting swap. In batching of blinded inputs, we distinguish between **commit & reveal** and **input aggregation** (fig. 4). Both schemes commit inputs in the input-phase of each round, thereby ensuring input independence. However, while input aggregation keeps the users' input private indefinitely, commit & reveal schemes leak individual user inputs when commitments are

		Input independence	Input privacy	Open challenges
Commit & reveal	Hash commitments*	-	-	*Output bias*
	Timed commitments*	●	-	*Delay parameters*
	Threshold encryption**	●	-	*Honest majority*
	Secure multi-party computation**	●	-	*Honest majority*
Input aggregation		●	●	*Abort penalty*
	Homomorphic encryption**	●	●	*Efficiency*

Fig. 4. Batching of blinded inputs sent to a smart contract* or committee**

opened, thereby offering no *input privacy* by definition. Input privacy is necessary to prevent front-running in *subsequent* interaction rounds: past inputs leak information about updates to private balances (§3.3), which in turn can be exploited by front-runners, as balances constrain the valid user input space.

$$\text{Past user inputs} \xrightarrow{\text{reveal}} \text{Private user balances} \xrightarrow{\text{reveal}} \text{Future user inputs}$$

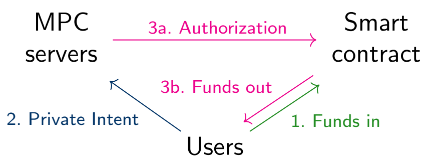
In contrast, input aggregation only outputs the application state update: for aggregated AMM swaps, only reserve updates are revealed, and updates to user balances remain private, if private balances are supported. Naturally, input aggregation can only offer input privacy up to the input batch size.

Commit & reveal: Although *hash commitments* collected by a smart contract may appear to be an obvious approach to implement the commit & reveal functionality, they suffer from *output bias*, as the adversary can selectively refrain from opening its commitment.

Time-lock puzzles [42] or *timed commitments* [11] generated by users and sent to a smart contract promise to eliminate output bias, since the adversary's commitment can be force-opened after a delay, guaranteeing the inclusion of its input in the output-phase. However, in the worst case, each user time-locked input must be solved separately by a constant number of squaring operations in a randomly sampled group, potentially rendering the approach impractical for larger batches of time-locked inputs [32]. Burdges and De Feo [14] propose a novel *delay encryption* notion and construction, which promises encryption of many inputs to a randomly sampled *session key*. Thus, all delay-encrypted inputs of a given batch can be decrypted after a single extraction process. Delay encryption [14] is constructed from isogeny-based cryptography, a recent and less-well studied class cryptographic assumptions. Finally, it remains an open challenge to match delay cryptography parameters to real-world delays which depend on assumed gate speeds used in practice.

Threshold encryption [22] can realize a commit & reveal scheme with the assumption of an *honest majority* committee holding trapdoor information of the encrypted inputs [44]. In each round, a key pair is produced by the execution of a *distributed key generation* (DKG) protocol and the public is opened, with which users encrypt their inputs in the given round. A subsequent opening of the corresponding secret key by the threshold committee enables the decryption of all inputs of the given round. However, should an encrypted user input fail to be finalized in the block-chain in a given round due to network congestion, the user's intent will be made public after the secret key is revealed for the given round without the user action being executed. Given this leakage, the front-running adversary may now anticipate the re-submission of the same user input in the next round.

Secure multi-party computation [47,27] (MPC) has been proposed [36,1] to realize a commit & reveal functionality with guaranteed input reveal in an anonymous fashion, also formalized as *anonymous committed broadcast* (ACB) in [1]. The anonymization of inputs is achieved by random *shuffling* of user inputs in an efficient manner. Here, *honest majority* MPC protocols [8,19] are favoured, as the output is guaranteed as long as the honest majority assumption holds true. To implement a DeFi application with MPC, an MPC-controlled smart contract is required, to which users send their funds prior to each round.



In the output phase of each MPC round, funds in the smart contract are redistributed to users according to the output(s) of the MPC execution. In practice, users can safely delegate the MPC execution to a group of servers [1].

Input aggregation: Naturally, MPC can realize any aggregation function over private user inputs, and in some instances in an efficient manner. Given the emphasis on the privacy of inputs, *dishonest majority* MPC protocols [15,10,20] are favoured, which ensure that private inputs can never be obtained by the adversary as long as a single participant remains honest. Informal proposals to implement AMM instances in a dishonest majority MPC have been proposed by Li et al. [35]. Although dishonest majority MPC can be aborted by a single dishonest party, a recent line of research [33,6,7] has realized an efficient set of protocols that identify and financially punish the aborting adversary. This achieves a weaker notion of fairness as the rational adversary is incentivized to never abort. Still, the penalty must exceed the financial *option* value of aborting in order to be effective: given that inputs are private, it remains an open research question on how to size financial penalties for identifiable abort in MPC.

Penumbra [40] proposes the use of *homomorpic encryption* to realize the secure aggregation of homomorphically encrypted AMM swap orders. The aggregated swap is then decrypted to reveal the updated AMM reserves. User balances are implemented with private coins (see §3.3), thus the privacy of the inputs are only dependent on the batch size. We note the non-trivial complexity of aggregating a batch of encrypted AMM swaps with swap limit constraints: *efficient* secure multi-party computation with fully homomorphic encryption schemes remains an open research problem [26]. In [40], consensus validators are proposed to perform the secure computation, consolidating MPC and consensus layers.

Speculative sandwich w/public user balances: We illustrate that batching of blinded inputs alone is not sufficient to prevent front-running attacks. Instead, speculative AMM sandwich attacks are possible in blinded input batching schemes as long as the direction of the victim swap is known by the adversary. This can be inferred from *public* user balances, as detailed in the subsequent example. Such speculative sandwich attacks on batched inputs also assume that the adversary in the permissionless setting can "isolate" a single victim's input in a given round, such that only front-run and victim transactions remain: we argue that each batching round has participant limits due to gas constraints or number of clients that MPC servers can support. Thus, the adversary can occupy any arbitrary number of user slots per round and provide invalid inputs[5] on slots not dedicated to the front-running swap.

Round r		Round r+1
$\mathsf{M} : \mathsf{SL}(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1)$	$\mathsf{A} : \mathsf{SL}(\mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}_{\mathsf{A}} : \tau_1)$	$\mathsf{M} : \mathsf{SR}(\mathsf{v}^{\mathsf{b}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} : \tau_1)$

Fig. 5. Speculative sandwich

In this speculative attack, we assume that private AMM swaps in each blinded input batch are evaluated in a *random* order, as proposed in [35,1]. The front-running M can only speculate on achieving the correct order to execute the sandwich. Since balances are public, M can observe that A's balance of τ_1 is zero: thus, A's submitted swap to the AMM (τ_0, τ_1) must be in the *left* direction. M submits the *front-run* swap in the same direction as the victim in the initial round r.

In the optimistic case shown in Figure 5, M's front-run swap is evaluated *prior* to the victim swap (in round r), thus enabling M to position the profitable back-run swap in round $\mathsf{r}+1$, where all other users are prevented from submitting inputs. M's front-run parameters can be chosen such that the front-run swap simply does not execute should the front-run *not* be ordered prior to the victim swap in round r, thereby aborting the attack. We refer to Appendix A for the proof that this speculative sandwich is rational for the attacker.

An execution of a speculative sandwich is shown in Figures 6 and 7: here, adversary M observes victim A's interaction with an AMM which batches blinded inputs. A has a public balance of $20 : \tau_0$ only, allowing M to infer that A can only perform a *left* swap from τ_0 to τ_1 with an input amount of at most $20 : \tau_0$. The attack strategy is executed over two subsequent rounds beginning in the initial state shown in Figure 6, where we assume unit values of τ_0 and τ_1 are equal.

In the first round r, M submits the *front-run* swap in the same direction as the victim's, with *arbitrarily chosen* input amount $7 : \tau_0$. The minimum output amount or swap limit of the front-run is then is chosen to be $6.5 : \tau_1$ such that $(100 + 7) \cdot (100 - 6.5) = 100^2$ holds: thus, if the front-run were executed in the

[5] e.g. AMM swap parameters which cannot be executed in the current AMM state.

$$\begin{array}{rl}
 & \mathsf{A}[20:\tau_0] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (100:\tau_0, 100:\tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(7:\tau_0,6.5:\tau_1)} & \mathsf{A}[20:\tau_0] \mid \mathsf{M}[21.5:\tau_1] \mid (107:\tau_0, 93.5:\tau_1) \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} & \mathsf{A}[5:\tau_0, 11.5:\tau_1] \mid \mathsf{M}[21.5:\tau_1] \mid (122:\tau_0, 82:\tau_1) \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(22:\tau_0,18:\tau_1)} & \mathsf{A}[5:\tau_0, 11.5:\tau_1] \mid \mathsf{M}[22:\tau_0, 3.5:\tau_1] \mid (100:\tau_0, 100:\tau_1)
\end{array}$$

Fig. 6. Successful speculative sandwich

initial state, M would receive *exactly* its swap limit. Since all other user orders (other than the victim swap of A) are suppressed, there is a probability of 0.5 that the front-run is randomly evaluated *before* the victim's swap, as shown in Figure 6. The *back-run* swap of M in the opposing direction then follows in the subsequent round with probability 1, since M suppresses all user actions other than its own back-run. Assuming equal unit value of both token types, the attack profit for M is 3.5.

Should the front-run ordering fail (Figure 7), then M's front-run parameters are chosen such that the front-run swap will not execute, resulting in an abort of the speculative sandwich attack. This is due to the chosen front-run parameters: following the execution step of A's swap in Figure 7, the constant product invariant can only hold if M receives $5:\tau_1$ for the $7:\tau_0$ it sends: $(115+7)\times(87-5)=100^2$. However, this contradicts M swap limit of $6.5:\tau_1$, such that the front-run cannot execute in the state following A's swap. M can still perform a back-run in round $\mathsf{r}+1$, thereby restoring the initial reserve ratio and extracting an arbitrage profit of 2, which is less than in the successful speculative sandwich execution in Figure 6. Still, the speculative sandwich attack is always profitable, as shown in Appendix A.

$$\begin{array}{rl}
 & \mathsf{A}[20:\tau_0] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (100:\tau_0, 100:\tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} & \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (115:\tau_0, 87:\tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(\cancel{7:\tau_0,6.5:\tau_1})} & \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (115:\tau_0, 87:\tau_1) \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(15:\tau_0,13:\tau_1)} & \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[22:\tau_0, 2:\tau_1] \mid (100:\tau_0, 100:\tau_1)
\end{array}$$

Fig. 7. Aborted speculative sandwich

Importantly, if victim A's swap direction were unknown, M would have to guess the direction of the front-running swap. An incorrect guess can result in a loss for M as shown in Appendix B. Thus, we argue that private user balances are necessary for batching of blinded inputs to be effective. Furthermore, for *scheduled* AMM orders introduced in [46], private user balances remain insufficient if scheduled orders are stored in public smart contracts: we sketch a speculative sandwich attack on publicly scheduled swaps in Appendix C. Finally, we note that hash-based commit & reveal schemes permit speculative sandwich attacks even when user balances are private, as the adversary can selectively reveal the appropriate sandwich strategy which matches on the swap first revealed by the victim (Appendix D).

3.3 Private & secret state

As argued in §3.2, both the *aggregation* of blinded inputs and use of *private balances and secret input stores* is necessary to mitigate front-running in the current and future rounds. Whilst it may be possible to maintain the *entire* DeFi application state secretly in an MPC instance in order to prevent front-running, this will naturally reduce its utility to users in the permissionless setting. Notably, Angeris et al. [2,16] argue that both *marginal price* and *validity* of a given AMM swap order must be queryable for an AMM interaction to be meaningful. Therefore, we restrict our study of secret state in DeFi applications to *user input stores* [46,24], which maintain submitted inputs until they are evaluated or executed at a later point in time.

Private user balances: Private block-chain currencies and tokens have been realized with zero-knowledge proof systems: *confidential transactions* [37] shield output amounts with efficient zero-knowledge *range proofs* [13], thereby ensuring that newly created output values do not exceed those spent by the same transaction. Confidential transactions only shield output amounts: a transaction graph connecting outputs can still be inferred from public transactions on the block-chain, permitting coin taint to propagate downstream.

Z-cash [43] style *decentralized anonymous payment* (DAP) schemes break such public links between outputs, as well-formed relations between new and spent outputs are not revealed but publicly verifiable with SNARK [28,25,39,9,29] zero-knowledge proofs. DAP schemes have also been proposed for DeFi functionality in Manta [17], but here front-running is not mitigated, since the AMM reserve state is public and swap inputs are not batched. Even though swap parameters are blinded in Manta, each individual swap execution results in a *public* update of AMM reserves. Thus, the *affect* of each swap on the current AMM reserves is known, leaking exchanged amounts and permitting sandwich attack strategies.

Importantly, when implementing input batching (Figure 4) with secure computation *and* block-chains supporting private user balances, zero-knowledge proofs must be generated inside the MPC instance in order to update private user balances. Doing so *efficiently* in MPC or even fully homomorphic encryption remains on open research question.

Finally, Submarine commitments [12] propose that users can rely on k-anonymity alone to privately commit funds during the input-phase without the use of private balances. Here, users commit value to an *k-anonymized* address which can only be withdrawn by a specific smart contract after the address is revealed together with the input by the user.

Secret input stores: We note that shielded scheduled AMM swaps [46] or long-running order lists [24] cannot be maintained by encryption alone: encryption of a scheduled swap by a user implies its decryption at a later stage, requiring repeated user interaction, and thus defeating the purpose of scheduled inputs. Alternatively, a decryption by an honest majority committee implies that the round or block-height of the input schedule is known. Instead, we suggest a long-running MPC instance to realize secret input stores in decentralized finance. Here, stored inputs are secret shared across MPC servers: in each round, both newly submitted inputs and secretly stored inputs are secretly evaluated together to update the application state, neither being visible to the front-running adversary.

References

1. Abraham, I., Pinkas, B., Yanai, A.: Blinder–Scalable, Robust Anonymous Committed Broadcast. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1233–1252 (2020). https://doi.org/10.1145/3372297.3417261
2. Angeris, G., Evans, A., Chitra, T.: A Note on Privacy in Constant Function Market Makers. arXiv preprint arXiv:2103.01193 (2021), https://arxiv.org/abs/2103.01193
3. Avalanche: Apricot Phase Four: Snowman++ and Reduced C-Chain Transaction Fees. https://medium.com/avalancheavax/apricot-phase-four-snowman-and-reduced-c-chain-transaction-fees-1e1f67b42ecf (2021)
4. Bartoletti, M., Chiang, J.H.y., Lluch-Lafuente, A.: A theory of Automated Market Makers in DeFi. In: International Conference on Coordination Languages and Models. pp. 168–187. Springer (2021), https://doi.org/10.1007/978-3-030-78142-2_11
5. Bartoletti, M., Chiang, J.H.y., Lluch-Lafuente, A.: Maximizing Extractable Value from Automated Market Makers. arXiv preprint arXiv:2106.01870 (2021), to appear in FC'22. https://arxiv.org/pdf/2106.01870
6. Baum, C., David, B., Dowsley, R.: Insured MPC: Efficient secure computation with financial penalties. In: International Conference on Financial Cryptography and Data Security. pp. 404–420. Springer (2020). https://doi.org/10.1007/978-3-030-51280-4_22
7. Baum, C., David, B., Frederiksen, T.K.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. In: International Conference on Applied Cryptography and Network Security. pp. 163–194. Springer (2021). https://doi.org/10.1007/978-3-030-78372-3_7
8. Beerliova-Trubiniova, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Theory of Cryptography Conference. pp. 305–328. Springer (2006). https://doi.org/10.1007/11681878_16
9. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Annual cryptology conference. pp. 90–108. Springer (2013). https://doi.org/10.1007/978-3-642-40084-1_6
10. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011). https://doi.org/10.1007/978-3-642-20465-4_11
11. Boneh, D., Naor, M.: Timed commitments. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2000). https://doi.org/10.1007/3-540-44598-6_15
12. Breidenbach, L., Daian, P., Tramèr, F., Juels, A.: Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1335–1352. USENIX Association, Baltimore, MD (Aug 2018), https://www.usenix.org/conference/usenixsecurity18/presentation/breindenbach
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334. IEEE (2018). https://doi.org/10.1109/SP.2018.00020

14. Burdges, J., Feo, L.D.: Delay encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–326. Springer (2021), https://doi.org/10.1007/978-3-030-77870-5_11
15. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC. pp. 494–503. ACM Press, Montréal, Québec, Canada (May 19–21, 2002). https://doi.org/10.1145/509907.509980
16. Chitra, T., Angeris, G., Evans, A.: Differential Privacy in Constant Function Market Makers. Cryptology ePrint Archive (2021), https://eprint.iacr.org/2021/1101
17. Chu, S., Xia, Y., Zhang, Z.: Manta: a Plug and Play Private DeFi Stack (2021), https://eprint.iacr.org/2021/743
18. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In: IEEE Symposium on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040
19. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2007). https://doi.org/10.1007/978-3-540-74143-5_32
20. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012). https://doi.org/10.1007/978-3-642-32009-5_38
21. Danos, V., Khalloufi, H.E., Prat, J.: Global Order Routing on Exchange Networks. In: International Conference on Financial Cryptography and Data Security. pp. 207–226. Springer (2021). https://doi.org/10.1007/978-3-662-63958-0_19
22. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990). https://doi.org/10.1007/0-387-34805-0_28
23. Eskandari, S., Moosavi, S., Clark, J.: SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In: Financial Cryptography. pp. 170–189. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_13
24. da Gama, M.B., Cartlidge, J., Polychroniadou, A., Smart, N.P., Alaoui, Y.T.: Kicking-the-Bucket: Fast Privacy-Preserving Trading Using Buckets. Cryptology ePrint Archive, Report 2021/1549 (2021), to appear in FC'22, https://ia.cr/2021/1549
25. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 626–645. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_37
26. Gentry, C.: Fully Homomorphic Encryption Using Ideal Lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
27. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987). https://doi.org/10.1145/28395.28420

28. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg, Germany, Singapore (Dec 5–9, 2010). https://doi.org/10.1007/978-3-642-17373-8_19
29. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_11
30. Kelkar, M., Deb, S., Kannan, S.: Order-Fair Consensus in the Permissionless Setting. IACR Cryptol. ePrint Arch. **2021**, 139 (2021), https://eprint.iacr.org/2021/139
31. Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.: Themis: Fast, Strong Order-Fairness in Byzantine Consensus. Cryptology ePrint Archive (2021), https://eprint.iacr.org/2021/1465
32. Khalil, R., Gervais, A., Felley, G.: Tex-a securely scalable trustless exchange. Cryptology ePrint Archive (2019), https://eprint.iacr.org/2019/265
33. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 705–734. Springer (2016). https://doi.org/10.1007/978-3-662-49896-5_25
34. Kursawe, K.: Wendy, the good little fairness widget: Achieving order fairness for blockchains. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 25–36 (2020). https://doi.org/10.1145/3419614.3423263
35. Li, Y.: HoneyBadgerSwap: Making MPC as a Sidechain. https://medium.com/initc3org/honeybadgerswap-making-mpc-as-a-sidechain-364bebdb10a5 (2021)
36. Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., Miller, A.: Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 887–903 (2019). https://doi.org/10.1145/3319535.3354238
37. Maxwell, G.: Confidential transactions. https://people.xiph.org/greg/confidential_values.txt, (2016)
38. Paradigm: Ethereum is a Dark Forest. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/ (2020)
39. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013). https://doi.org/10.1109/SP.2013.47
40. Penumbra: ZSwap documentation. https://protocol.penumbra.zone/main/zswap.html (2021)
41. Perez, D., Werner, S.M., Xu, J., Livshits, B.: Liquidations: DeFi on a Knife-edge. In: International Conference on Financial Cryptography and Data Security. pp. 457–476. Springer (2021). https://doi.org/10.1007/978-3-662-64331-0_24
42. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-locked Puzzles and Time-release Crypto. https://people.csail.mit.edu/rivest/pubs/RSW96.pdf (1996)
43. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE (2014). https://doi.org/10.1109/SP.2014.36
44. Shutter: Shutter Network. https://shutter.network/ (2022)

45. Wang, Y., Chen, Y., Deng, S., Wattenhofer, R.: Cyclic Arbitrage in Decentralized Exchange Markets. Available at SSRN 3834535 (2021), https://dx.doi.org/10.2139/ssrn.3834535
46. White, D., Robinson, D., Adams, H.: Time-weighted Average Market Maker (TWAMM) (2021), https://www.paradigm.xyz/2021/07/twamm/
47. Yao, A.C.C.: Theory and applications of trapdoor functions (extended abstract). In: 23rd FOCS. pp. 80–91. IEEE Computer Society Press, Chicago, Illinois (Nov 3–5, 1982). https://doi.org/10.1109/SFCS.1982.45
48. Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A.: On the just-in-time discovery of profit-generating transactions in defi protocols. arXiv preprint arXiv:2103.02228 (2021), https://arxiv.org/abs/2103.02228

A Formalization: speculative sandwich

We formalize the example attack trace introduced in Figure 5 and prove that the attack strategy is either profitable or cost-neutral for the attacker. Again, we assume unit value of τ_0, τ_1 to be equal, and the initial AMM reserve state to be $(\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$: in this state, there is no arbitrage opportunity to be exploited, simplifying our analysis. We omit both AMM and transaction fees.

The victim A swap direction is *left*, inferred by M from A's public balance of $\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} : \tau_0$ (A holds no units of τ_1). The attack strategy is as follows:

1. **Round r**: Front-run victim with $\mathsf{M} : \mathsf{SL}(\mathsf{v}_{\mathsf{M}}^{\mathsf{f}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} : \tau_1)$ such that
$$(\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}) \cdot (\mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}}) = \mathsf{r}^2 \tag{1}$$
2. **Round $\mathbf{r+1}$**: Back-run victim in opposing direction to reestablish initial AMM reserve ratio, or if attacker balance is insufficient, back-run with largest amount available to attacker M.

We must show that this strategy is always profitable (when the victim swap direction can be inferred by the attacker). We note that there are several variables beyond the attackers control. The ordering of both front-run and victim swap in round r is random. Thus the desired "front-run" ordering of the victim swap in round r may not succeed (the sandwich is unsuccessful if the victim swap precedes attacker front-run swap). Furthermore, the victim swap parameters can be arbitrarily chosen, so that the victim swap may not be *enabled* or execute in a given sequence. Thus, we must exhaustively demonstrate the profitability of the attacker strategy for all possible cases:

1) Successful sandwich & enabled victim swap
2) Successful sandwich & disabled victim swap
3) Unsuccessful sandwich & enabled victim swap
4) Unsuccessful sandwich & disabled victim swap

Case 1: *(Successful sandwich & enabled victim swap)*: We illustrate the symbolic execution of the attack trace below in terms of initial balances, chosen swap parameters and exchanged amounts.

$$\textcircled{0}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$$

Round r

$$\xrightarrow{\mathsf{M}:\mathsf{SL}(\mathsf{v}_{\mathsf{M}}^{\mathsf{f}}:\tau_0,\mathsf{w}_{\mathsf{M}}^{\mathsf{f}}:\tau_1)} \textcircled{1}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} + \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} : \tau_0, \mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} : \tau_1)$$

$$\xrightarrow{\mathsf{A}:\mathsf{SL}(\mathsf{v}_{\mathsf{A}}:\tau_0,\mathsf{w}_{\mathsf{A}}:\tau_1)} \textcircled{2}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}_{\mathsf{A}}' : \tau_1] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} + \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} + \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} - \mathsf{w}_{\mathsf{A}}' : \tau_1)$$

Round r + 1

$$\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_{\mathsf{M}}^{\mathsf{b}}:\tau_0,\mathsf{w}_{\mathsf{M}}^{\mathsf{b}}:\tau_1)} \textcircled{3}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}_{\mathsf{A}}' : \tau_1] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} + \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} - \mathsf{w}_{\mathsf{M}}^{\mathsf{b}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}} + \mathsf{v}_{\mathsf{A}} - \mathsf{v}_{\mathsf{M}}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} - \mathsf{w}_{\mathsf{A}}' + \mathsf{w}_{\mathsf{M}}^{\mathsf{b}} : \tau_1)$$

We show that the attack is profitable. For τ_0 and τ_1 of equal unit value, the net change in *value* exchanged by M must be positive. Thus, we must prove

$$\text{profit}_{\mathsf{M}} = -\mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} + \mathsf{v}^{\mathsf{b}}_{\mathsf{M}}{}' > 0 \tag{2}$$

Note that the amounts exchanged in the front-run are equal to the front-run parameters $(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}, \mathsf{w}^{\mathsf{f}}_{\mathsf{M}})$, as they are chosen such that (1) holds. We consider the **sub-case (a)** in which the attacker M has sufficient balance to perform the back-run swap such that the AMM reserves are restored to the original state and the **sub-case (b)** in which the attacker initially has no balance of τ_1 to perform the back-run: $\mathsf{w}^{\mathsf{init}}_{\mathsf{M}} = 0$. Here, the funds of τ_1 required to execute the back-run are received entirely in the front-run execution.

For **sub-case (a)**, we rewrite (2) in terms of independently chosen parameters $\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}$, v_{A} (the attacker only knows the victim swap direction) and initial reserve amounts r. The reserves of the AMM are restored to the initial state in final state ③: summing all step changes to the reserves across the sandwich execution yields

$$\begin{aligned} \mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}} - \mathsf{v}^{\mathsf{b}}_{\mathsf{M}}{}' &= \mathsf{r} \qquad & \mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \mathsf{w}'_{\mathsf{A}} + \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} &= \mathsf{r} \\ \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}} - \mathsf{v}^{\mathsf{b}}_{\mathsf{M}}{}' &= 0 \qquad & - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \mathsf{w}'_{\mathsf{A}} + \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} &= 0 \end{aligned}$$

or

$$\mathsf{v}^{\mathsf{b}}_{\mathsf{M}}{}' = \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}} \qquad \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} = \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{w}'_{\mathsf{A}}$$

Inserting RHS of equations above into our proof obligation (2) yields

$$\begin{aligned} \text{profit}_{\mathsf{M}} = -\cancel{\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} + \cancel{\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} + \mathsf{v}_{\mathsf{A}} + \cancel{\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}} - \cancel{\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}} - \mathsf{w}'_{\mathsf{A}} &>^{?} 0 \\ \mathsf{v}_{\mathsf{A}} - \mathsf{w}'_{\mathsf{A}} &>^{?} 0 \end{aligned} \tag{3}$$

To evaluate whether this inequality holds, we must solve for w'_{A} in terms of v_{A} and $\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}$ chosen independently by the victim and adversary respectively. We exploit the constant reserve product invariant which holds for across the entire execution.

$$\begin{aligned} (\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}) \cdot (\mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}}) &= \mathsf{r}^2 \quad \text{(front-run swap)} \\ (\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) \cdot (\mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \mathsf{w}'_{\mathsf{A}}) &= \mathsf{r}^2 \quad \text{(victim swap)} \end{aligned}$$

We can derive $\mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} = \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}$ from the first equation, and substitute the RHS for $\mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}}$ in the second equation to obtain

$$(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) \cdot (\frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} - \mathsf{w}'_{\mathsf{A}}) = \mathsf{r}^2$$

Solving for w'_{A} ...

$$\begin{aligned}\mathsf{w}'_{\mathsf{A}} &= \frac{\mathsf{r}^2}{\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} - \frac{\mathsf{r}^2}{\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}}} \\ &= \frac{\mathsf{r}^2(\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}}) - \mathsf{r}^2(\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})}{(\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}})} \\ &= \frac{\mathsf{r}^2}{\mathsf{r}^2+(2\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}})\mathsf{r}+(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})^2+\mathsf{v}_{\mathsf{A}}\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} \cdot \mathsf{v}_{\mathsf{A}}\end{aligned}$$

and substituting the RHS for w'_{A} in the proof obligation in (3) finally yields

$$\text{profit}_{\mathsf{M}} = \left(1 - \frac{\mathsf{r}^2}{\mathsf{r}^2+(2\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}})\mathsf{r}+(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})^2+\mathsf{v}_{\mathsf{A}}\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}\right) \cdot \mathsf{v}_{\mathsf{A}} > 0 \tag{4}$$

The fraction expression above is less than 1 for any choice of positive $\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}$ and v_{A} as the numerator is smaller than the denominator. The attacker profit is thus positive and increases with v_{M}, justifying the front-run swap by M.

Next, we consider the **sub-case (b)**, where the attacker initially has no balance of τ_1, and restate the profit of attacker for the reader's convenience.

$$\text{profit}_{\mathsf{M}} = -\mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} + \mathsf{v}^{\mathsf{b}\,\prime}_{\mathsf{M}} >^? 0$$

We assume initial attacker balance in $\mathsf{w}^{\mathsf{init}}_{\mathsf{M}} : \tau_1$ to be $0 : \tau_1$, so that all the amount of τ_1 available for the back-run in state ② is received in the front-run: thus, substituting $\mathsf{w}^{\mathsf{b}}_{\mathsf{M}} = \mathsf{w}^{\mathsf{f}}_{\mathsf{M}}$ into the equation above yields

$$\text{profit}_{\mathsf{M}} = -\mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}^{\mathsf{b}\,\prime}_{\mathsf{M}} >^? 0 \tag{5}$$

To prove this inequality, we solve for $\mathsf{v}^{\mathsf{b}\,\prime}_{\mathsf{M}}$ in terms of $\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}$ and v_{A} chosen independently by the victim and adversary respectively and initial reserves amounts r. We exploit the constant reserve product invariant which holds throughout the execution.

$$\begin{aligned}(\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})\cdot(\mathsf{r}-\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}) &= \mathsf{r}^2 \quad \text{(Front-run)} \\ (\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}})\cdot(\mathsf{r}-\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}-\mathsf{w}'_{\mathsf{A}}) &= \mathsf{r}^2 \quad \text{(Victim swap)} \\ (\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}}-\mathsf{v}^{\mathsf{b}\,\prime}_{\mathsf{M}})\cdot(\mathsf{r}-\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}-\mathsf{w}'_{\mathsf{A}}+\mathsf{w}^{\mathsf{b}}_{\mathsf{M}}) &= \mathsf{r}^2 \quad \text{(Back-run)}\end{aligned}$$

Since $\mathsf{w}^{\mathsf{f}}_{\mathsf{M}} = \mathsf{w}^{\mathsf{b}}_{\mathsf{M}}$ is assumed in sub-case (b), the 3rd equation (back-run) yields

$$\mathsf{v}^{\mathsf{b}\,\prime}_{\mathsf{M}} = \mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}} - \frac{\mathsf{r}^2}{\mathsf{r}-\mathsf{w}'_{\mathsf{A}}} \tag{6}$$

From the 2nd equation (victim swap), we solve for w'_{A} in terms of independent parameters $\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}$, v_{A} and r

$$\mathsf{w}'_{\mathsf{A}} = \mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} - \frac{\mathsf{r}^2}{\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}+\mathsf{v}_{\mathsf{A}}}$$

From the 1st equation (front-run) $\mathsf{w}^{\mathsf{f}}_{\mathsf{M}} = \frac{\mathsf{r} \cdot \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}{\mathsf{r}+\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}$, so we can rewrite the above as

$$\mathsf{w}'_{\mathsf{A}} = \mathsf{r} - \frac{\mathsf{r} \cdot \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}} = \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}} = \frac{\mathsf{r}^2 \cdot \mathsf{v}_{\mathsf{A}}}{(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}})}$$

$$\mathsf{r} - \mathsf{w}'_{\mathsf{A}} = \frac{\mathsf{r}(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) - \mathsf{r}^2 \cdot \mathsf{v}_{\mathsf{A}}}{(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}})}$$

Substituting the RHS above for $\mathsf{r} - \mathsf{w}'_{\mathsf{A}}$ in the denominator expression of (6) and then substituting the RHS of (6) for ${\mathsf{v}^{\mathsf{b}}_{\mathsf{M}}}'$ in (5) yields

$$\begin{aligned}
\text{profit}_{\mathsf{M}} &= -\cancel{\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} + \mathsf{r} + \cancel{\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}} + \mathsf{v}_{\mathsf{A}} - \frac{\mathsf{r}^2(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}})}{\mathsf{r}(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) - \mathsf{r}^2 \cdot \mathsf{v}_{\mathsf{A}}} \\
&= \mathsf{v}_{\mathsf{A}} - \frac{\mathsf{r}^3 \mathsf{v}_{\mathsf{A}}}{\mathsf{r}(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) - \mathsf{r}^2 \cdot \mathsf{v}_{\mathsf{A}}} \\
&= (1 - \frac{\mathsf{r}^2 \mathsf{v}_{\mathsf{A}}}{(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}})(\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} + \mathsf{v}_{\mathsf{A}}) - \mathsf{r} \cdot \mathsf{v}_{\mathsf{A}}}) \cdot \mathsf{v}_{\mathsf{A}} \\
&= (1 - \frac{\mathsf{r}^2}{\mathsf{r}^2 + 2\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}\mathsf{r} + (\mathsf{v}^{\mathsf{f}}_{\mathsf{M}})^2 + \mathsf{v}_{\mathsf{A}}\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}}) \cdot \mathsf{v}_{\mathsf{A}} \qquad (7)
\end{aligned}$$

The attacker profit is positive but strictly less than the gain (4) obtained in sub-case (a).

Case 2 *(Successful sandwich & disabled victim swap)*: Should the victim swap not execute in round r, then M can simply revert the state of the AMM with a back-run in the round $r+1$ with the same parameter values as in the front-run.

$$⓪\ \mathsf{A}[\mathsf{v}^{\mathsf{init}}_{\mathsf{A}} : \tau_0] \mid \mathsf{M}[\mathsf{v}^{\mathsf{init}}_{\mathsf{A}} : \tau_0, \mathsf{w}^{\mathsf{init}}_{\mathsf{A}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$$

Round r

$$\xrightarrow{\mathsf{M}:\mathsf{SL}(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}:\tau_0,\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}:\tau_1)} ①\ \mathsf{A}[\mathsf{v}^{\mathsf{init}}_{\mathsf{A}} : \tau_0] \mid \mathsf{M}[\mathsf{v}^{\mathsf{init}}_{\mathsf{M}} - \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{init}}_{\mathsf{M}} + \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1)$$

$$\xrightarrow{\cancel{\mathsf{A}:\mathsf{SL}(\mathsf{v}_{\mathsf{A}}:\tau_0,\mathsf{w}_{\mathsf{A}}:\tau_1)}} ②\ \mathsf{A}[\mathsf{v}^{\mathsf{init}}_{\mathsf{A}} : \tau_0] \mid \mathsf{M}[\mathsf{v}^{\mathsf{init}}_{\mathsf{M}} - \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{init}}_{\mathsf{M}} + \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{r} - \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1)$$

Round $\mathsf{r}+1$

$$\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}}:\tau_0,\mathsf{w}^{\mathsf{f}}_{\mathsf{M}}:\tau_1)} ③\ \mathsf{A}[\mathsf{v}^{\mathsf{init}}_{\mathsf{A}} : \tau_0] \mid \mathsf{M}[\mathsf{v}^{\mathsf{init}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{init}}_{\mathsf{M}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$$

The attack execution is trivially cost-neutral for M.

Case 3 *(Failed sandwich & enabled victim swap)*: We must show that the attacker front-run must be disabled assuming the attacker parameters are chosen as described in the attack strategy. Further, we can demonstrate that the back-run by the attacker is profitable.

$$\textcircled{0}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$$

Round r

$$\xrightarrow{\mathsf{A}:\mathsf{SL}(\mathsf{v}_{\mathsf{A}}:\tau_0,\mathsf{w}_{\mathsf{A}}:\tau_1)} \textcircled{1}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}'_{\mathsf{A}} : \tau_1] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{r} - \mathsf{w}'_{\mathsf{A}} : \tau_1)$$

$$\xrightarrow{\cancel{\mathsf{M}:\mathsf{SL}(\mathsf{v}_{\mathsf{M}}^{\mathsf{f}}:\tau_0,\mathsf{w}_{\mathsf{M}}^{\mathsf{f}}:\tau_1)}} \textcircled{2}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}'_{\mathsf{A}} : \tau_1] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{r} - \mathsf{w}'_{\mathsf{A}} : \tau_1)$$

Round $\mathsf{r}+1$

$$\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_{\mathsf{M}}^{\mathsf{b}}:\tau_0,\mathsf{w}_{\mathsf{M}}^{\mathsf{b}}:\tau_1)} \textcircled{3}\ \mathsf{A}[\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} - \mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}'_{\mathsf{A}} : \tau_1] \mid \mathsf{M}[\mathsf{v}_{\mathsf{M}}^{\mathsf{init}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{init}} - \mathsf{w}_{\mathsf{M}}^{\mathsf{b}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$$

As described in step (1) of attack strategy, M's front-run parameters are chosen such that

$$(\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}) \cdot (\mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}}) = \mathsf{r}^2$$

$$\mathsf{w}_{\mathsf{M}}^{\mathsf{f}} = \frac{\mathsf{r} \cdot \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}}{\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}} \tag{8}$$

Thus, the front-run swap is only enabled if the received amount is equal or greater to $\mathsf{w}_{\mathsf{M}}^{\mathsf{f}}$ shown above. Note, that this doesn't hold if the front-run is executed in state $\textcircled{1}$ of case (3) following the enabled victim swap. We prove this by contradiction: assume that the front-run executes following the victim swap, then the constant reserve product invariant must hold.

$$(\mathsf{r} + \mathsf{v}_{\mathsf{A}}) \cdot (\mathsf{r} - \mathsf{w}'_{\mathsf{A}}) = \mathsf{r}^2 \quad \text{(Victim swap)}$$

$$(\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}) \cdot (\mathsf{r} - \mathsf{w}'_{\mathsf{A}} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime}) = \mathsf{r}^2 \quad \text{(Front-run)}$$

We solve for $(\mathsf{r} - \mathsf{w}'_{\mathsf{A}})$ in the first equation and insert into the second equation to obtain

$$(\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}) \cdot (\frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_{\mathsf{A}}} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime}) = \mathsf{r}^2$$

Further, we solve for $\mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime}$ in terms of r, v_{A} and $\mathsf{v}_{\mathsf{M}}^{\mathsf{f}}$

$$\frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_{\mathsf{A}}} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime} = \frac{\mathsf{r}^2}{(\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}})}$$

$$\mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime} = \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_{\mathsf{A}}} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}} = \frac{\mathsf{r}^2 \cdot \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}}{(\mathsf{r} + \mathsf{v}_{\mathsf{A}}) \cdot (\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}})} = \frac{\mathsf{r}}{\mathsf{r} + \mathsf{v}_{\mathsf{A}}} \cdot \frac{\mathsf{r} \cdot \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}}{(\mathsf{r} + \mathsf{v}_{\mathsf{A}} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}})}$$

Comparing with $\mathsf{w}_{\mathsf{M}}^{\mathsf{f}}$ in (8), we can infer the following inequality

$$\mathsf{w}_{\mathsf{M}}^{\mathsf{f}\,\prime} < \mathsf{w}_{\mathsf{M}}^{\mathsf{f}}$$

which cannot hold in a valid execution by definition of swaps: a user cannot receive less than the chosen swap limit. Thus, the front-run cannot be enabled in state $\textcircled{1}$ of case (3).

Next, we prove the profitability of the back-run. Assuming a sufficient balance of the attacker to revert the effect of the victim swap, the swap parameters of the back-run can be chosen to reverse the affects of victim swap on the AMM reserves, which M observes following the output-phase of round r: namely, $v_M^b = v_A$ and $w_M^b = w_A'$. We insert these into the reserve product invariant from the victim swap

$$(r + v_A) \cdot (r - w_A') = r^2 \quad \text{(Victim swap)}$$

to obtain

$$\begin{aligned}(r + v_M^b) \cdot (r - w_M^b) &= r^2 \\ w_M^b &= \frac{r}{r + v_M^b} \cdot v_M^b \\ w_M^b &< v_M^b\end{aligned}$$

For equal unit value of both token types, this is clearly profitable, as M receives more value (v_M^b) as it sends (w_M^b). If attacker has no balance of τ_1 it simply omits the back-run and the attack is aborted, resulting in a cost-neutral execution for the attacker.

Case 4 *(Failed sandwich & disabled victim swap)*: As in case (2) - should the victim swap not execute in round r, then M can simply revert the state of the AMM with a back-run in the round $r + 1$

$$\textcircled{0}\ A[v_A^{init} : \tau_0] \mid M[v_A^{init} : \tau_0, w_A^{init} : \tau_1] \mid (r : \tau_0, r : \tau_1)$$

Round r

$$\xrightarrow{\cancel{A:SL(v_A:\tau_0, w_A:\tau_1)}} \textcircled{1}\ A[v_A^{init} : \tau_0] \mid M[v_M^{init} : \tau_0, w_M^{init} : \tau_1] \mid (r : \tau_0, r : \tau_1)$$

$$\xrightarrow{M:SL(v_M^f:\tau_0, w_M^f:\tau_1)} \textcircled{2}\ A[v_A^{init} : \tau_0] \mid M[v_M^{init} - v_M^f : \tau_0, w_M^{init} + w_M^f : \tau_1] \mid (r + v_M^f : \tau_0, r - w_M^f : \tau_1)$$

Round $r + 1$

$$\xrightarrow{M:SR(v_M^f:\tau_0, w_M^f:\tau_1)} \textcircled{3}\ A[v_A^{init} : \tau_0] \mid M[v_M^{init} : \tau_0, w_M^{init} : \tau_1] \mid (r : \tau_0, r : \tau_1)$$

The attack execution is trivially cost-neutral for M.

B Speculative sandwich with private user balances

Importantly, when performing the speculative AMM swap attack as shown in A, the direction of the victim swap must be known. If user balances are private, M will have to guess the direction of the front-running swap. However, this is not a profitable strategy: an incorrect guess can result in a loss for M as shown in the trivial example execution below.

$$\begin{array}{rl}
 & \mathsf{A}[10:\tau_0, 10:\tau_1] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (100:\tau_0, 100:\tau_1) \\
\hline
\multicolumn{2}{c}{\text{Round } \mathsf{r}} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(7:\tau_0,6.5:\tau_1)} & \mathsf{A}[10:\tau_0, 10:\tau_1] \mid \mathsf{M}[21.5:\tau_1] \mid (107:\tau_0, 93.5:\tau_1) \\
\xrightarrow{\mathsf{A}:\mathsf{SR}(17:\tau_0,6.5:\tau_1)} & \mathsf{A}[7:\tau_0, 3.5:\tau_1] \mid \mathsf{M}[21.5:\tau_1] \mid (100:\tau_0, 100:\tau_1)
\end{array}$$

Again, assuming equal unit value of τ_0 and τ_1, M realizes a loss of $7+15-21.5 = 0.5$. No back-run swap is possible that extracts any arbitrage value given that the reserve ratio is already consistent with the assumption that unit values of τ_0 and τ_1 are equal [4]. Thus, speculative sandwich attacks are only rational if the victim swap direction can be inferred, motivating the need for private user balances.

C Example: speculative sandwich of scheduled swap

We illustrate an example of a sandwich of a scheduled swap. Such an attack can be exploited despite the batching of blinded user inputs §3.2, as long as input schedules remain public. Let $\mathsf{A} : \mathsf{SL}(20 : \tau_0, 15 : \tau_1, \mathsf{r})$ be a swap action that is scheduled to execute as soon as possible *following* block-chain round r, thus requiring no further interaction from the user. Further, let the set of scheduled swap orders be captured in a publicly observable state fragment, i.e. $\Gamma = [\, \mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1, \mathsf{r}) \,]$. In practice, such a scheduled swap order will be *evaluated* prior to the first swap order in round $\mathsf{r}+1$, so that it is not possible for the adversary to place a front-run swap before it in round $\mathsf{r}+1$.

However, the sandwich attack can still be executed by an adversary which prevents honest users from submitting swap. The adversary simply submits the front-run to round r, and the back-run to round $\mathsf{r}+1$, whilst suppressing all other user inputs.

$$\begin{array}{rl}
 & \mathsf{A}[15:\tau_0] \mid \mathsf{M}[15:\tau_0, 10:\tau_1] \mid (100:\tau_0, 100:\tau_1) \mid \Gamma \\
\hline
\multicolumn{2}{c}{\text{Round } \mathsf{r}} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(15:\tau_0,13:\tau_1)} & \mathsf{A}[15:\tau_0] \mid \mathsf{M}[23:\tau_1] \mid (115:\tau_0, 87:\tau_1) \mid \Gamma \\
\hline
\multicolumn{2}{c}{\text{Round } \mathsf{r}+1} \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1,\mathsf{r})} & \mathsf{A}[10:\tau_1] \mid \mathsf{M}[23:\tau_1] \mid (130:\tau_0, 77:\tau_1) \mid \\
 & \Gamma \setminus [\, \mathsf{A} : \mathsf{SL}(15:\tau_0, 10:\tau_1), \mathsf{r} \,] \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(30:\tau_0,23:\tau_1)} & \mathsf{A}[10:\tau_1] \mid \mathsf{M}[30:\tau_0] \mid (100:\tau_0, 100:\tau_1) \mid \\
 & \Gamma \setminus [\, \mathsf{A} : \mathsf{SL}(15:\tau_0, 10:\tau_1, \mathsf{r}) \,]
\end{array}$$

We emphasize that scheduled swap orders do not require the submitting user A to participate in the round it is scheduled: it is evaluated automatically by the application. Furthermore, since the victim's swap parameters are public, the front-run and back-run parameters can be chosen to optimize M's profit.

D Speculative sandwich in hash-based commit & reveal schemes

As shown in Appendix A, the speculative sandwich attack is rational as long as the direction of the victim swap is known. Hash-based commit & reveal schemes suffer from selective output by the adversary (fig. 4), permitting a speculative attack to succeed even if the swap direction cannot be inferred from public user balances. Here the attacker simply commits two front-run swaps of opposing directions in the same round as the victim swap, whilst suppressing other user inputs. In the output-phase, the adversary learns the direction of the victim swap before having to open its own commitments and selectively opens the front-run of the same direction as the victim swap, whilst refraining from opening the other front-run swap. The back-run is then executed as in Appendix A.

FairPoS: Input Fairness in Proof-of-Stake with Adaptive Security

Contribution

- Co-author.

Remarks

Under submission.

FairPoS: Input Fairness in Proof-of-Stake with Adaptive Security

James Hsin-yu Chiang[1], Bernardo David[2], Ittay Eyal[3], Tiantian Gong[4]

[1] Technical University of Denmark, Denmark
jchi@dtu.dk
[2] IT University of Copenhagen, Denmark
bernardo@bmdavid.com
[3] Technion, IC3, Haifa, Israel
ittay@technion.ac.il
[4] Purdue University, West Lafayette, USA
tg@purdue.edu

Abstract. We present "FairPoS", the first blockchain protocol that achieves input fairness with adaptive security. Here, we introduce a novel notion of "input fairness": the adversary cannot learn the plain-text of any finalized client input *before* it is included in a block in the chain's common-prefix. Should input fairness hold, input ordering attacks which depend on the knowledge of plain-text of client inputs are thwarted. In FairPoS, input fairness with adaptive security is achieved by means of the delay encryption scheme of DeFeo *et al.* [9], a recent cryptographic primitive related to time-lock puzzles, allowing *all* client inputs in a given round to be encrypted under the same key, which can only be extracted after enough time has elapsed. In contrast, alternative proposals that prevent input order attacks by encrypting user inputs are not adaptively secure as they rely on small static committees to perform distributed key generation and threshold decryption for efficiency's sake. Such small committees are easily corrupted by an adaptive adversary with a corruption budget applicable over a large set of participants in a permissionless blockchain system. The key extraction task in delay encryption can, in principle, be performed by any party and is secure upon adaptive corruption, as no secret key material is learned. However, the key extraction requires highly specialized hardware in practice. Thus, FairPoS requires resource-rich, staking parties to insert extracted keys to blocks which enables light-clients to decrypt past inputs. Note that naive application of key extraction can result in chain stalls lasting the *entire* key extraction period. In FairPoS, this is addressed by a novel *longest-extendable-chain* rule. We formally prove that FairPoS achieves input fairness and the original security of Ouroborous Praos against an adaptive adversary.

1 Introduction

Blockchain protocols permit the elected leader of each round to produce a block containing an ordered list of transactions chosen by its leader. This ordering privilege is exploited in front-running [12], where adversarial inputs can be interleaved with honest inputs to extract financial value from the honest victim in applications such as automatic market makers [4]. Such behaviour financially penalizes the honest user, but also generates excess demand for block-space since front-running attacks [4] always require additional inputs from the adversary, thereby inflicting block congestion, as acutely observed on Avalanche [2].

We introduce FairPoS which achieves a novel notion of *input fairness* whilst retaining the asymptotic security of Ouroborous Praos [13]. Informally, input fairness ensures that the plain-text content of any finalized input (in the common-prefix) could not have been observed by the adversary prior to its finalization. FairPoS achieves this by encrypting inputs with a delay encryption scheme by DeFeo et al. [9], which is similar to time-lock puzzles [28], but allows all client inputs of a given round to be encrypted under the same key, thereby requiring only a single key extraction for each block. The *extraction* procedure to recover the decryption key is parameterized to run in at least time d, and can be be performed by any party with access to specialized hardware to perform extractions in d time. This preserves adaptive-security, as no relevant key material is learned upon corruption of an honest party.

Still, it is not practical for light-clients or non-staking parties to perform key extraction. Firstly, we expect only resource-rich participants to have access to the specialized hardware [1] necessary to perform extractions in exactly d time. Secondly, any party joining the protocol would need to perform key extractions for all blocks beginning from genesis, which becomes increasingly impractical at higher chain lengths. In FairPoS, staking parties must insert the extracted keys from past inputs into blocks within a fixed schedule, thus ensuring decryption keys are made publicly available in lock-step with chain growth. We note that adversarial delay of blocks propagation can impede chain growth if honest parties cannot finish key extractions on time due to delayed arrival. In FairPoS, such attacks are addressed with a novel *longest-extractable-chain* rule, which asserts a notion of *timeliness* on the arrival of blocks, ensuring that honest leaders can complete the key extraction on schedule.

Comparison to related work. Recent proposals achieve a similar notion with distributed key generation and threshold encryption committees [5,26,27], where a small subset of protocol participants jointly generate a public key, to which user inputs are encrypted in each round. Inputs are subsequently decrypted by the same committee when the inputs are finalized. This approach is not secure against an adaptive adversary, unlike the underlying permissionless blockchains.

Protocols such as Bitcoin [16] and Ouroborous Praos [13] achieve security against an adversary that can adaptively corrupt parties in a large participant set as the protocol execution progresses; performing any additional interactive cryptographic protocol over such a large set of intermittently available parties is not practical, so instead, committees assigned to the protocol task can be elected from the larger global set of parties executing the blockchain protocol [18]. For the *interactive* task of performing distributed key generation and threshold decryption, however, the *adaptive* adversary can easily identify the active committee parties once the first protocol message is sent. Upon corruption it will learn secret key material, which cannot be erased by parties as it must be maintained as interactive protocol state[5].

Another line of research [23,22,21,10] proposes a notion of *fair input ordering*. A block leader will order inputs based on their order of arrival. However, fair ordering is only meaningful in a setting with a secure connection between client and round leaders: in a peer-to-peer gossip network setting common in massively distributed permissionless blockchain protocols, the receipt-order of messages is adversarially controlled, rendering the notion of fair ordering highly impractical. A secure connection with the next block leader implies public knowledge of its identity, contradicting adaptive security again.

Overview. In Section 2, we introduce Delay Encryption and an abstract model of an Ouroborous Praos execution (δ-PoS). In Section 3, we then define our proposed notion of Input Fairness and extend the PoS model with delay encryption and a novel "longest-extendable-chain" selection rule to obtain a formal model of $\mathsf{FairPoS}$. In Section 4, we prove that $\mathsf{FairPoS}$ achieves input fairness whilst maintaining the asymptotic security of Ouroborous Praos (PoS) against an adaptive adversary. Proofs of stated theorems and lemmas are provided in Appendix D.

2 Preliminaries

2.1 Delay Encryption

The delay encryption (DE) scheme by De Feo *et al.* [9] consists of the following four algorithms: A global $\mathsf{DE.Setup}$ parameterized with a security parameter $\lambda \in \{0,1\}^*$ and delay parameter d generates public encryption ($\mathsf{DE.pk}$) and extraction ($\mathsf{DE.ek}$) keys. In each round, a public session $\mathsf{id} \in \{0,1\}^*$ is sampled, and $\mathsf{DE.Encaps}$ can be used to generate a pair (c,k) of a ciphertext c and a key k corresponding to id and the encryption key $\mathsf{DE.pk}$. The $\mathsf{DE.Extract}$ algorithm runs in at least d time, and returns a session key idk, with which the $\mathsf{DE.Decaps}$ algorithm can compute a key k from ciphertext c for all (c,k) generated with the same session id and public paramaters from a given setup.

1. $\mathsf{DE.Setup}(\lambda, d) \rightarrow (\mathsf{DE.ek}, \mathsf{DE.pk})$
2. $\mathsf{DE.Encaps}(\mathsf{DE.pk}, \mathsf{id}) \rightarrow (c,k)$
3. $\mathsf{DE.Extract}(\mathsf{DE.ek}, \mathsf{id}) \rightarrow \mathsf{idk}$
4. $\mathsf{DE.Decaps}(\mathsf{DE.pk}, \mathsf{id}, \mathsf{idk}, c) \rightarrow k$

Delay encryption is an isogeny-based delay protocol, and similar to [14] is built from isogeny walks in graphs of pairing friendly supersingular elliptic curves. In implementations [14], such isogeny evaluations occupy memory space in the terabytes. Parties performing *Extract* are expected to deploy specialized FPGA hardware [1] in order to achieved the parameterized extraction time.

[5] We note a line of work which achieves adaptive security in large scale cryptographic protocols via anonymous committees [7,19,11,17,15]. However, the efficiency of such approaches remains impractical. We consider this as an orthogonal line of research.

2.2 Longest-chain PoS model and security

We present a model of *longest-chain proof-of-stake* protocols, formalized by the Ouroborous line of work [25,13,3] and subsequent improvements [8,24]. We adopt modelling approach in [25,13,3,24], where the PoS protocol is modelled by two orthogonal components: the first describes the *leader election process* and the second part models the views of blockchain trees which result from a protocol execution *induced by a given leader schedule.*

Idealized leader elections. Time in PoS is divided into units named slots, each capturing the duration of a single protocol round. In a given round, a party with relative stake $\alpha \in (0, 1]$ becomes a slot leader for a given slot with probability

$$\phi(\alpha) = 1 - (1 - f)^{\alpha}$$

where parameter *active slot coefficient* f is the probability that a leader holding all stake will be elected leader in given slot: importantly, $\phi(\alpha)$ is maintained even if share α is split amongst multiple, virtual parties (eq. 2 in [13]). Let a characteristic string w be defined as a sequence of leader election results, where an election result at slot t is defined as follows.

$$w_t = \begin{cases} 0 & \text{a single honest leader} \\ 1 & \text{multiple honest leaders / adversarial leader} \\ \perp & \text{no leader} \end{cases}$$

In PoS [13], leader election is modelled by sampling characteristic strings from an idealized, *dominant distribution* $\mathcal{D}^f_\alpha$ that is strictly *more adversarial* than the *true* setting where the *adaptive adversary* corrupts up to $(1 - \alpha)$ of the stake during the protocol execution (Theorem 8 in [13]). Thus, any security that holds in PoS executions induced by characteristic strings sampled from $\mathcal{D}^f_\alpha$ must also hold in the true protocol execution against an adaptive adversary dynamically corrupting up to $1 - \alpha$ stake.

Definition 1 (Dominant distribution $\mathcal{D}^f_\alpha$ (Definition 11 in [13])). *For an adaptive adversary corrupting up to $1-\alpha$ stake fraction and active slot coefficient $f \in [0, 1)$, the dominant distribution $\mathcal{D}^f_\alpha$ is defined by the following probabilities:*

$$p_\perp = 1 - f \qquad p_0 = \phi(\alpha) \cdot (1 - f) \qquad p_1 = 1 - p_\perp - p_1 \tag{1}$$

Definition 2 (Blocks, chains, trees and branches). *A block $B = (\mathsf{sl}, \mathsf{st}, \mathsf{d}, \mathsf{ldr})$ generated at slot sl contains state $\mathsf{st} \in \{0, 1\}^*$, data $\mathsf{d} \in \{0, 1\}^*$ and party ldr that generated B and was the leader of slot sl. A chain is a sequence of blocks $B_0, ..., B_n$ associated with a strictly increasing sequence of slots, where B_0 is the genesis block, and the state of B_i is $H(B_{i-1})$, $H(\cdot)$ denoting a collision-resistant hash function. We write $\mathcal{C}.\mathsf{tip}$ to denote the block at the tip of chain $\mathcal{C}$ and $\mathcal{C}_j$ to denote a block $B \in \mathcal{C}$ such that $B.\mathsf{sl} = j$. If such a block does not exist, $\mathcal{C}_j = \perp$. Let $\mathcal{C}^{\lceil k}$ denote the chain obtained from $\mathcal{C}$ by removing the last k blocks. Multiple chains form a tree if their blocks share state. A branch $\mathcal{B}$ in a tree $\mathcal{T}$ is a chain* *which ends with a leaf block. We write $\mathcal{C} \preceq \mathcal{B}$ to indicate $\mathcal{C}$ is a prefix of $\mathcal{B}$. When quantifying over all chains in a tree, $\forall \mathcal{C} \in \mathcal{T}$, we quantify over all prefixes of all tree branches. Let $\mathsf{len}(\mathcal{C})$ denote the number of blocks in chain $\mathcal{C}$.*

δ-PoS execution model

The evolution of a PoS execution state $\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i \in \mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right)$ in a single protocol round $\Gamma_t \xrightarrow{w_{t+1}} \Gamma_{t+1}$ induced by environment $\mathcal{Z}$, adversary $\mathcal{A}$ and characteristic string w occurs as follows.

For local tree views $\{\mathcal{T}^{(i)}\}_{i \in \mathcal{H}}$ and $\mathcal{T}^{\mathcal{A}}$:

T0. $\mathcal{T}_0^{(i)}$ consists of the genesis block.
T1. If $w_{t+1} = 0$, a single honest party runs **Extend** on the longest chain $\mathcal{C}$ in $\mathcal{T}^{(i)}$, upon which the extended chain is added to $\mathcal{T}^{(i)}$ and $\mathcal{T}^{\mathcal{A}}$.
T2. At any time during the round, adversary $\mathcal{A}$ may:
- **a)** Run **Extend** on a chain in $\mathcal{T}^{\mathcal{A}}$ for slot $t+1$ if $w_{t+1} = 1$, upon which the updated chain is added to $\mathcal{T}^{\mathcal{A}}$.
- **b)** Update any honest tree view $\mathcal{T}^{(i)}$ with a chain from $\mathcal{T}^{\mathcal{A}}$ or an honest message queue $\{\mathbf{m}^{(i)}\}_{i \in \mathcal{H}}$ (see **M2**).

For honest message queues $\{\mathbf{m}^{(i)}\}_{i \in \mathcal{H}}$:

M1. For a chain $\mathcal{C}'$ extended by honest party i during the round (**T2**), the entry $(\mathcal{C}', \mathcal{H})$ is added to local message queue $\mathbf{m}^{(i)}$.
M2. At any time during the round, adversary $\mathcal{A}$ may deliver a chain from an entry $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$ to a subset of honest users $\mathcal{I} \subseteq \mathcal{H}$:
- **a)** The entry $(\mathcal{C}, \mathcal{P})$ in $\mathbf{m}^{(i)}$ is updated to $(\mathcal{C}, \mathcal{P}')$, where $\mathcal{P}' = \mathcal{P} \backslash \mathcal{I}$.
- **b)** Each $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$ must be delivered to all honest parties $\mathcal{H}$ by slot $\mathcal{C}.\mathsf{sl} + \delta$, and is removed from $\mathbf{m}^{(i)}$ when delivery is completed.

Extend. To extend $\mathcal{C}$, the party i generates $B = (t+1, H(\mathcal{C}.\mathsf{tip}), d, i)$ containing an *ordering* of inputs $d = \{\mathsf{in}_i\}_{i \in [m]}$ provided by $\mathcal{Z}$ for slot t.

Fig. 1. δ-PoS model induced by environment $\mathcal{Z}$ and characteristic string w.

A model of δ-PoS exections. As in [16,25,13,3], we model the execution of PoS initiated upon the activation of an environment $\mathcal{Z}$, which spawns both honest parties $\mathcal{H}$ and an adversary $\mathcal{A}$. Upon each activation by the environment, each party executes the protocol according to Figure 1, which precisely models the adversarial powers to influence the round-wise evolution of block tree structures in the local view parties as the full PoS protocol in [13], but omits details such as block proofs, signatures or individual leader election procedures such as evaluation of verifiable random functions. A given characteristic string w *induces* executions of our δ-PoS model that generate local tree structures identical to those resulting from a full PoS [13] protocol execution that *induces* a leader election sequence consistent with w and activates the same parties and adversarial actions.

Let the protocol execution state $\mathit{\Gamma}_t$ in slot t, consist of honest party states, including the local block tree view $\mathcal{T}^{(i)}$ and the outbound message queue $\mathbf{m}^{(i)}$ for each honest party $i \in \mathcal{H}$. Further, let $\mathit{\Gamma}_t$ include the blockchain tree view $\mathcal{T}^{\mathcal{A}}$ of the adversary.

$$\mathit{\Gamma}_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right) \tag{2}$$

The outbound message queue $\mathbf{m}^{(i)} = \{(\mathcal{C}, \mathcal{P}), ...\}$ in $\mathit{\Gamma}_t$ is the set of broadcast, yet undelivered chains previously sent by honest party i. For each entry $(\mathcal{C}, \mathcal{P}) \in \mathbf{m}^{(i)}$, $\mathcal{C}$ was initially broadcast and added to the local message queue at slot $\mathcal{C}.\mathsf{sl} \leq t$. Each entry in $\mathbf{m}^{(i)}$ consists of a chain $\mathcal{C}$ and honest party subset $\mathcal{P} \subset \mathcal{H}$, which has yet to receive the message. $\mathcal{A}$ is required to deliver all honestly broadcast chains with a delay of no more than δ slots. The model executes round-wise beginning from initial state $\mathit{\Gamma}_0$, where the tree views of all parties consist only of the genesis block.

In each round from slot t to $t+1$, the leader is implied by by $w_{t+1} \in \{0, 1, \perp\}$. For a uniquely honest slot, the environment $\mathcal{Z}$ is permitted to activate any honest party to extend the longest chain in its local view, where the inputs for insertion in the block are provided by $\mathcal{Z}$. We interpret $w_{\mathsf{slot}} = 1$ as a strictly adversarial slot, since the adversary could affect the structure of local trees views in the same way as multiple honest leaders: namely, by producing multiple blocks associated with the same slot.

PoS Security. The seminal work on formalizing the Bitcoin backbone protocol [16] proved *liveness* and *persistence* of longest-chain proof-of-work (PoW) protocols in terms of common-prefix, chain growth and chain quality properties, which are also achieved for PoS in Ouroboros Praos [13]. We restate these below and formally prove them for FairPoS in Section 4.

Definition 3 (Common prefix, k-CP; with parameter $k \in \mathbb{N}$). *The chains $\mathcal{C}_1$, $\mathcal{C}_2$ possessed by two honest parties at the onset of the slots $t_1 < t_2$ are such that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$, where $\mathcal{C}_1^{\lceil k}$ denotes the chain $\mathcal{C}_1^{\lceil k}$ obtained by removing the last k blocks from $\mathcal{C}_1$, and $\preceq$ denotes the prefix relation.*

Definition 4 (Chain growth, (τ, s)-CG; with parameter $\tau \in (0, 1]$ and $s \in \mathbb{N}$). *Consider the chains $\mathcal{C}_1$, $\mathcal{C}_2$ possessed by two honest parties at the onset of two slots t_1, t_2 with t_2 at least s slots ahead of t_1. Then it holds that $\mathit{len}(\mathcal{C}_2) - \mathit{len}(\mathcal{C}_1) \geq \tau \cdot s$. We call τ the speed coefficient.*

Definition 5 (Chain quality, (μ, k)-CQ; with parameters $\mu \in (0, 1]$ and $k \in \mathbb{N}$). *Consider any portion of length at least k of the chain possessed by an honest party at the onset of a round; the ratio of blocks originating from the adversary is at most $1 - \mu$. We call μ the chain quality coefficient.*

3 The FairPoS protocol

We introduce the FairPoS model in parts. In Section 3.1, we formally define a novel notion of *input fairness*, for clients sending transactions to a FairPoS execution. In order for an encrypted input to reach the k-common-prefix, the duration implied by the delay parameter d must be sufficiently long.

In Section 3.2, we introduce the formal model of FairPoS, which extends δ-PoS with key extraction and a novel *longest-extractable-chain* selection rule. Here, encrypted inputs are generated as in Section 3.1 and given by the environment $\mathcal{Z}$ to parties executing the protocol. We first describe a naive application of key extraction, demonstrating adversarial chain stalls up to $d-1$ slots, thereby motivating the design of the *longest-extendable-chain* selection rule in the FairPoS model, which mitigates such adversarial impedance and ensures honest chain growth *independent* of chosen delay encryption parameter d.

A security analysis of FairPoS follows in Section 4, where the precise relationship between input fairness, chain growth, common-prefix and chain quality properties is formalized.

3.1 Input fairness & input encryption

Definition 6 (Input fairness, IF). *Consider the chain $\mathcal{C}$ possessed by an honest party at the onset of a round, where k-CP holds true. Input fairness holds if for all blocks $B \in \mathcal{C}^{\lceil k}$: 1. the adversary cannot decrypt an encrypted input in B before B is in the common-prefix; 2. encrypted inputs in B are eventually decrypted by all honest parties.*

Input fairness is conditioned on k-common-prefix property in FairPoS. Intuitively, the extraction delay d in FairPoS must be parameterized, such that the encrypted input can reach the common-prefix before d time passes. For simplicity, we denote d as time in slots. Note that input fairness permits an encrypted input to *not* become finalized and decrypted by the adversary: we argue this outcome is acceptable as the client transaction is not executed and thus cannot be exploited in any input ordering attacks. This is consistent with [5,26,27].

We sketch the input encryption procedure for FairPoS shown in Figure 2, where the environment $\mathcal{Z}$ provides the plain-text input for a party to encrypt and sign. For a block B, inputs are encrypted to a session id which is set to the chain tip that B is extending, such that $\mathsf{id} = \mathcal{C}.\mathsf{tip}$ and $\mathcal{C}.\mathsf{tip} = B.\mathsf{st}$. To ensure that a slot leader cannot insert an encrypted input to a *later* block, potentially deferring its insertion until the key extraction is completed, we ensure that the input is *bound* to a child block of $\mathcal{C}.\mathsf{tip}$ with a signature.

In the adaptive corruption setting, we deploy an efficient key evolving signature scheme (KES) [6,20] as used in [13]. Such schemes evolve secret key material *forward* with each signature, thereby erasing any information that could be used to generate verifying signatures of past rounds (See Definition 10). An adaptive adversary could always corrupt a user who has just *broadcast* a newly delay encrypted and signed input; with static key material only, the adversary would learn the signature key and generate verifying signatures of the delay encrypted input to insert it into a block, potentially *after* decrypting the encrypted input.

For public verification of such signatures, we assume the presence of logical accounts for all parties, each associated with a public signature verification key inferred from the chain tip.

FairPoS input encryption procedure

Let $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ and $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ denote a key evolving signature scheme and a symmetric-key encryption scheme. Let the genesis block of a chain contain a delay encryption parameter $\mathsf{DE.pk}$ and a chain tip imply account keys $\{\mathsf{KES.vk}_i\}_{i\in[n]}$.

Gen: Upon $(\textsc{Gen})$, set $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^k, T)$, return vk.

Sign: Upon $(\textsc{Sign}, \mathcal{C}, \mathsf{in})$

```
1. Let $\mathsf{id} = \mathcal{C}.\mathsf{tip}$. Assert $\mathsf{vk} \in \mathsf{accts}(\mathcal{C}.\mathsf{tip})$. Set $\mathsf{pk} \leftarrow \mathsf{DE.pk}(\mathcal{C}_0)$
2. Compute $(c, k) \leftarrow \mathsf{DE.Encaps}(\mathsf{pk}, \mathsf{id})$.
3. Encrypt input with key $k$: $m \leftarrow \mathsf{SKE.Enc}_k(\mathsf{in})$.
4. Generate $\sigma \leftarrow \mathsf{KES.Sign}_{\mathsf{sk}}(c \,|\, m \,|\, \mathsf{id})$.
5. Set $\mathsf{sk} \leftarrow \mathsf{Update}(\mathsf{KES.sk})$, thereby erasing signing previous key.
6. Return $(c, m, \sigma)$.
```

Fig. 2. Input encryption in FairPoS

3.2 The (d, δ, Δ)-FairPoS protocol

A key contribution of FairPoS is to achieve security that is independent of the chosen delay encryption parameter d. Towards this, we reintroduce Δ-monotonicity from [13], which states that an honest chain tip can always be extended by the first honest leader following Δ slots, a key property required to realize security in longest-chain proof-of-stake.

Definition 7 (Δ-Monotonicity, from [13]). *Let $\mathcal{T} = \bigcup_{i\in\mathcal{H}} \mathcal{T}^{(i)}$ be an honest tree rooted in genesis resulting from an execution of protocol π in a δ-synchronous network: it consists of all chains broadcast by honest parties. Further, let $\mathsf{depth}(i)$ denote the length of the chain extended by the uniquely honest leader of slot i. $\mathcal{T}$ exhibits the Δ-monotonicity property if*

$$\textit{For all uniquely honest slots } (i, j) \textit{ s.t. } j \geq i + \Delta : \mathsf{depth}(j) > \mathsf{depth}(i)$$

In PoS executed in a δ-synchronous setting , δ-monotonicity is trivially achieved: any honest block tip must arrive in the view of other honest party after δ slots, and is thus considered as a chain candidate for extension by any honest leader applying the longest chain selection rule.

Towards achieving Δ-monotonicity when extending PoS with key extraction with delay d, such that d can independently be parameterized from Δ, we first illustrate a naive key extraction application where this is not achieved to motivate the final FairPoS design.

Here, we must introduce a formal notion of *receipt delay*, which, informally, quantifies how far a local key extraction process is "behind schedule" due to adversarial delays.

Definition 8 (Receipt delay). *Let $\mathbf{r}^{(i)}(B) : \mathcal{B} \rightarrow \mathbb{Z}_0$ be the delay in slots between $B.\mathsf{sl}$ and the local arrival of block B from the view of the party (i). If B is an empty-block $(\perp)$, we define the receipt delay to be $\perp$.*

When the extraction window is defined as a slot interval preceding genesis, where slot indices are "negative", let the receipt delay is defined as $\perp$. We allow a receipt delay of $\perp$ to be interpreted as a receipt delay of 0.

Naive key extraction causing honest chain stall. Let an extraction schedule be defined as $D = d + \delta$, where d is the delay encryption parameter in slots, and δ be the maximum network delay δ. We initially assume a leader election sequence with no empty slots.

For an honest leader of slot t extending chain $\mathcal{C}$ (according to the longest-chain rule), let $\mathcal{C}_{t-D}$ denotes the block in $\mathcal{C}$ with a key extraction due in the honest block of slot t extending $\mathcal{C}$. In order to ensure that *current and future* slot leaders can extend a chain $\mathcal{C}$, the following must hold.

1. The honest slot leader must receive $\mathcal{C}_{t-D}$ *on time* (latest at slot $t - D + \delta$).
2. The honest slot leader must ensure that blocks in $\mathcal{C}$, which are extracted by *future* honest leaders, must also arrive *on time*, e.g. $(\mathcal{C}_{t-D+1}, ..., \mathcal{C}_{t-1})$.

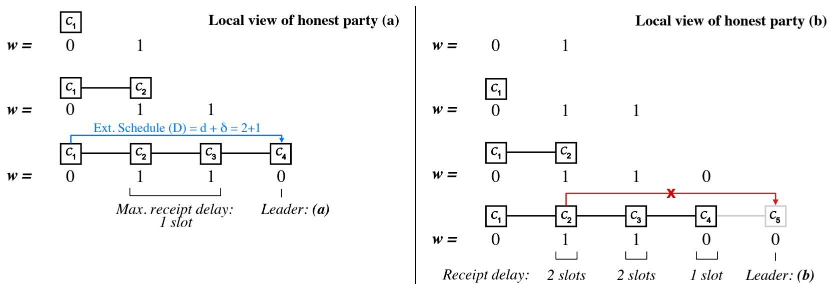

Local view of honest party (a)
w =
0
1
Ext. Schedule (D) = d + δ = 2+1
Max. receipt delay: 1 slot
Leader: (a)
Local view of honest party (b)
Receipt delay:
2 slots
2 slots
1 slot
Leader: (b)

Fig. 3. Example: Naive application of key extraction

Let the maximum network delay of a single slot be $\delta = 1$ in Figure 3. Consider honest party (a)'s view shown in the figure below, who is the leader of slot 4. Party (a) asserts that:

1. It can extract $\mathcal{C}_1$ by slot $\mathcal{C}_4$ given extraction schedule $D = 2 + 1 = 3$.
2. Blocks $\mathcal{C}_2$ and $\mathcal{C}_3$ were received in party (a)'s view with a *receipt delay* of no more than $\delta = 1$ slot.

Recall, condition (2) is intended to ensure that future honest leaders will be able to perform extractions of $\mathcal{C}_2, \mathcal{C}_3$ on time. However, the adversarial blocks $\mathcal{C}_{2,3}$ are only forwarded to party (a) with the maximally permitted delay $\delta = 1$, who then rebroadcasts (see M1 in Figure 1) to other honest parties:

$$\mathcal{A} \xrightarrow{\mathcal{C}_{2,3}} \mathcal{P}_{(a)} \xrightarrow{\mathcal{C}_{2,3}} \{\mathcal{P}_i\}_{i \in \mathcal{H}} \tag{3}$$

This inflicts an *additional* receipt delay on $\mathcal{C}_{2,3}$ in the view of other honest parties, such as party (b), the leader of slot 5. In the view of party (b),

1. It *cannot* extract $\mathcal{C}_2$ by slot 5 as the extraction of $\mathcal{C}_2$ is only 1/3 complete.
2. Blocks $\mathcal{C}_3$ and $\mathcal{C}_4$ were received in party (b)'s view with a *receipt delay* that *exceeds* 1 slot.

Thus, we have a honest chain stall: any honest party other than party (a) cannot extend honest chain tip $\mathcal{C}_4$ until slot 6, inflicting a chain stall of $d-1$.

Key extraction and longest-extendable-chains. We provide an informal overview to key extraction in FairPoS. Consider the view of honest party (a) shown in Figure 4, who is the leader of slot t. Let the protocol be executed in a 1-synchronous setting. Honest party (a), considering the extension of the chain $\mathcal{C}$, asserts the following:

1. It has completed the extraction of blocks in the *extraction window* with an *extraction schedule* $D = 12$ slots.
2. All blocks in the chain with *pending extractions* must have been received in party (a)'s view with maximum receipt delays shown in Figure 4.

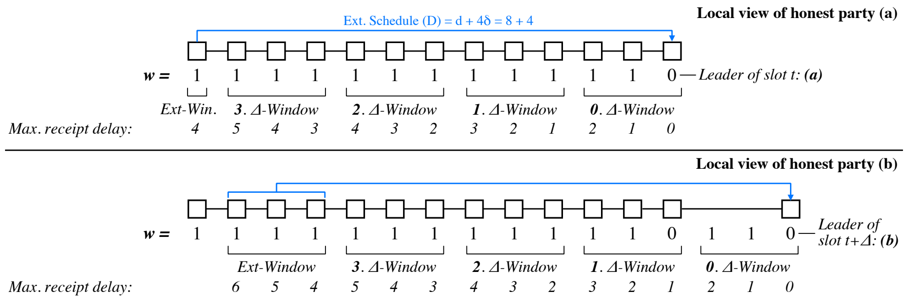

Ext. Schedule (D) = d + 4δ = 8 + 4
Local view of honest party (a)
w = 1 1 1 1 1 1 1 1 1 1 1 1 0
Leader of slot t: (a)
Ext-Win.
3. Δ-Window
2. Δ-Window
1. Δ-Window
0. Δ-Window
Max. receipt delay: 4 5 4 3 4 3 2 3 2 1 2 1 0
Local view of honest party (b)
w = 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0
Leader of slot t+Δ: (b)
Ext-Window
3. Δ-Window
2. Δ-Window
1. Δ-Window
0. Δ-Window
Max. receipt delay: 6 5 4 5 4 3 4 3 2 3 2 1 2 1 0

Fig. 4. Key extraction in $(d, \delta, \varDelta)$-FairPoS

Maximally permitted receipt delays are specific to each $\varDelta$-window, which divides the D slots of the extraction schedule. For the n'th $\varDelta$-window, the maximum permitted receipt delay is $n \cdot \delta = n \cdot 1$ for the first slot to the right of the window. Each subsequent slot in the n'th window is permitted an additional receipt delay of 1 slot. Observe that $\varDelta$-monotonicity holds for $\varDelta$=3: consider party (b), which is the leader of slot $t + \varDelta = t + 3$. Even if the adversary can induce an additional receipt delay of $\delta = 1$ in the view of party (b) as shown in Figure 4, party (b) will be able to assert conditions (1) and (2) above. This is because each block in the n'th $\varDelta$-window of party (a) is now in the $n+1$'th $\varDelta$-window of party (b)'s view and is permitted an additional delay of $\delta = 1$.

Extraction window. We define an extraction window as the the slots for which associated blocks have key extractions that are due in the current slot. This accounts for the possibility of a *gap* between the chain tip and the current slot, where no extracted keys could have been inserted.

Let the FairPoS protocol be parameterized with extraction schedule D. We define $\mathsf{gap2tip}(t, \mathcal{C})$ as the number of *empty* slots between slot t and $\mathcal{C}.\mathsf{tip}.\mathsf{sl}$.

$$\mathsf{gap2tip}(t, \mathcal{C}) = t - \mathsf{tip}(\mathcal{C}).\mathsf{sl} \quad \text{for} \quad t > \mathsf{tip}(\mathcal{C}).\mathsf{sl}$$

Then, the extraction window of current slot t and chain $\mathcal{C}$ with extraction schedule D can be defined as the range of slots, which are at least D slots in the past and are associated with blocks in $\mathcal{C}$ *without* key extractions already inserted in $\mathcal{C}$ due to the absence of blocks.

$$\mathsf{extWin}_D(t, \mathcal{C}) = (t - D - \mathsf{gap2tip}(t, \mathcal{C}) : t - D\,] \tag{4}$$

Extendable chains. Let $(d, \delta, \varDelta)$-$\mathsf{FairPoS}$ be parameterized by delay encryption parameter $d > 0$, maximum network delay $\delta \geq 0$ and desired monotonicity parameter $\varDelta > \delta$ (Definition 7), such that d divides $\varDelta - \delta$. If this holds, then we can define n, the number of $\varDelta$-windows which divide extraction schedule D.

$$D = n\varDelta \quad \text{where} \quad n = d/(\varDelta - \delta) \quad \text{s.t.} \quad n \in \mathbb{Z} \tag{5}$$

The delay extraction schedule D can be expanded to $D = n\varDelta = d + n\delta$, as the right equality is implied by $n = d/(\varDelta - \delta)$ from Equation 5.

We define the m'th "$\varDelta$-window" of current slot t as the following interval, consistent with Figure 4.

$$\varDelta\mathsf{Win}_D(t, m) = (t - (m+1)\varDelta : t - m\varDelta] \quad \text{for} \quad m \in [0 : \frac{D}{\varDelta}) \tag{6}$$

We formalize *chain extendability.* A chain $\mathcal{C}$ is extendable by leader of slot t with local receipt delay view $\mathbf{r}^{(i)}$ if the conditions ① and ② in Equation 7 hold.

$$\begin{gathered}\mathsf{ext}(t, \mathcal{C}, \mathbf{r}^{(i)}) = \begin{cases} 1 & ① \wedge ② \\ 0 & \text{otherwise} \end{cases} \\ ① \ \forall m \in [0 : \tfrac{D}{\varDelta}) : \forall j \in \varDelta\mathsf{Win}_D(t, m) : \mathbf{r}^{(i)}(\mathcal{C}_j) \leq m\delta + (t - m\varDelta - j) \\ ② \qquad \forall j \in \mathsf{extWin}_D(t, \mathcal{C}) : \mathbf{r}^{(i)}(\mathcal{C}_j) \leq \tfrac{D}{\varDelta}\delta + (t - D - j)\end{gathered} \tag{7}$$

Observe that extendability holds for the respective chain in the view of parties (a) and (b) in Figure 4 .

Theorem 1. *($\varDelta$-Monotonicity of $\mathsf{FairPoS}$) Every protocol execution of $(d, \delta, \varDelta)$-$\mathsf{FairPoS}$ results in an honest tree $\mathcal{T}$ that exhibits the $\varDelta$-monotonicity property.*

Longest-extendable-chain selection. In $\mathsf{FairPoS}$, an honest slot leader choses to extend the *longest extendable chain* in its local tree view $\mathcal{T}^{(i)}$.

$$\mathsf{maxExtChain}(t, \mathcal{T}^{(i)}, \mathbf{r}^{(i)}) = \underset{\mathcal{C} \in \mathcal{T}^{(i)} : \mathsf{ext}(t, \mathcal{C}, \mathbf{r}^{(i)})}{\arg\max} \mathsf{len}(\mathcal{C}) \tag{8}$$

The formal model of $(d, \delta, \varDelta)$-$\mathsf{FairPoS}$ is stated in Figure 5, which extends the δ-PoS in Figure 1 with the longest-extendable-chain selection rule, key extractions and the insertion of delay encrypted inputs, generated by the procedure in Figure 2. The protocol execution state of $(d, \delta, \varDelta)$-$\mathsf{FairPoS}$ is extended with local receipt delays:

$$\varGamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i \in \mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right) \tag{9}$$

(d, δ, Δ)-FairPoS execution model

The evolution of a (d, δ, Δ)-FairPoS execution state $\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i \in \mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right)$ in a single execution round induced by environment $\mathcal{Z}$, adversary $\mathcal{A}$ and characteristic string w adheres to **T1-T3** and **M1-M2** of PoS execution model (Figure 1) with the following differences:

T2' differs from **T2** of PoS in the following:
- *Longest-extendable-chain selection* replaces *longest-chain-selection.*
- **Extend'** procedure replaces **Extend** for honest parties and $\mathcal{A}$.

The receipt delays $\{\mathbf{r}^{(i)}\} i \in \mathcal{H}$ in Γ_t evolve as follows:

R1: For each chain $\mathcal{C}$ delivered to honest party i by $\mathcal{A}$ at slot $t+1$: for each newly seen block $\mathcal{C}_j \in \mathcal{C}$, party i records $\mathbf{r}^{(i)}(\mathcal{C}_j) \leftarrow t + 1 - \mathcal{C}_j.\mathsf{sl}$.

Each honest party and the adversary must perform:

E1: In each round, exactly a single Extract step on each block in its local view for which key extraction is pending.

Extend': Party i generates $B = (t, H(\mathcal{C}.\mathsf{tip}), d_{\mathsf{ext}} | d_{\mathsf{ins}}, P_i)$, where:
- $d_{\mathsf{ext}} = (\mathsf{idk}, \mathsf{idk}', ...)$ are extractions of blocks in $\mathcal{C}$ associated with $\mathsf{extWin}_D(t, \mathcal{C})$.
- $d_{\mathsf{ins}} = \{(c_j, m_j, \sigma_j)\}_{j \in [n]}$ is an *ordering* of encrypted inputs from $\mathcal{Z}$ for slot t. When party i receives d_{ins}, it asserts for $\mathsf{id} = \mathcal{C}.\mathsf{tip}$ and each entry in d_{ins}:
  - $\exists \mathsf{vk}_j \in \mathsf{accts}(\mathcal{C}.\mathsf{tip}) : 1 \leftarrow \mathsf{KES.Verify}_{\mathsf{vk}_j}((c_j | m_j | \mathsf{id}), \sigma_j)$.

Then, the party adds $\mathcal{C}' = \mathcal{C} | B$ to its local view.

Fig. 5. FairPoS execution

Views in initial state Γ_0 contain a genesis block which includes public parameters $\mathsf{DE.pk}, \mathsf{DE.ek}$. Importantly, we *require* the adversary $\mathcal{A}$ and each honest party to perform exactly one extraction step for each pending key extraction in each round of the execution. Thus, no party or adversary gains a time advantage in extracting session keys from blocks (See E1 in Figure 5).

4 FairPoS security

4.1 Common-prefix in FairPoS

Demonstrating k-common-prefix in FairPoS is accomplished by formally relating the tree views generated in an execution of (d, δ, Δ)-FairPoS with those that could have resulted from δ-PoS.

Let the *honest tree* of protocol execution state $\Gamma_t = \left(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}, \mathbf{r}^{(i)}\}_{i \in \mathcal{H}}, \mathcal{T}^{\mathcal{A}}\right)$ be given be the union of honest tree views at slot t:

$$\mathcal{T}^{\mathcal{H}}(\Gamma_t) = \bigcup_{i \in \mathcal{H}} \mathcal{T}^{(i)} \tag{10}$$

In the analysis of PoS [13,24], the branching structure of the honest tree informs us about events where a local chain was abandoned for a longer, alternative branch according to the *longest chain* selection rule. Informally, the common-prefix property is violated if the k-common-prefix shared between prior and newly adopted chains if their shared prefix is "too short". We begin our analysis with the (viable) branches which could have been adopted by honest parties in the first place.

Viable branches in PoS. A viable branch $\mathcal{B}$ in a tree $\mathcal{T}$ must exceed all honest chains-tips generated more than δ slots prior to $\mathcal{B}$.tip.slot in length: this property arises from the honest application of the *longest chain rule*. The honest leader of $\mathcal{B}$.tip.slot must have received any honest chain-tips generated δ slots prior and considered them as *alternative candidates* for extension. Therefore, the resulting, *viable* branch $\mathcal{B}$ must exceed these in length. For branch $\mathcal{B} \in \mathcal{T}$, we formalize this set of alternative chain candidates as

$$\mathsf{altChns}_\delta(\mathcal{B}, \mathcal{T}) = \{\, \mathcal{C} \subseteq \mathcal{T} \mid \mathcal{C}.\mathsf{tip.ldr} \in \mathcal{H} \ \wedge\ \mathcal{B}.\mathsf{tip.sl} > \mathcal{C}.\mathsf{tip.sl} + \delta \,\} \tag{11}$$

For a PoS protocol execution state $\varGamma_t = \big(\{\mathcal{T}^{(i)}, \mathbf{m}^{(i)}\}_{i\in\mathcal{H}}, \mathcal{T}^{\mathcal{A}}\big)$, the set of well-formed, viable branches is formalized as the set of branches with lengths exceeding their honest, alternative chains.

$$\mathsf{viableBranches}^{\mathsf{PoS}}_\delta(\varGamma_t) = \{\forall \mathcal{B} \in \mathcal{T}^{\mathcal{H}}(\varGamma_t) \mid \mathsf{len}(\mathcal{B}) > \underset{\mathcal{C}\,\in\,\mathsf{altChns}_\delta(\mathcal{B},\mathcal{T}^{\mathcal{H}}(\varGamma_t))}{\arg\max} \mathsf{len}(\mathcal{C})\ \} \tag{12}$$

Viable chains in FairPoS. The notion of viable branches must be strengthened for FairPoS since the *longest-extendable-chain* rule introduces additional constraints for the adoption of a chain in the local honest tree view. Let the *extendable prefix* of a branch $\mathcal{B}$ in the view of honest parties at slot t be defined as the "longest extendable prefix" of a branch.

$$\mathsf{extPrefix}(t, \mathcal{B}, \{\mathbf{r}^{(i)}\}_{i\in\mathcal{H}}) = \underset{\mathcal{C}\preceq\mathcal{B}\,:\,\exists i\in\mathcal{H}\,:\,\mathsf{ext}(t,\mathcal{C},\mathbf{r}^{(i)})}{\arg\max} \mathsf{len}(\mathcal{C}) \tag{13}$$

For a $(d, \delta, \varDelta)$-FairPoS state, let the set of viable chains be defined as the extendable prefixes (Equation 13) of branches in the honest tree with lengths which exceed those of its alternative chains (Equation 11) generated $\varDelta$ slots prior: by the $\varDelta$-monotonicity property (Theorem 1), these chains must have been extendable by the leader that generated the respective prefix and considered as candidates for extension.

$$\begin{aligned}\mathsf{viableChains}^{\mathsf{FairPoS}}_\varDelta(\varGamma_t) = \{\, \mathcal{C} \subseteq \mathcal{T}^{\mathcal{H}}(\varGamma_t) \mid \exists \mathcal{B} \in \mathcal{T}^{\mathcal{H}}(\varGamma_t) : \mathcal{C} = \mathsf{extPrefix}(t, \mathcal{B}, \{\mathbf{r}^{(i)}\}_{i\in\mathcal{H}})\\ \wedge\ \mathsf{len}(\mathcal{C}) > \underset{\mathcal{C}'\,\in\,\mathsf{altChns}_\varDelta(\mathcal{C},\mathcal{T}^{\mathcal{H}}(\varGamma_t))}{\arg\max} \mathsf{len}(\mathcal{C}')\ \}\end{aligned} \tag{14}$$

We restate the *divergence* notion from [13,24] which formally describes the *magnitude* of branching caused by the switching between viable chains.

Definition 9 (Divergence). *For two chains $\mathcal{C}_1$ and $\mathcal{C}_2$, define their divergence to be the quantity*

$$\mathsf{div}(\mathcal{C}_1, \mathcal{C}_2) = \mathsf{min}(\mathsf{len}(\mathcal{C}_1), \mathsf{len}(\mathcal{C}_2)) - \mathsf{len}(\mathcal{C}_1 \cap \mathcal{C}_2)$$

where $\mathcal{C}_1 \cap \mathcal{C}_2$ denotes the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$. We extend this notion of divergence to the protocol execution state Γ resulting from the execution of protocol π induced by characteristic w in the δ-synchronous setting: here, the maximum divergence over any two viable chains is quantified.

$$\mathsf{div}^{\pi}_{\delta}(\Gamma) = \mathsf{max}_{\mathcal{C}_1,\mathcal{C}_2 \in \mathsf{viableBranches}^{\pi}_{\delta}(\Gamma)} \mathsf{div}(\mathcal{C}_1, \mathcal{C}_2)$$

Finally, we define the divergence of a characteristic string w to be the maximum divergence observable over all states which could have resulting from protocol executions induced by w. More formally, let $\mathsf{exec}^{\pi}_{\delta}(\Gamma_0, w)$ denote all possible executions of π beginning with state Γ_0 which could have been induced by w in δ-synchronous network. Then the divergence of a characteristic string w is defined as:

$$\begin{gathered}\mathsf{div}^{\pi}_{\delta}(w) = \mathsf{max}_{\Gamma \in \mathsf{reachable}^{\pi}_{\delta}(\Gamma_0, w)} \mathsf{div}^{\pi}_{\delta}(\Gamma) \\ \textit{where } \mathsf{reachable}^{\pi}_{\delta}(\Gamma_0, w) = \{\Gamma \mid \exists \lambda \in \mathsf{exec}^{\pi}_{\delta}(\Gamma_0, w) : \Gamma_0 \xrightarrow{\lambda} \Gamma\}\end{gathered} \tag{15}$$

For PoS, the probability that that the divergence exceeds k-blocks over an execution of R, is given by the following theorem from [13].

Theorem 2. *(PoS Divergence, Theorem 4 in [13]) Let active slot coefficient $f \in (0, 1]$ and α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for some $\epsilon > 0$. Further, let w be a string drawn from $\{0, 1, \perp\}^R$ according to D^f_{α}. Then we have $\Pr_{w \leftarrow\$ D^f_{\alpha}}[\mathit{div}^{PoS}_{\Delta}(w) \geq k] \leq \exp(\ln(R) - \Omega(k - \Delta))$.*

Towards demonstrating k-common prefix in FairPoS, we first present a central theorem, which states that for all executions of (d, δ, Δ)-FairPoS in the δ-synchronous setting, there exists an execution of PoS in the Δ-synchronous setting, such that viable chains of the d, δ, Δ-FairPoS honest tree are *equivalent* ($\equiv$) to the viable branches of the PoS honest tree: here, we define the equivalence of chains such that only their structural properties are considered, formally stated in Definition 11.

Theorem 3. (Equivalent trees) For any (d, δ, Δ)-FairPoS execution λ induced by a charactistic string $w \in \{0, 1, \perp\}^*$, $\Gamma_0 \rightarrow^{\lambda} \Gamma$, there exists a Δ-PoS execution λ' induced by same w, $\Gamma'_0 \rightarrow^{\lambda'} \Gamma'$, such that the viable chains in Γ are equivalent to the viable branches in Γ'.

$$\begin{gathered}\forall w \in \{0, 1, \perp\}^* : \forall \lambda \in \mathsf{exec}^{\mathsf{FairPoS}}_{\delta}(\Gamma_0, w), \Gamma_0 \xrightarrow{\lambda} \Gamma : \exists \lambda' \in \mathsf{exec}^{\mathsf{PoS}}_{\Delta}(\Gamma'_0, w), \Gamma_0 \xrightarrow{\lambda'} \Gamma' : \\ \mathsf{viableChains}^{\mathsf{FairPoS}}_{\delta}(\Gamma) \equiv \mathsf{viableBranches}^{\mathsf{PoS}}_{\Delta}(\Gamma')\end{gathered}$$

Since divergence is defined over viable chains and viable branches, we can infer Corollary 1 from Theorem 3.

Corollary 1. $\forall w \in \{0, 1, \perp\}^* : \mathit{div}^{\mathit{FairPoS}}_{\delta}(w) \leq \mathit{div}^{\mathit{PoS}}_{\Delta}(w)$

This allows us to infer k-common-prefix from bounding the probability of the event $\mathsf{div}^{\mathsf{PoS}}_{\Delta}(w) > k$ for $w \leftarrow\$ \mathcal{D}^f_{\alpha}$ in PoS [13].

Theorem 4. (k-Common prefix in FairPoS) *Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. The probability that $\mathcal{A}$ makes the protocol violate the k-common-prefix property in a δ-synchronous environment throughout a period of R slots is no more than* $\exp(\ln R + \Delta - \Omega(k))$.

4.2 Chain growth, chain quality and input fairness

Both chain growth and chain quality of FairPoS can be derived from Δ-monotonicity of FairPoS (Theorem 1) and probabilities bounding security failure from PoS [13].

Theorem 5. ((τ, s)-Chain growth in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1-f)\Delta = (1+\epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes the protocol violate the chain growth property with parameters $s \geq 4\Delta$ and $\tau = c\alpha/4$ throughout a period of R slots, is no more than $\exp(-c\alpha s/(20\Delta) + lnR\Delta + O(1))$, where c denotes the constant $c := c(f, \Delta) = f(1 - f)^{\Delta}$.

Theorem 6. ((μ, k)-Chain quality in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1 - \alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes FairPoS violate the chain quality property with parameters k and $\mu = 1/k$ throughout a period of R slots, is no more than $\exp(\ln R - \Omega(k))$.

Input fairness is obtained from chain growth, common prefix and chain quality. Informally, given time d and chain growth rate τ, we can determine common-prefix and chain quality parameters such that a decrypted input must have sufficient time (d slots) to reach finalization or lie in an abandoned chain.

Lemma 1. (Input fairness from CG, CP and CQ in (d, δ, Δ)-FairPoS) If for an execution of (d, δ, Δ)-FairPoS, (τ, d)-chain growth, $(d\tau(\tau - \delta/(\Delta - \delta)) - 1)$-common prefix, and $(1/(D+1), D+1)$-chain quality hold, where $D = d\Delta/(\Delta - \delta)$, then input fairness is implied.

Theorem 7. (Input fairness in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then, the probability that $\mathcal{A}$ makes the FairPoS violate the input fairness property falls exponentially with d.

We refer to Appendix C for a discussion of (d, δ, Δ)-FairPoS security for specific parameterizations of d, δ, Δ.

5 Conclusion

We contribute FairPoS, the first longest-chain, proof-of-stake protocol achieving input fairness against an adaptive adversary. When adopting the leader election procedure from Ouroborous Praos [13] or one that induces leader sequences (*i.e.* characteristic strings) consistent with the *dominant distribution* (Definition 1), FairPoS achieves input fairness in addition to the common-prefix, chain-growth and chain-quality properties of PoS [13]. We note that Kiayas *et al.* [24] provide tighter bounds for k-common prefix in PoS. Applying this updated analysis framework to security of FairPoS is planned as future work.

6 Acknowledgements

We thank Stefan Dziembowski and Sebastian Faust for exploratory discussions on mitigating front-running at the consensus protocol level. We thank Luca De Feo for his insights on isogeny-based cryptography[6] and his views on deploying Delay Encryption in practice.

[6] At the crypt@b-it 2022 summer school on cryptography.

References

1. Alliance, V.: VDF Alliance Official Wiki. https://supranational.atlassian.net/wiki/spaces/VA/overview (2022)
2. Avalanche: Apricot Phase Four: Snowman++ and Reduced C-Chain Transaction Fees. https://medium.com/avalancheavax/apricot-phase-four-snowman-and-reduced-c-chain-transaction-fees-1e1f67b42ecf (2021)
3. Badertscher, C., Gaži, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 913–930 (2018), https://doi.org/10.1007/978-3-319-78375-8_3
4. Bartoletti, M., Chiang, J.H.y., Lluch-Lafuente, A.: Maximizing extractable value from automated market makers. arXiv preprint arXiv:2106.01870 (2021), https://arxiv.org/pdf/2106.01870
5. Bebel, J., Ojha, D.: Ferveo: Threshold Decryption for Mempool Privacy in BFT networks. Cryptology ePrint Archive (2022), https://eprint.iacr.org/2022/898
6. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: CRYPTO'99 (Aug 1999)
7. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: TCC 2020, Part I (Nov 2020)
8. Blum, E., Kiayias, A., Moore, C., Quader, S., Russell, A.: Linear consistency for proof-of-stake blockchains. arXiv preprint arXiv:1911.10187 (2019), https://arxiv.org/abs/1911.10187
9. Burdges, J., Feo, L.D.: Delay encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 302–326. Springer (2021), https://doi.org/10.1007/978-3-030-77870-5_11
10. Cachin, C., Mićić, J., Steinhauer, N.: Quick Order Fairness. arXiv preprint arXiv:2112.06615 (2021), https://arxiv.org/abs/2112.06615
11. Cascudo, I., David, B., Garms, L., Konring, A.: YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint Archive, Report 2022/242 (2022), https://eprint.iacr.org/2022/242
12. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: IEEE Symposium on Security and Privacy. pp. 910–927. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00040, https://doi.org/10.1109/SP40000.2020.00040
13. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 66–98. Springer (2018), https://doi.org/10.1007/978-3-319-78375-8_3
14. De Feo, L., Masson, S., Petit, C., Sanso, A.: Verifiable delay functions from supersingular isogenies and pairings. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 248–277. Springer (2019), https://doi.org/10.1007/978-3-030-34578-5_10
15. Erwig, A., Faust, S., Riahi, S.: Large-scale non-interactive threshold cryptosystems through anonymity. Cryptology ePrint Archive, Report 2021/1290 (2021), https://eprint.iacr.org/2021/1290

16. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 281–310. Springer (2015), https://doi.org/10.1007/978-3-662-46803-6_10
17. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You only speak once - secure MPC with stateless ephemeral roles. In: CRYPTO 2021, Part II (Aug 2021)
18. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th symposium on operating systems principles. pp. 51–68 (2017), https://doi.org/10.1145/3132747.3132757
19. Goyal, V., Kothapalli, A., Masserova, E., Parno, B., Song, Y.: Storing and retrieving secrets on a blockchain. In: Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I (2022). https://doi.org/10.1007/978-3-030-97121-2_10
20. Itkis, G., Reyzin, L.: Forward-secure signatures with optimal signing and verifying. In: CRYPTO 2001 (Aug 2001)
21. Kelkar, M., Deb, S., Kannan, S.: Order-fair consensus in the permissionless setting. In: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop. pp. 3–14 (2022), https://doi.org/10.1145/3494105.3526239
22. Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.: Themis: Fast, Strong Order-Fairness in Byzantine Consensus (2021), https://eprint.iacr.org/2021/1465
23. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Annual International Cryptology Conference. pp. 451–480. Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_16
24. Kiayias, A., Quader, S., Russell, A.: Consistency of proof-of-stake blockchains with concurrent honest slot leaders. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). pp. 776–786. IEEE (2020), https://doi.org/10.1109/ICDCS47774.2020.00065
25. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual international cryptology conference. pp. 357–388. Springer (2017), https://doi.org/10.1007/978-3-319-63688-7_12
26. Malkhi, D., Szalachowski, P.: Maximal Extractable Value (MEV) Protection on a DAG. arXiv e-prints pp. arXiv–2208 (2022), https://arxiv.org/abs/2208.00940
27. Momeni, P.: Fairblock: Preventing blockchain front-running with minimal overheads. Master's thesis, University of Waterloo (2022), https://eprint.iacr.org/2022/1066
28. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996), http://bitsavers.trailing-edge.com/pdf/mit/lcs/tr/MIT-LCS-TR-684.pdf

A Key evolving signature schemes

We present the formal definitions of key evolving signature scheme of [6,20]

Definition 10. *A key evolving signature scheme* $\mathsf{KES} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Update})$ *is a tuple of algorithms such that:*

1. $\mathsf{Gen}(1^k, T)$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^k *and the total number of periods* T*, outputting a key pair* $(\mathsf{KES.sk}_1, \mathsf{KES.vk})$*, where* $\mathsf{KES.vk}$ *is the verification key and* $\mathsf{KES.sk}_1$ *is the initial signing key (we assume that the period* j *to which a signing key* $\mathsf{KES.sk}_j$ *corresponds is encoded in the signing key itself).*
2. $\mathsf{Sign}_{\mathsf{KES.sk}_j}(m)$ *is a probabilistic signing algorithm that takes as input a secret key* $\mathsf{KES.sk}_j$ *for the time period* $j \leq T$ *and a message* m*, outputting a signature* σ_j *on* m *for time period* j *(we assume that the period* j *for which a signature* σ_j *was generated is encoded in the signature itself).*
3. $\mathsf{Verify}_{\mathsf{KES.vk}}(m, \sigma_j)$ *is a deterministic verification algorithm that takes as input a public key* $\mathsf{KES.vk}$*, a message* m *and a signature* σ_j*, outputting* 1 *if* σ_j *is a valid signature on message* m *for time period* j *and* 0 *otherwise.*
4. $\mathsf{Update}(\mathsf{KES.sk}_j)$ *is a probabilistic secret key update algorithm that takes as input a secret key* $\mathsf{KES.sk}_j$ *for the current time period* j *and outputs a new secret key* $\mathsf{KES.sk}_{j+1}$ *for time period* $j+1$*. We define* $\mathsf{KES.sk}_{T+1}$ *as the empty string and set it as the output of* $\mathsf{Update}(\mathsf{KES.sk}_T)$*.*

Correctness: *for every key pair* $(\mathsf{KES.sk}_1, \mathsf{KES.vk}) \leftarrow \mathsf{Gen}(1^k, T)$*, every message* m *and every time period* $j \leq T$*,* $\mathsf{Verify}_{\mathsf{KES.vk}}(m, \mathsf{Sign}_{\mathsf{KES.sk}_j}(m)) = 1$*.*

B Chain equivalence

Definition 11 (Equivalence $\equiv$). *Two chains* $\mathcal{C}_0$ *and* $\mathcal{C}_1$ *are equivalent,* $\mathcal{C}_0 \equiv \mathcal{C}_1$*, if* $\mathcal{C}_0' = \mathcal{C}_1'$*, where* $\mathcal{C}'$ *is obtained from* $\mathcal{C}$ *with the following procedure:*

```
- Let $\mathcal{C} = (B_0, ..., B_R)$ and set $B_0' \leftarrow (0, \epsilon, \epsilon, 0)$, $\mathcal{C}' \leftarrow B_0'$,
- For $k \in [1, R]$:
    - $B_k' \leftarrow (B_k.\mathsf{sl}, H(B_{k-1}'), \epsilon, B_k.\mathsf{ldr})$
    - $\mathcal{C}' \leftarrow \mathcal{C}'|B_k'$
```

We lift this definition of equivalence to chain sets (or trees), where each chain in one set has exactly one equivalent chain in the other.

C Protocol parameterizations

We illustrate parameterizations of (d, δ, Δ)-FairPoS in Table 1 for discussion. Observe that (d, δ, Δ)-FairPoS can be parameterized with any d, δ and Δ such that Equation 5 holds. Delay parameter d can always be parameterized sufficiently long for any k-common prefix, since the probability that k-common prefix is violated falls exponentially with decreasing parameter Δ, which is chosen independently of d.

Note that a smaller Δ will increase security, as the Δ-monotonicity property will ensure faster honest chain growth. The extractions schedule D is a function of d, δ and Δ (Equation 5). Then, let the difference between the extraction schedule and delay parameter "$D-d$" be interpreted as the "time gap" between the moment session keys are extracted by parties and the slot when these will appear on chain. A shorter "$D-d$" interval implies that light-clients which are not performing key extractions will observe the chain-state sooner, as decryption keys are posted to the chain within a shorter time period.

d	δ	Δ	n	D	$D-d$
30	6	8	15	120	90
60	6	8	30	240	180
60	6	21	4	84	24
90	6	21	6	126	36

Table 1. Parameterizations of (d, δ, Δ)-FairPoS

We observe following trade-offs in Table 1: for a smaller Δ, which increases the security of FairPoS, we obtain a larger $D-d$, implying a larger lag in chain-state observability for non-extracting parties. This is intuitive, as a shorter Δ parameter implies that honest parties must "converge" sooner on extendable chains. This is achieved by permitting additional "slack" or "$D-d$" in the extraction schedule. We argue that this is necessary trade-off which can be acceptable in practice, as parties which need to decrypt inputs as soon as possible are likely to perform key extractions (e.g. DeFi market arbitrageurs). Even at larger "$D-d$" intervals, key extraction remains critical for practicality: newly joining parties will immediately obtain keys to decrypt the entire input history beginning from genesis from the blockchain itself.

D Proofs

Theorem 1. *(Δ-Monotonicity of FairPoS) Every protocol execution of (d, δ, Δ)-FairPoS results in an honest tree $\mathcal{T}$ that exhibits the Δ-monotonicity property.*

Proof (Theorem 1). Δ-monotonicity holds if every chain-tip that is honestly generated at slot i iconsidered a candidate for extension by the first honest leader at or following slot $i+\Delta$. In other words, the proof obligation is to demonstrate honest chain extendability (Equation 7) holds for every honest chain-tip, Δ slots following its generation. We prove this by induction:

Base case (Genesis). The genesis block at slot 0 is trivially extendable at any slot $\mathsf{sl} \geq \Delta$. Recall that we permit negative chain indices, e.g. $\mathcal{C}_{-j}$ for

$j \in \mathbb{Z}$, which denote "empty blocks", for which the local receipt delay $\mathbf{r}^{(i)}(\mathcal{C}_{-j})$ is always $\perp$, and by the definition of receipt delay (Definition 8) is interpreted as 0. Thus, when extending genesis, every "empty block" in the extraction window and the "m"th Δ-window will not violate the receipt delay bounds imposed by the *extendable-chain* criteria in Equation 7.

Inductive case (Slot i). A chain $\mathcal{C}$ honestly extended at slot i must fulfill the receipt delay constraints in Equation 7 in the view of the slot leader. This honest leader will have rebroadcast every block in $\mathcal{C}$ upon arrival: thus, in the worst case, all other honest parties will have received blocks in $\mathcal{C}$ with an additional receipt delay of δ slots compared to the leader of slot i. The following must hold for $\mathcal{C}$ in the view of any honest leader of slot $j \geq i + \Delta$:

- For $m \in [0 : n)$: each block of $\mathcal{C}$ in the "m"-th Δ-window (Equation 6) in the view of leader of slot i is now in the "$m+1$"-th Δ-window in the view of the honest leader of slot j, it is permitted an additional *worst-case* additional delay of δ by the extendable chain definition (Equation 7).
- Blocks in the "$n-1$"th window of $\mathcal{C}$ in the view of leader i are in the extraction window of leader of slot j, and are also permitted an additional *worst-case* delay of δ (Equation 7).

□

Theorem 3. (Equivalent trees) For any (d, δ, Δ)-FairPoS execution λ induced by a charactistic string $w \in \{0, 1, \perp\}^*$, $\Gamma_0 \to^{\lambda} \Gamma$, there exists a Δ-PoS execution λ' induced by same w, $\Gamma_0' \to^{\lambda'} \Gamma'$, such that the viable chains in Γ are equivalent to the viable branches in Γ'.

$$\forall w \in \{0,1,\perp\}^* : \forall \lambda \in \mathsf{exec}_{\delta}^{\mathsf{FairPoS}}(\Gamma_0, w), \Gamma_0 \xrightarrow{\lambda} \Gamma : \exists \lambda' \in \mathsf{exec}_{\Delta}^{\mathsf{PoS}}(\Gamma_0', w), \Gamma_0 \xrightarrow{\lambda'} \Gamma' :$$
$$\mathsf{viableChains}_{\delta}^{\mathsf{FairPoS}}(\Gamma) \equiv \mathsf{viableBranches}_{\Delta}^{\mathsf{PoS}}(\Gamma')$$

Proof (Theorem 3). Let $\mathsf{exec}_{\delta}^{\pi}(\Gamma, w_t)$ denote all possible *single round* executions of π in a δ-synchronous setting induced by $w_t \in \{0, 1, \perp\}$ at slot t, beginning with protocol state Γ. Further, we define the *honest extendable tree* of FairPoS state Γ_t as the union of the extendable prefixes (Equation 13) of all tree branches in the view of honest parties.

$$\mathcal{T}_{\mathsf{ext}}^{\mathcal{H}}(\Gamma_t) = \bigcup_{\mathcal{C} \in \mathcal{T}^{\mathcal{H}} : \exists \mathcal{B} \in \mathcal{T}^{\mathcal{H}} : \mathcal{C} = \mathsf{extPrefix}(t, \mathcal{B}, \{\mathbf{r}^{(i)}\}_{i \in \mathcal{H}})} \mathcal{C}$$

For a FairPoS state Γ_t and PoS state Γ_t' where $\mathcal{T}_{\mathsf{ext}}^{\mathcal{H}}(\Gamma_t) \equiv \mathcal{T}^{\mathcal{H}}(\Gamma_t')$ we observe that viable branches of Γ_t and viable chains of Γ_t' converge by definition.

$$\mathsf{viableChains}_{\delta}^{\mathsf{FairPoS}}(\Gamma_t) \equiv \mathsf{viableBranches}_{\Delta}^{\mathsf{PoS}}(\Gamma_t')$$

We prove the theorem round-wise, by induction.

Base step $(0 \to 1)$. For the first round executed on the genesis block (slot 0) and given a characterstic string w we must prove:

$$\forall\, r \in \mathsf{exec}_{\delta}^{\mathsf{FairPoS}}(\Gamma_0, w_1)\,,\; \Gamma_0 \xrightarrow{r} \Gamma_1 : \exists\, r' \in \mathsf{exec}_{\Delta}^{\mathsf{PoS}}(\Gamma'_0, w_1)\,,\; \Gamma'_0 \xrightarrow{r'} \Gamma'_1 \\ \mathcal{T}^{\mathcal{H}}_{\mathsf{ext}}(\Gamma_1) \equiv \mathcal{T}^{\mathcal{H}}(\Gamma'_1) \tag{16}$$

We describe the translation from r to r'. If w_1 is honest, then the elected honest parties will generate a block extending genesis in the rounds of both protocols (genesis is immediately extendable, as no key extractions are due in the block associated at slot 1). If w_1 is dishonest, then whatever the blocks the adversary generates in r is performed by the adversary in r': resulting extendable honest tree in Γ_1 and honest tree in Γ'_1 must be equivalent.

Induction step $(t \to t+1)$. We must prove

$$\forall\, r \in \mathsf{exec}_{\delta}^{\mathsf{FairPoS}}(\Gamma_t, w_{t+1})\,,\; \Gamma_t \xrightarrow{r} \Gamma_{t+1} : \exists\, r' \in \mathsf{exec}_{\Delta}^{\mathsf{PoS}}(\Gamma'_t, w_{t+1})\,,\; \Gamma'_t \xrightarrow{r'} \Gamma'_{t+1} \\ \mathcal{T}^{\mathcal{H}}_{\mathsf{ext}}(\Gamma_{t+1}) \equiv \mathcal{T}^{\mathcal{H}}(\Gamma'_{t+1}) \tag{17}$$

By induction hypothesis it holds that this holds that $\mathcal{T}^{\mathcal{H}}_{\mathsf{ext}}(\Gamma_t) \equiv \mathcal{T}^{\mathcal{H}}(\Gamma'_t)$. For any honest or adversarial action in round r, we illustrate the respective action in round r', before arguing the equivalence of extendable honest trees in Γ_{t+1} and honest trees in Γ'_{t+1}.
Honest actions in FairPoS round r are also performed in PoS round r':

H1 If w_{t+1} is a uniquely honest slot, the longest (extendable) chains will be equivalent in the honest party's view, and the honest leader(s) will extend the equivalent chains of both protocol executions in rounds r and r' respectively. (By induction hypothesis, the the honest extendable tree in Γ_t is equivalent to the honest tree in Γ'_t). A newly extended chain from honest party i is added to the message queue $\mathbf{m}^{(i)}$ and $\mathcal{T}^{\mathcal{A}}$ in both r and r'.

Adversarial actions include dishonest block generation (T2 in Figure 1) and the delivery of messages (M2 in Figure 1). The translation of adversarial actions from r round to the r' is described.

A1 If the adversary adds dishonest blocks to $\mathcal{T}^{\mathcal{A}}$ at any adversarial slot $t' \leq t+1$ in the round r, this action is performed in r' round.

A2 If the adversary delivers a $\mathcal{C}$ with adversarial chain tip from $\mathcal{T}^{\mathcal{A}}$ to $\mathcal{T}^{(i)}$ in r:
   a. If $\mathcal{C}$ is extendable by party i in state Γ_{t+1} following r, $\mathcal{A}$ delivers equivalent $\mathcal{C}'$ from $\mathcal{T}^{\mathcal{A}\,\prime}$ to $\mathcal{T}^{(i)\prime}$ in the r' round: $\mathcal{C}, \mathcal{C}'$ must both exist in the adversarial tree views in states Γ_t, Γ'_t due to **A1**.
   b. Else if $\mathcal{C}$ *remains* unextendable in state Γ_{t+1} by party i, equivalent $\mathcal{C}'$ is not added to the equivalent honest tree $\mathcal{T}^{(i)\prime}$ in r'.

A3 If the adversary delivers a chain $\mathcal{C}$ from message queue $\mathbf{m}^{(i)}$ to an honest tree view $\mathcal{T}^{(i)}$ in r:
   a. If $\mathcal{C}$ is extendable by party i in state Γ_{t+1} following r, $\mathcal{A}$ delivers equivalent $\mathcal{C}'$ to $\mathcal{T}^{(i)\prime}$ in the r' round.

  b Else if $\mathcal{C}$ is not extendable by party i in state Γ_{t+1}, its equivalent $\mathcal{C}'$ is not added to the honest tree in round r': since $\Delta > \delta$, it is always possible for $\mathcal{A}$ defer an honest chain delivery for a longer delay in PoS (Δ-synchrony) than in the ΔPoS (δ-synchrony) execution.

A4 If any $\mathcal{C} \in \mathcal{T}^{(i)}$ in an honest tree view *becomes* extendable in slot $t+1$ following round r in the view of party i.

  a. If $\mathcal{C}$ and equivalent $\mathcal{C}'$ feature an honest chain tip it must be present in an honest message queue $\mathbf{m}^{(j)\prime}$ of the PoS execution (**H1**) and will be delivered to party i in round r'. An honest chain tip in a FairPoS execution can remain unextendable for up to Δ slots (Theorem 1), which is exactly the maximum message delay permitted for messages in $\mathbf{m}^{(j)\prime}$ in the PoS Δ-synchronous setting.

  b. If $\mathcal{C}$ and equivalent $\mathcal{C}'$ feature an adversarial chain tip, $\mathcal{C}'$ cannot be in an honest tree view nor an honest message queue in the PoS execution prior to slot $t+1$: **A2** requires an adversarial chain tip in FairPoS to be extendable before it is introduced to an honest tree in PoS. Instead, any $\mathcal{C}'$ with adversarial tip must be in $\mathcal{T}^{\mathcal{A}}$, $\mathcal{T}^{\mathcal{A}\prime}$ of both executions (**A1**) at the beginning of the round, and $\mathcal{C}'$ can therefore still be delivered to $\mathcal{T}^{(i)\prime}$ from $\mathcal{T}^{\mathcal{A}\prime}$ by the adversary in round r'.

The extendable honest tree in Γ_{t+1} and honest tree in Γ'_{t+1} must be equivalent since for an addition of a *newly extendable chain* (**A2**(a), **A3**(a), **A4**) to the honest tree in FairPoS round r, the equivalent chain in PoS is added to the honest tree in round r'. □

Theorem 4. (k-Common prefix in FairPoS) *Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1-f)\Delta = (1+\epsilon)/2$ for active slot coefficient $f \in (0,1]$ and some $\epsilon > 0$. The probability that $\mathcal{A}$ makes the protocol violate the k-common-prefix property in a δ-synchronous environment throughout a period of R slots is no more than* $\exp(\ln R + \Delta - \Omega(k))$.

Proof. (Theorem 4) Let w be drawn from dominant distribution $\mathcal{D}_{\alpha}^{f}$ (Equation 1), with honest stake α and parameter f satisfying $\alpha(1-f)\Delta = (1+\epsilon)/2$ for some $\epsilon > 0$. From Corollary 1 and Theorem 2 we infer the following:

$$\Pr_{w \leftarrow \mathcal{D}_{\alpha}^{f}}[\mathsf{div}_{\delta}^{\mathsf{FairPoS}}(w) \geq k] \leq \Pr_{w \leftarrow \mathcal{D}_{\alpha}^{f}}[\mathsf{div}_{\Delta}^{\mathsf{PoS}}(w) \geq k] \leq \exp(\ln R + \Delta - \Omega(k)) \quad (18)$$

From Corollary 1, $\mathsf{div}_{\delta}^{\mathsf{FairPoS}}(w) \geq k \implies \mathsf{div}_{\Delta}^{\mathsf{PoS}}(w) \geq k$, implying the left equality in Equation 18. The right inequality is inferred from Theorem 2.

Theorem 5. ((τ, s)-Chain growth in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1-f)\Delta = (1+\epsilon)/2$ for active slot coefficient $f \in (0,1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes the protocol violate the chain growth property with parameters $s \geq 4\Delta$ and $\tau = c\alpha/4$ throughout a period of R slots, is no more than $\exp(-c\alpha s/(20\Delta) + lnR\Delta + O(1))$, where c denotes the constant $c := c(f, \Delta) = f(1-f)^{\Delta}$.

Proof (Theorem 5). The proof of chain-growth in FairPoS closely follows that of Ouroborous Praos, as both analyses assume the dominant distribution (Definition 1) to model leader elections during protocol executions. Thus, we reproduce the main proof argument of Theorem 6 in [13] for the convenience of the reader, and refer to [13] for the derivation of the exact probability bounds, which are directly inferred from the dominant distribution.

Recall that the definition of chain growth requires that if the longest chain possessed by an honest party at the onset of some slot sl_1 is $\mathcal{C}_1$, and the longest chain possessed by a (potentially different) honest party at the onset of slot $\mathsf{sl}_2 \geq \mathsf{sl}_1 + s$ is $\mathcal{C}_2$, then $\mathsf{len}(\mathcal{C}_2) - \mathsf{len}(\mathcal{C}_1) \geq \tau s$.

Let a uniquely honest slot (0) be Δ-right-isolated if it is followed with Δ consequent slots which are empty ($\perp$). Let $\mathcal{C}$ be a chain with a tip that is generated in a Δ-right-isolated uniquely honest slot. Then the next slot leader will necessarily consider $\mathcal{C}$ candidate for extension since

- Chain $\mathcal{C}$ must be have arrived in the view of all honest parties after δ slots in a δ-synchronous setting, where $\delta < \Delta$ (Equation 5).
- Chain $\mathcal{C}$ must be extractable after Δ slots (Theorem 1).

Thus, in the view of all honest parties, chain $\mathcal{C}$ must be a candidate for extension according to the longest-extractable-chain rule (Equation 8) in FairPoS.

Now, let $\hat{\mathsf{sl}}_1, ..., \hat{\mathsf{sl}}_h$ be the increasing sequence of all Δ-right-isolated uniquely honest slots among the slots in $T := \{\mathsf{sl}_1 + \Delta, \mathsf{sl}_1 + \Delta + 1, ..., \mathsf{sl}_2 - \Delta\}$. Observe that since $\hat{\mathsf{sl}}_1 \geq \mathsf{sl}_1 + \Delta$, the leader of $\hat{\mathsf{sl}}_1$ will append a block to a chain that is at least as long as $\mathcal{C}_1$, since $\mathcal{C}_1$ will be known to him and will be considered in the longest-extractable-chain selection rule. Therefore, the chain that the leader of $\hat{\mathsf{sl}}_1$ diffuses will be at least 1 block longer than $\mathcal{C}_1$. Analogously, the leader of every $\hat{\mathsf{sl}}_i$ will diffuse a chain that is at least 1 block longer than the chain diffused by the leader of $\hat{\mathsf{sl}}_{i-1}$ since $\hat{\mathsf{sl}}_{i-1}$ is Δ-right-isolated. Finally, the chain diffused by the leader of $\hat{\mathsf{sl}}_h$ will be known to all parties at slot sl_2 and hence $\mathsf{len}(C2)$ will be at least as long as this chain. It follows that $\mathsf{len}(\mathcal{C}_2) - \mathsf{len}(\mathcal{C}_1) \geq h$.

It remains to bound the number h of Δ-right-isolated uniquely honest slots among the slots with indices in T . To make our notation more flexible, let $H_T(x)$ denote the number of Δ-right-isolated uniquely honest slots among the slots from T in $x \in \{0, 1, \perp\}^R$. From Theorem 6 in [13] we have for $c = f(1 - \Delta)^{\Delta}$:

$$\Pr_{x \leftarrow \mathcal{D}_\alpha^f}[H_T(x) < c\alpha s/4] = \Delta \cdot e^{-\frac{c\alpha(s-3\Delta)}{20\Delta}}$$

Applying the union bound over R slots, we conclude that the probability that there is a chain growth violation with parameters s and $\tau = c\alpha/4$ is no more than

$$R\Delta \exp(-c\alpha(s - 3\Delta)/(20\Delta)) = \exp(-c\alpha(s - 3\Delta)/(20\Delta) + \ln R\Delta)$$

□

Theorem 6. ((μ, k)-Chain quality in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1-f)\Delta = (1+\epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then the probability that $\mathcal{A}$ makes FairPoS violate the chain quality property with parameters k and $\mu = 1/k$ throughout a period of R slots, is no more than $\exp(\ln R - \Omega(k))$.

Proof (Theorem 6). The proof of chain-growth in FairPoS closely follows that of Ouroborous Praos, as both analyses assume the dominant distribution (Definition 1) to model leader elections during protocol executions. Thus, for the convenience of the reader, we restate and adapt Lemma 4 in [13] and its main proof argument, from which Theorem 6 follows.

Lemma 2. *(Adapted from Lemma 4 in [13]) Let $k, \Delta \in \mathbb{N}$ and $\epsilon \in (0, 1)$. Let A be an adaptive adversary against the protocol (d, δ, Δ)-FairPoS corrupting up to $(1-\alpha)$ stake for some $\alpha > 0$ satisfying $\alpha(1-f)^{\Delta} = (1+\epsilon)/2$. Let $B_1, ..., B_k$ be a sequence of consecutive blocks in a chain $\mathcal{C}$ possessed by an honest party. Then at least one block B_i was created in a Δ-right-isolated uniquely honest slot, except with probability* $\exp(-\Omega(k))$.

For convenience, let us call a slot good if it is Δ-right-isolated uniquely honest, and bad if it is neither empty nor good. Moreover, we call a block good (resp. bad) if it comes from a good (resp. bad) slot.

Towards contradiction, assume that all blocks $B_1, ..., B_k$ are bad. Let G_1 denote the latest good block preceding B_1 in $\mathcal{C}$, and G_2 the earliest good block appearing after B_k in $\mathcal{C}$ (or the last block of $\mathcal{C}$, if there is no good one). Note that all blocks between G_1 and G_2 are bad.

Let $\hat{\mathsf{sl}}_1$ (resp. $\hat{\mathsf{sl}}_2$) denote the good slot in which G_1 (resp. G_2) was created (if G_2 is not good, let $\hat{\mathsf{sl}}_2$ be the current slot). Denote by T the continuous sequence of slots between $\hat{\mathsf{sl}}_1$ and $\hat{\mathsf{sl}}_2$, excluding $\hat{\mathsf{sl}}_1$ and including $\hat{\mathsf{sl}}_2$. As we argued in the proof of Theorem 5, in each good slot in T the (unique) leader creates a block that has depth increased by at least 1 compared to the block from the previous good slot. Therefore, we have $\mathsf{depth}(G_2) \geq \mathsf{depth}(G_1) + g$, where g is the number of good slots in T . However, in chain $\mathcal{C}$ we have $\mathsf{depth}(G_2) \leq \mathsf{depth}(G_1) + b$, where b is the number of bad slots in the same sequence T . These two conditions can only be satisfied at the same time if $g \leq b$, we will now show that this is very unlikely.

We can bound $\Pr_{x \leftarrow_\$ \mathcal{D}^f_\alpha}[g(x) \leq b(x)]$ as follows: we know that $\alpha(1-f)^{\Delta} = (1+\epsilon)/2$ and this implies that good slots are sampled with higher probability than bad slots. Therefore, $\Pr_{x \leftarrow_\$ \mathcal{D}^f_\alpha}[g(x) \leq b(x)]$ falls exponentially with k. Lemma 2 directly implies Theorem 6. □

Lemma 1. (Input fairness from CG, CP and CQ in (d, δ, Δ)-FairPoS) If for an execution of (d, δ, Δ)-FairPoS, (τ, d)-chain growth, $(d\tau(\tau - \delta/(\Delta - \delta)) - 1)$-common prefix, and $(1/(D+1), D+1)$-chain quality hold, where $D = d\Delta/(\Delta - \delta)$, then input fairness is implied.

Proof (Lemma 1). The proof requires us to infer input fairness from chain growth and common-prefix parameters as stated in the Lemma 1.

More informally, we frame the proof obligation as follows. At the onset of a given slot t, we are given a time budget of $\sim d$ slots, and must show that chain-growth will result in the sufficient growth in the length of the chain possessed by any honest party, such that for any encrypted input inserted at a block in slot t, it will reach the k-common-prefix within the given time budget, unless it becomes part of an abandoned branch.

Let $\mathcal{C}$ be the chain extended by an honest party at honest slot t. For simplicity, let us first assume that all slots are uniquely honest. An adversary begins the extraction of $\mathsf{id} = \mathcal{C}.\mathsf{tip}$ at the onset of its generation. Then, there remains $d-1$ slots between the encryption of the input at slot t and its decryption, since the input at t is encrypted with the parent block as session id. A chain growth rate of τ implies that the longest chain possessed by any honest party after $d-1$ time must increase by $k = \tau(d-1)$. Thus, in this naive scenario, input fairness would be implied by (τ, d)-CG and $(\tau(d-1))$-CP.

In the case that slots are *not all uniquely honest*, the adversary must be permitted a head-start in extracting the session key idk from any block: let us denote this the *time advantage*, which comes from two properties of the chain possessed by the honest user at the onset of slot t:

1. *Leading empty slots* between $\mathcal{C}.\mathsf{tip.sl}$ and current slot t. These empty slots represent a head-start the adversary has in decrypting inputs inserted in a child block of $h(\mathcal{C}.\mathsf{tip})$ generated at slot t.
2. *Leading adversarial block span* in $\mathcal{C}$ including $\mathcal{C}.\mathsf{tip}$, allowing it to generate blocks immediately after the extraction period d instead of waiting the full extraction schedule $D = d + n\delta$, as an honest party would. In the worst case, such a adversarial block span always leads up to $\mathcal{C}.\mathsf{tip}$, which is also adversarially generated, permitting the adversary an additional head-start in decrypting inputs.

For the (1) *leading empty slots*, the (τ, d)-CG property gives us the maximum number of slots in d, in which *no blocks* were generated for $\mathcal{C}$, namely $d(1-\tau)$. For the (2) *leading adversarial block span*, we quantify the *time advantage* gained from the leading adversarial block span as follows: for every D consecutive adversarial blocks, the adversary gains an additional $n\delta$ *time advantage*, since it does not have to wait the entire extraction schedule $D = d + n\delta$, where $n = D/\Delta = d/(\Delta - \delta)$. Note that we are granted $(1/(D+1), D+1)$-chain quality in Lemma 1, where $D = n\Delta = d\Delta/(n-\delta)$ as in Equation 5. We assume the worst case, namely, that all D slots leading up to t are indeed adversarial, and thus permit the adversary the maximum possible extraction time-advantage of $n\delta$ slots.

Thus the total time advantage in producing the adversarial block $\mathcal{C}.\mathsf{tip}$ obtained from (1) and (2) is given by $t_{\mathsf{adv}} = d(1-\tau)+n\delta = d(1-\tau+\delta/(\Delta-\delta))$. Thus we require a *contracted* common prefix parameter $k = \tau(d-t_{\mathsf{adv}})-1$, in order for the honest input at slot t to either reach the common prefix before the adversary (with *time advantage*) can complete the key extraction of $\mathsf{id} = h(\mathcal{C}.\mathsf{tip})$, or not join the common prefix at all. Rewriting gives us $k = d\tau(\tau - \delta/(\Delta - \delta)) - 1$ □

Theorem 7. (Input fairness in (d, δ, Δ)-FairPoS) Let $\mathcal{A}$ be an an adaptive adversary against the protocol (d, δ, Δ)-FairPoS that corrupts up to $(1-\alpha)$ stake, where α be such that $\alpha(1 - f)\Delta = (1 + \epsilon)/2$ for active slot coefficient $f \in (0, 1]$ and some $\epsilon > 0$. Then, the probability that $\mathcal{A}$ makes the FairPoS violate the input fairness property falls exponentially with d.

Proof (Theorem 7). With Lemma 1 we obtain input fairness from (τ, d)-CG and $(d\tau(\tau - \delta/(\Delta - \delta)) - 1)$-CP. Further, for an execution of (d, δ, Δ)-FairPoS for R slots,

- (τ, d)-CG with parameters $d \geq 4\Delta$ and $\tau = c\alpha/4$ is violated with probability no more than $\exp(-d\alpha c/(20\Delta) + lnR\Delta + O(1))$, where c denotes the constant $c := c(f, \Delta) = f(1 - f)^{\Delta}$ (Theorem 5).
- k-CP with parameter $k = d\tau(\tau - \delta/(\Delta - \delta)) - 1$ is violated with probability no more than $\exp(\ln R + \Delta - \Omega(k))$ (Theorem 4).
- $(1/(D + 1), D + 1)$-CQ with parameter $D = d\Delta/(\Delta - \delta)$ is violated with probability no more than $\exp(\ln R + \Omega(D))$ (Theorem 6).

Probabilities above decline exponentially with increasing d. □

Eagle: Efficient Privacy Preserving Smart Contracts

Publication Information

Carsten Baum, James Hsin-yu Chiang, Bernardo David and Tore Kasper Frederiksen. “Eagle: Efficient Privacy Preserving Smart Contracts.” To appear in *Financial Cryptography and Data Security: 27th International Conference, FC 2023, Bol, May 1–5, 2023.*

Contribution

- Co-author.

Remarks

Accepted and presented at conference. Proceedings by publisher are work-in-progress.

Eagle: Efficient Privacy Preserving Smart Contracts

Carsten Baum[1], James Hsin-yu Chiang[1], Bernardo David[2], Tore Kasper Frederiksen[3],

[1] Technical University of Denmark, Denmark
`cabau@dtu.dk` [⋆] , `jchi@dtu.dk`
[2] IT University of Copenhagen, Denmark
`bernardo@bmdavid.com` [⋆⋆]
[3] `jot2re@gmail.com` [⋆ ⋆ ⋆]

Abstract. The proliferation of Decentralised Finance (DeFi) and Decentralised Autonomous Organisations (DAO), which in current form are exposed to front-running of token transactions and proposal voting, demonstrate the need to shield user inputs and internal state from the parties executing smart contracts. In this work we present "Eagle", an efficient UC-secure protocol which efficiently realises a notion of privacy preserving smart contracts where both the amounts of tokens and the auxiliary data given as input to a contract are kept private from all parties but the one providing the input. Prior proposals realizing privacy preserving smart contracts on public, permissionless blockchains generally offer a limited contract functionality or require a trusted third party to manage private inputs and state. We achieve our results through a combination of secure multi-party computation (MPC) and zero-knowledge proofs on Pedersen commitments. Although other approaches leverage MPC in this setting, these incur impractical computational overheads by requiring the computation of cryptographic primitives within MPC. Our solution achieves security without the need of any cryptographic primitives to be computed inside the MPC instance and only require a constant amount of exponentiations per client input.

1 Introduction

Ethereum introduced the first implementation of Turing-complete smart contracts for blockchains, widely adopted for financial and contracting applications

[⋆] Part of the work was carried out while the author was visiting Copenhagen University and supported by Partisia. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of Partisia.

[⋆⋆] The project was supported by the Concordium Foundation, by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B, and by Copenhagen Fintech.

[⋆ ⋆ ⋆] The work was carried out while at the Alexandra Institute, supported by Copenhagen Fintech as part of as part of the "National Position of Strength programme for Finans & Fintech" funded by the Danish Ministry of Higher Education and Science.

since its introduction in 2015. Smart contracts offer auditability and correctness guarantees, and as a consequence expose both their state and any submitted inputs to all participants of the blockchain network. This lack of privacy not only leaks user data but also gives rise to concrete attacks. For example, current Decentralised Finance (DeFi) and Decentralised Autonomous Organisations (DAO) are vulnerable to front-running [27] in token transactions and proposal voting. This motivates the need to shield user inputs and internal contract state from the very parties who execute smart contracts in a decentralized environment.

Challenges. Hawk [44] introduced the first notion of general-purpose privacy preserving smart contracts, which required users to privately submit both input strings and confidential balances to a trusted contract manager. Upon evaluation of the contract over private inputs, the contract manager settles the confidential outputs to a confidential ledger, proving in zero knowledge that these outputs have been obtained according to the contract's instructions. Importantly, in order to accommodate real-world applications such as DeFi or DAO's, we must extend the Hawk notion of confidential contracts as follows:

1. Distribute the role of the trusted third party in an efficient manner, avoiding a single point of failure without significantly sacrificing performance.
2. Only require clients to be online during a short input phase; as in the standard client-blockchain interaction model, clients only broadcast signed inputs.
3. Allow privacy preserving smart contracts to be long-running applications over indefinite rounds, as is the case in standard, public smart contracts.

Our Contributions. In this work we present "Eagle", a Universally Composable [20] protocol for achieving efficient privacy preserving smart contracts, which handles all the three challenges explained above: (1) is achieved by evaluating the contract's instructions via an *outsourced* secure multi-party computation (MPC) protocol [37], where clients provide private inputs and servers execute the bulk of the protocol to compute a function on these inputs without learning them. We use a MPC protocol, known as *insured MPC*, which allows a public verifier to identify servers aborting at the output phase, so that cheating servers can be identified and financially punished, incentivizing fairness (*i.e.* if a server gets the output, all servers/clients also get it) [7]. That is, by combining outsourced and insured MPC we get a protocol where client computation and interaction is independent of the circuit computed in MPC and where *reliability* is incentivized and *security* is obtained as long as only a single MPC is honest. (2) is accomplished with a novel input protocol which pre-processes data necessary for the servers to generate private outputs (*e.g.* token amounts) that are posted directly to the public ledger but can only be read by specific clients. (2) facilitates (3), realized by a *reactive* version of our MPC protocol, which maintains a secret off-chain state over multiple rounds. Here, we contribute a model of long-running, privacy preserving contracts, which at the onset of each round accepts new inputs from any subset of clients. At the end of each round, clients get public outputs and servers keep a secret internal state, allowing evaluation to take place in a continuous, *multi-round* fashion, even if clients are offline (2).

Applications. Several general applications for privacy preserving smart contracts have already been proposed. **Auctions:** can be realized securely on-chain with privacy preserving smart contracts, as auctions implemented without privacy are vulnerable to front-running (miners can trivially observe individual bids posted to the ledger). **Identity management:** Decentralized Identity (DID) management considers the setting where user-attributes are posted to a ledger, in a certified, yet hidden manner. DID implemented with privacy preserving smart contracts enables proofs and computations on private identity attributes, facilitating their integration with blockchain applications. **KYC Mixing:** We can construct a privacy preserving smart contract to realize a mixer that enforces Anti Money Laundering (AML) policies. For example, such a mixer could use DID to integrate Know Your Customer (KYC) information to either limit user permissions or the quantity of mixed tokens allowed per month. **Side-chains:** The MPC servers alone could be considered a privacy preserving side-chain. Multiple sets of MPC servers could work together with a single smart contract to realize a privacy preserving sharding scheme on any layer 1 chain with Turing complete smart contracts. **AMMs and DeFi via Cross-chain contracts:** Using ideas of P2DEX [9], we show that the MPC servers can interact with smart contracts on many different ledgers. Hence, privacy preserving smart contracts can work *across* multiple ledgers and different native tokens. This realizes cross-chain, front-running resistant automated market makers (AMMs) with strong privacy guarantees. We discuss these applications in more detail in Appendix F.

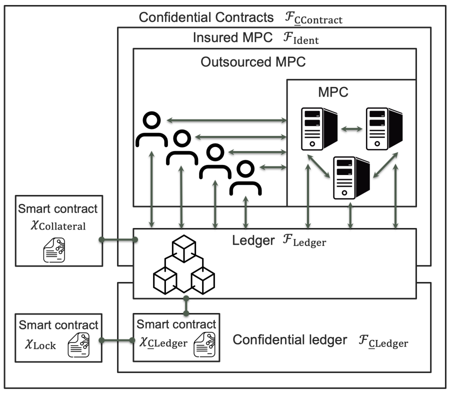

Confidential Contracts $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$
Insured MPC $\mathcal{F}_{\mathsf{Ident}}$
Outsourced MPC
MPC
Smart contract $\mathcal{X}_{\mathsf{Collateral}}$
Ledger $\mathcal{F}_{\mathsf{Ledger}}$
Smart contract $\mathcal{X}_{\mathsf{Lock}}$
Smart contract $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$
Confidential ledger $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

Fig. 1: Outline of our protocol for confidential contracts. The wrapping and interaction of functionalities are shown.

Our Techniques. We sketch our protocol in Fig. 1. This only considers execution of a *single* instance of a privacy preserving smart contract for simplicity. We discuss the multi-round setting in Sec. E where computations are executed continuesly with different sets of clients. We assume a set of clients $\mathcal{C}$ and MPC servers $\mathcal{P}$, both interacting with a ledger functionality $\mathcal{F}_{\mathsf{Ledger}}$. The ledger hosts two deployed smart contract instances: $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ maintains a confidential ledger and is extended with $\mathcal{X}_{\mathsf{Lock}}$, which locks and redistributes confidential balances, output and jointly signed, by the MPC servers. Concretely our protocol runs the following phases:

Init Before any execution, the servers setup the system by sampling a threshold signature key pair and provide sufficient collateral for the insured MPC execution, and setup smart contact $\mathcal{X}_{\mathsf{Lock}}$, administered by the distributed signature key. We note that in the multi-round setting this only needs to be executed *once* for the specific set of MPC servers, and is thus independent of the clients and the amount of computations that will get carried out later.

Enroll When a privacy preserving smart contract is to be executed, each client who wish to participate transfers confidential tokens to $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ which they wish to use as input to the confidential smart contract $\underline{\mathsf{C}}\mathsf{Contract}$. The client then gives any auxiliary input, along with the opening information to the commitment containing their confidential balance v, to the insured MPC functionality $\mathcal{F}_{\mathsf{Ident}}$ from Fig. 6, extended with a secure client input interface (See Appendix B.1). Each client constructs an appropriate amount of "mask" commitments; one for each round of confidential contract computation, for which they wish their input to be used. A masking commitment is simply a commitment to a random value.

Verify input The servers validate the input received from the clients using outsourced MPC, and ensure that $\mathcal{X}_{\mathsf{Lock}}$ has also received the appropriate confidential tokens. The servers and the clients also execute a proof to ensure that the opening information supplied by clients are indeed valid for the confidential token commitments. They do this following a standard Σ-protocol where each client commits to a random commitment a and servers select a random challenge γ and ask the client to open $\mathrm{com}(c) = \mathrm{com}(a) \oplus (\gamma \odot \mathrm{com}(v))$. Similarly the servers use MPC to securely open $[c] = [a] + \gamma \cdot [v]$ and check consistency[4].

Evaluate After the checks are completed the servers evaluate the circuit expressing the private smart contract $\underline{\mathsf{C}}\mathsf{Contract}$, using insured MPC. For the clients who are supposed to get output from this round of computation, shares of messages and randomness for a new commitment for *each* client are computed, and blinded with the "masking" values the clients provided during *Enroll*. If this goes well, the servers distributedly sign a message saying that they have reached this stage and post it to $\mathcal{X}_{\mathsf{Lock}}$.

Open For clients that receive output after this round of computation the servers open the masked output. They publish these values and sign them, as part of the transcript of the current round execution, and post this to $\mathcal{X}_{\mathsf{Lock}}$. Note that $\mathcal{X}_{\mathsf{Lock}}$ can generate the output coins in commitment form, due to the homomorphism of the commitments and since it obtained the mask commitments from the clients in *Enroll*. $\mathcal{X}_{\mathsf{Lock}}$ can then transfer the new confidential tokens back to the client's address. We show an extension to our protocol (Section E) that ensures no token minting can occur even if all servers are corrupted.

Withdraw Based on the masks they constructed, the clients who are supposed to receive outputs can compute the coin commitment openings from their masked outputs signed and posted to $\mathcal{X}_{\mathsf{Lock}}$ by servers during *Open*.

Abort In case a server stops responding or acts maliciously, an honest server can request the entering of an abort phase. Any server can do this, either by

[4] In our full protocol we optimize this by batching client input checks.

submitting a proof that the malicious server sent wrong information or by requesting missing information from the accused servers. At this point the accused malicious server has a constant amount of time left to prove to the smart contract that they did not abort, by submitting the message that the accusing server claims they didn't get. If they don't, they will have their collateral revoked and it will be shared among the honest servers and clients, and the contract state will roll back one round, i.e. to the contract state preceding *Evaluate*. Concretely $\mathcal{X}_{\mathsf{Lock}}$ will refund the clients their input funds, plus a compensation obtained from the cheating servers' collateral.

Related Works. A long line of work realizes notions of privacy preserving smart contracts that sacrifice privacy [44,61,38,48,41,62,60,24] or flexibility [15,16]. Zexe [15] extends the ZCash model of confidential transactions to enable Bitcoin Script-like stateless privacy preserving smart contracts supporting only very simple logic. Zether [16] implements confidential transactions on top of Ethereum, allowing for very simple privacy preserving applications (*e.g.* auctions). Zkay [61] allows for computing on encrypted private inputs by means of keeping data encryption on the blockchain, and using NIZKs to validate that any updates done to the encrypted is carried out correctly. Follow-up work, Zeestar [60] uses additively homomorphic encryption to allow for *limited* private computation on data from multiple owners, without them having to share their private data with each other. Secret Network [62] and Ekiden [24] implement general purpose contracts but rely on notoriously vulnerable trusted execution environments (*e.g.* Intel SGX [51]) for privacy and correctness. Arbitrum [38] relies on a full quorum of parties (the servers in our setting) being honest to achieve privacy for general purpose contracts. Finally, Kachina [41] subsumes these approaches with a framework based on state oracles [48] that yields privacy preserving smart contracts, where either flexibility is limited (*i.e.* contract state is only updated by one client's private input at a time) or privacy is compromised (*i.e.* a trusted third party must learn clients' private inputs in order to update the state). The ideal functionality of Kachina is designed to permit *input concurrency*, allowing honest inputs to be finalized on a global ledger in a different order as their generation; the Kachina protocol requires private inputs to be accompanied with NIZKs proving a valid update of the private state fragment. Here, the NIZKs are not bound to a specific, public contract state and thus remain valid even if the public contract state observed by the user was updated by another user input in the meantime.

Combining MPC with blockchain based cryptocurrencies and smart contracts has been investigated in a long line of works [1,2,12,46,45,47,43,25,13,11,31,7,9,8] aiming at achieving fairness in the dishonest majority setting via financial punishments. The core idea of these works is having all parties, who execute the MPC protocol, provide a collateral deposit, which is taken from them in case they are caught cheating. Thus incentivizing honest behavior. However, this approach publicly reveals the amount of collateral deposited by each party, which falls short of achieving our notion of privacy preserving smart contracts, where both auxiliary data *and* the amount of tokens given as input to the contract must remain private. Notice that revealing the deposit amount is an issue in applications where this amount is directly related to the client's private input, *e.g.* in sealed-bid auctions, where the collateral deposit must be equal to at least the client's private bid. An auction protocol using collateral deposits with private amounts was proposed in [32] but it cannot be generalized to other tasks.

Hawk [44, App. G] does suggest to use MPC to achieve a decentralized confidential smart contracts on both token amount and auxiliary input. However, Hawk works in the ZCash model and thus their MPC solution would require the computation of SNARKs to realize the ZCash transactions, within the MPC circuit. Although it has been shown [53,39] that integrating NIZKs with MPC can be done without degrading performance too much, there is still a performance hit. Since the construction of a single ZCash transaction SNARK still takes a non-negligible amount of time plain, this would naturally be inefficient to realize in MPC, as MPC is orders of magnitude slower than regular computation. Furthermore, they need *all* users to take part in the MPC computation. zkHawk [4] improves upon this, by forgoing the need of doing SNARKs in MPC, but still require *all* users taking part in a confidential smart contract to facilitate an MPC computation which *must* compute Schnorr style ZKPs on Pedersen commitments to the bit-decomposition of the amount of coins each of them hold. While V-zkHawk [5] forgoes the need of proofs of the bit-decomposed commitments, they replace it with the computation of commitments in a larger fields and a signature, in MPC instead. While more efficient, this approach would still require MPC over a large domain and contributes non-negligible overhead.

In Appendix A we further discuss related works.

2 Preliminaries

Let $y \leftarrow\!\!\$\, F(x)$ denote running the randomized algorithm F with input x and implicit randomness, and obtaining output y. Similarly, $y \leftarrow F(x)$ is used for a deterministic algorithm. For a set $\mathcal{X}$, let $x \leftarrow\!\!\$\, \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$. s denotes the computational and κ the statistical security parameter. Let $[\,x\,]$ denote secret x maintained in an MPC instance: we lift the $[\,\cdot\,]$ notation to any object that can be encoded over secrets securely input to an MPC scheme, e.g. $[\,g\,]$, where g is an arithmetic circuit over field $\mathbb{F}$. We use a group $\mathbb{G}$ where the discrete log problem is hard, and which is a source group of pairing scheme. For simplicity we assume $|\mathbb{G}| = |\mathbb{F}| = p$. Unless noted otherwise we use log to denote the logarithm to base 2, rounded up. We use $\bar{v}_{\mathsf{max}}$ to denote the maximum amount of tokens we want to represent and say $l = \log(\bar{v}_{\mathsf{max}})$. For simplicity, we assume $|\mathcal{C}| \cdot \bar{v}_{\mathsf{max}} < |\mathbb{G}|$, where $\mathcal{C}$ is the set of participating clients. We

Table 1: Notation.

$\mathcal{P}$	The set of servers.
$\mathcal{C}$	The set of clients.
n	Number of servers $n = \lvert\mathcal{P}\rvert$.
m	Number of clients; $m = \lvert\mathcal{C}\rvert$.
l	Number of bits representing balances.
z	Number of input/output per client.
κ	Computational security parameter.
s	Statistical security parameter.
$\mathcal{F}$	An ideal functionality.
Π	A protocol.
$\mathcal{L}$	A ledger map indexed by vk.
$\mathcal{X}$	A smart contract program.
g	A smart contract in circuit form.
vk	A public key for signature verification.
x	A client input.
y	A client output.
$\bar{\mathbf{v}}$	A token balance.
$\bar{v}_{\mathsf{max}}$	The maximum permitted balance.
$\bar{\mathbf{v}}_{\mathsf{max}}$	A vector of the maximum permitted balance.

denote set $\{1, 2, \ldots, n\}$ by $[n]$ and vectors
by bold faced Latin letters, e.g. $\mathbf{v}, \mathbf{w}$.

2.1 Security Model and Building Blocks

We analyse our results in the the (Global) Universal Composability or (G)UC framework [21,23]. We consider static malicious adversaries. Our protocols work in a synchronous communication setting, which is modelled by assuming parties have access to a global clock ideal functionality $\mathcal{F}_{\mathsf{Clock}}$ as seen in multiple works [3,40,43]. The core component of our protocols is publicly verifiable MPC with cheater identification in the output phase, which is modelled as an ideal functionality $\mathcal{F}_{\mathsf{Ident}}$, which can be realized as described by Baum *et al.* [7,9]. This functionality produces a proof that either a certain output was obtained after the MPC or that a certain party has misbehaved in the output phase, while cheating before the output phase causes an abort without cheater identification. We further extend this functionality to handle reactive computation [30,29] and an *outsourced* computation with inputs provided by clients and computation done by servers [37,28]. Moreover, we use Pedersen Commitments [54], digital signatures represented by an ideal functionality $\mathcal{F}_{\mathsf{Sig}}$ as in [22], threshold signatures represented by an ideal functionality $\mathcal{F}_{\mathsf{TSig}}$ as defined by Baum *et al.* [9] and non-interactive zero knowledge proofs represented by $\mathcal{F}_{\mathsf{NIZK}}$ as defined by Groth [36]. Further discussion on our security model and building blocks is presented in Appendix B.

2.2 Ledgers & Smart Contracts

We model a ledger functionality $\mathcal{F}_{\mathsf{Ledger}}$ in Appendix C.1 featuring a smart contract virtual machine which is adapted from an authenticated, public bulletin board functionality, an approach adopted from the work of Baum *et al.* [7,9]. For this work, we emphasize accurate modelling of confidential balances, which are implemented on a public ledger, and omit the full consensus details in our UC model, similar to previous works [43,3].

Token universe. $\mathcal{F}_{\mathsf{Ledger}}$ supports a token universe consisting of t token types: $\mathbb{T} = (\tau_1, ..., \tau_t)$. A ledger in $\mathcal{F}_{\mathsf{Ledger}}$ maintains a map from signature verification key to balances of each token type: $\mathcal{L} : \{0,1\}^* \rightarrow \mathbb{Z}^t$. We write $\bar{\mathbf{v}} = (v_1, ..., v_t)$ for a balance over all supported token types. In addition to balances associated to signature verification keys, $\mathcal{F}_{\mathsf{Ledger}}$ also maintains token balances for each deployed smart contract instance. The ledger functionality enforces the preservation of token supplies over $\mathbb{T}$.

Overview of smart contracts. In this work, we present smart contracts as human-readable programs and assume the presence of a compiler which translates program $\mathcal{X}$ to a valid circuit T and initial state γ_{init}. The following smart contract programs are deployed in the protocol which realizes the proposed confidential contract functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$.

- $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ (Figure 11) describes a smart contract which implements a confidential token wrapper for each token in $\mathbb{T}$ supported on the base ledger $\mathcal{F}_{\mathsf{Ledger}}$.
- $\mathcal{X}_{\mathsf{Lock}}$ (Figure 14) is an extension to $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. It permits the locking and redistribution of confidential balances authorized by verifying threshold signatures generated by the servers (via global functionality $\mathcal{F}_{\mathsf{TSig}}$).
- $\mathcal{X}_{\mathsf{Collateral}}$ (Figure 15) accepts collateral deposits from servers, which upon being identified as cheating parties lose their collateral to clients.

2.3 Confidential ledgers from $\mathcal{F}_{\mathsf{Ledger}}$

We briefly describe a confidential ledger functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, presented in full detail in Appendix C.2, that can be implemented from a hybrid $\mathcal{F}_{\mathsf{Ledger}}$ functionality, enabling both confidential balances and the confidential transfer of default tokens types $\mathbb{T}$ exposed by the underlying public ledger $\mathcal{F}_{\mathsf{Ledger}}$. This modeling choice maximizes the generality of our construction, as it can be implemented on any standard ledger and a basic smart contract machine.

Confidential ledger. Confidential coins in $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ are identifiable by a unique public id, and a confidential balance $\bar{\mathbf{v}}$ over $\mathbb{T}$, as in [55]. Each confidential token is publicly associated with an account verification key vk, owned by a party that generated it with GenAcct. A confidential transfer consumes two input coins $(\mathsf{id}_1, \mathsf{id}_2)$ with confidential balances $(\bar{\mathbf{v}}_1, \bar{\mathbf{v}}_2)$ and mints fresh output coins $(\mathsf{id}'_1, \mathsf{id}'_2)$ with confidential balances $(\bar{\mathbf{v}}'_1, \bar{\mathbf{v}}'_2)$, such that $(\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 = \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2)$. Here, coin id'_1 is now held by the owner of the receiving account, who also learns the confidential amount $\bar{\mathbf{v}}'_1$.

Functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ exposes Mint and Redeem interfaces: a mint activation *locks* a public amount of tokens $\mathbb{T}$ and generates a fresh confidential token of the same balance. Conversely, a redeem activation will *release* the balance of a confidential coin back to the public ledger.

Realizing a confidential ledger. A confidential token is realized in protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ described in full detail in Appendix D.1 with Pedersen Commitments [54]. Let $g, g_1, ..., g_t, h$ denote generators of group $\mathbb{G}$ of safe prime order p, such that s_i in $g_i = g^{s_i}$ and w in $h = g^w$ are given by $\mathcal{F}_{\mathsf{Setup}}$ (parameterized with $g \in \mathbb{G}$) that publicly outputs $g_1, ..., g_t, h$. The commitment to a balance $\bar{\mathbf{v}} = (v_1, ..., v_t)$ over tokens $\mathbb{T}$ with blinding r is $\mathsf{com}(\bar{\mathbf{v}}, r) = \mathbf{g}^{\bar{\mathbf{v}}} h^r = g_1^{v_1} ... g_t^{v_t} h^r$. Pedersen commitments are additively homomorphic: $\mathsf{com}(\bar{\mathbf{v}}_1, r_1) \circ \mathsf{com}(\bar{\mathbf{v}}_2, r_2) = \mathsf{com}(\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2, r_1 + r_2)$. Thus, during a confidential transfer, the sum equality between consumed input and freshly constructed output coin commitments holds if total token balances are preserved and r'_1 and r'_2 are correlated such that $r_1 + r_2 = r'_1 + r'_2$.

$$\mathsf{com}(\bar{\mathbf{v}}_1, r_1) \circ \mathsf{com}(\bar{\mathbf{v}}_1, r_1) = \mathsf{com}(\bar{\mathbf{v}}'_1, r'_1) \circ \mathsf{com}(\bar{\mathbf{v}}'_2, r'_2) \tag{1}$$

However, since the equality above holds for any $\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 \equiv \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2 \mod p$ and correlated r'_1, r'_2, an additional p units of each token in $\mathbb{T}$ can be minted:

$\bar{v}_1 + \bar{v}_2 + p \equiv \bar{v}'_1 + \bar{v}'_2 \mod p$. Thus, each confidential token is associated with NIZK π which proves $\mathcal{R}(c; \bar{\mathbf{v}}, r) = \{c = \mathsf{com}(\bar{\mathbf{v}}, r) \wedge \bar{\mathbf{v}} \leq \bar{\mathbf{v}}_{\mathsf{max}} = 2^l - 1\}$, such that such wrap-around never occurs undetected.

We note that $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in itself affords a fully decentralized layer 2 confidential token transfer solution, since it is independent of the MPC servers. Thus allowing client's to send a receive confidential tokens in a peer-to-peer manner. This is needed to prevent leakage of exchange orders after-the-fact by analysing client's non-confidential tokens given as input and withdrawn as output from a privacy preserving smart contract execution. By allowing the privacy preserving smart contract executions to integrate in a greater payment ecosystem reasonably ensures that it is possible to hide token inputs and outputs from a privacy preserving smart contract execution by using them for confidential payment, similar to other confidential token systems.

We present a protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ which GUC-realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in Appendix D.1, where we also prove the following statement:

Theorem 1. *Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ GUC-realizes functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$-hybrid model against any PPT-adversary corrupting any minority of committee $\mathcal{Q}$.*

3 Confidential contracts

We present our formal model of confidential contracts. We assume m clients $\{C_1, \ldots, C_m\}$ and servers $\{P_1, \ldots, P_n\}$ that interact with $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$, which extends $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. For simplicity of presentation, we first present a single-round confidential contract functionality in Figure 2, and subsequently illustrate how it is easily extended to a multi-round contract functionality where clients can selectively choose to participate in specific rounds.

The choice of modelling $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ as an extension of $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ arises from the relation between underlying protocols: confidential coins in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ that are committed to a confidential contract evaluation must be *locked* and subsequently *replaced* by a new set of output coins reflecting a new distribution of balances, determined by $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$. However, this requires verification operations over the homomorphic *commitment* representation of coins in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$, which are not exposed by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

We provide a brief sketch of the interface exposed by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. Upon initialization with an arithmetic circuit g encoding only the contract logic, users can enroll, specifying input string x and a confidential coin to input, identified by its id. Upon a completed *Enroll*, the functionality is prompted by servers to evaluate circuit g on both client input strings, interpreted as numerical values, and input balances, with checks to ensure g does not mint tokens. $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ permits clients to withdraw anytime to retrieve the private output string and output balance. $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ permits the simulator to abort and indicate cheating servers, which are then penalized by the functionality.

Model of confidential contracts. Unlike public smart contracts deployed to $\mathcal{F}_{\mathsf{Ledger}}$, an instance of $\mathcal{F}_{\mathsf{Ident}}$ permits the computation of any arithmetic circuit

Functionality $\mathcal{F}_{\underline{\text{C}}\text{Contract}}$, extends $\mathcal{F}_{\underline{\text{C}}\text{Ledger}}$

$\mathcal{F}_{\underline{\text{C}}\text{Contract}}$ interacts with clients $\mathcal{C} = \{C_1, ..., C_m\}$ and servers $\mathcal{P} = \{P_1, ..., P_n\}$. The functionality exposes interfaces and and accesses the state of $\mathcal{F}_{\underline{\text{C}}\text{Ledger}}$. It is parameterized with max. circuit depth d_T, and collateral balance $\bar{\mathbf{v}}_{\mathsf{coll}}$.

```
Init: On (INIT, $sid, g$) from $P_i \in \mathcal{P}$ forward messages to $\mathcal{S}$. If $\mathcal{S}$ continues.
1. Run GenAcct and Init procedures on $\mathcal{F}_{\underline{\text{C}}\text{Ledger}}$.
2. Assert that $g$ is a circuit and that $\mathsf{depth}(g) \leq d_T$, store $g$.
3. Assert $\mathsf{vk} \in \mathcal{K}[P_i]$ and $\mathcal{L}[\mathsf{vk}] \geq \bar{\mathbf{v}}_{\mathsf{coll}}$.
4. Set $\mathcal{L}[\mathsf{vk}] \leftarrow \mathcal{L}[\mathsf{vk}] - \bar{\mathbf{v}}_{\mathsf{coll}}$.
   - If all servers have successfully called Init, set state to enroll, tick $\mathcal{F}_{\mathsf{Clock}}$.

Enroll: Upon input (ENROLL, $sid, x, \mathsf{id}, \mathsf{vk}$) from client $C_j \in \mathcal{C}$,
1. Assert $\mathsf{vk} \in \mathcal{K}[C_j]$ and $\langle \mathsf{id}, \bar{\mathbf{v}} \rangle \in \mathcal{L}[\mathsf{vk}]$.
2. Forward (ENROLL, $sid, \mathsf{id}, \mathsf{vk}$) to $\mathcal{S}$, if $\mathcal{S}$ aborts, run Abort. Otherwise, continue.
3. Assert state is in enroll and $\exists \langle \mathsf{id}, \bar{\mathbf{v}} \rangle \in \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}]$: then remove $\langle \mathsf{id}, \bar{\mathbf{v}} \rangle$.
4. Store input $(x_j, \bar{\mathbf{v}}_j)$.
   - If all clients have successfully called Enroll, tick $\mathcal{F}_{\mathsf{Clock}}$.

Execute: Upon input (EXECUTE, $sid$) from $P_i \in \mathcal{P}$,
1. If EXECUTE received from all $\mathcal{P}$ and $\mathcal{F}_{\mathsf{Clock}}$ ticked since state update to enroll,
   forward (EXECUTE, $sid$) to $\mathcal{S}$ and wait for Ok or Abort. If Ok, continue.
   a. Evaluate circuit $g$ over current user inputs $\{(x_j, \bar{\mathbf{v}}_j)\}_{j \in [m]}$ and client state.
   c. Store client states $\{(y_j, \bar{\mathbf{w}}_j)\}_{j \in [m]}$ read from output gates of $g$.
       - Assert $\sum_{j \in [m]} \bar{\mathbf{v}}_j = \sum_{j \in [m]} \bar{\mathbf{w}}_j$. Tick $\mathcal{F}_{\mathsf{Clock}}$.
2. Forward (EVALUATE, $sid$) to $\mathcal{S}$ and wait for Ok or Abort.
   - If Ok returned, set state to evaluated and tick $\mathcal{F}_{\mathsf{Clock}}$.
3. Send (OUTPUT, $sid, \{y_j, \bar{\mathbf{w}}_j\}_{j \in [m]}$) to $\mathcal{S}$ and wait for Ok or Abort.
   - If $\mathcal{S}$ aborts, it provides cheating server set $\mathcal{J}$, run Abort with $\mathcal{J}$.
4. For $j \in [m]$, get a unique $\mathsf{id}'_j$ from $\mathcal{S}$, and set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \cup \{\langle \mathsf{id}'_j, \bar{\mathbf{w}}_j \rangle\}$.
5. Set state to enroll and tick $\mathcal{F}_{\mathsf{Clock}}$.

Withdraw: Upon (WITHDRAW, $sid$) from $C_j \in \mathcal{C}$, obtain newly stored outputs since
last Withdraw by $C_j \in \mathcal{C}$ . Return $((y_{j,1}, \langle \mathsf{id}'_{j,1}, \bar{\mathbf{w}}_{j,1} \rangle), ..., (y_{j,l}, \langle \mathsf{id}'_{j,l}, \bar{\mathbf{w}}_{j,l} \rangle))$.
```

```
Abort: Tick $\mathcal{F}_{\mathsf{Clock}}$,
a. If state is enroll, return server and client funds: update $\mathcal{L}$, $\mathcal{L}_{\mathsf{Conf}}$.
b. Else if state in evaluated, obtain cheating servers $\mathcal{J}$ from $\mathcal{S}$:
   - If $\mathcal{J} \neq \emptyset$, reimburse clients $\mathcal{C}$ and honest servers $\mathcal{P} \backslash \mathcal{J}$, then
     distribute $\mathcal{J}$'s collateral amongst $\mathcal{C}$: update $\mathcal{L}$, $\mathcal{L}_{\mathsf{Conf}}$ accordingly.
   - Else if $\mathcal{J} = \emptyset$, obtain $\{(y_j, \bar{\mathbf{v}}_j)\}_{j \in [m]}$ from last evaluation of circuit $g$.
     - For $C_j \in \mathcal{C}'$, sample $\mathsf{id}_j \leftarrow\$ \mathbb{F}$ and set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \cup \{\langle \mathsf{id}_j, \bar{\mathbf{v}}_j \rangle\}$.
     - Return collateral for all $\mathcal{P}$: update $\mathcal{L}$ accordingly.
c. Terminate.
```

Fig. 2: Functionality for Confidential Contracts

on both private and public inputs. We model a confidential contract as an arithmetic circuit over a field $\mathbb{F}_p$ consistent with the domain that $\mathcal{F}_{\mathsf{Ident}}$ is realized with. A well-formed confidential contract permits the writing of both *numerical* and *financial* inputs from each client to its input gates. Further, we enforce a maximum circuit depth d_T prior to the circuit evaluation to bound the rounds of interaction in the MPC instance.

$$\big(([\,y_1\,],[\,\bar{\mathbf{w}}_1\,]),...,([\,y_m\,],[\,\bar{\mathbf{w}}_m\,])\big) \leftarrow \mathsf{eval}_g\big(([\,x_1\,],[\,\bar{\mathbf{v}}_1\,])....,([\,x_m\,],[\,\bar{\mathbf{v}}_m\,])\big)$$

Upon confidential evaluation of a contract circuit g with well-formed depth and gates, the following assertion must be performed at each run-time over confidential inputs and outputs of evaluated g: namely, that token supplies have been preserved.

$$\sum_{i\in[m]}[\,\bar{\mathbf{v}}_i\,] =^? \sum_{i\in[m]}[\,\bar{\mathbf{w}}_i\,] \tag{2}$$

One-round client-server interaction. Upon providing inputs to a confidential contract execution, clients can go off-line and retrieve confidential outputs with *Withdraw* at any later point in time.

Collateral. Our need for collateral follows the same logic as in Insured MPC [7]. The collateral contract incentivizes the servers to continue to participate in the privacy preserving smart contract computation, and behave honestly as they would otherwise suffer a financial loss. While the underlying maliciously secure MPC system will ensure that a server acting maliciously will cause an abort except with negligible probability, such an abort the adversary might have learned the output of the computation. This can in some situations have high value. Thus we require each server to give as collateral, strictly *more* than the maximum value they could gain from learning the output of a privacy preserving computation.

3.1 Realizing the confidential contract functionality

Overview of Protocol. Having provided a high-level overview of the protocol phase in Section 1, we now proceed to detail the individual protocol phases for the single-round privacy preserving smart contract execution and refer to Appendix D.2 for the full protocol description and UC-security proof, and to Sec. E for an outline of the multi-round protocol.

Setup of contracts. Servers deploy instances of $\mathcal{X}_{\mathsf{Lock}}[\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$, $\mathcal{X}_{\mathsf{Collateral}}$ on $\mathcal{F}_{\mathsf{Ledger}}$. Since wrapper $\mathcal{X}_{\mathsf{Lock}}$ extends $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, both are deployed and initialized as a single contract instance on $\mathcal{F}_{\mathsf{Ledger}}$ with shared contract id $(\mathsf{cn}_{\mathsf{Lock}})$ and shared state such as the confidential ledger $(\mathcal{L}_{\mathsf{Conf}})$. Here, the function of $\mathcal{X}_{\mathsf{Lock}}$ is to lock the confidential coins of clients input to the confidential contract evaluation, and to *replace* these with a new confidential distribution according to result of the contract evaluation. Further, $\mathcal{X}_{\mathsf{Lock}}$ is initialized with a threshold signature verification key $\mathsf{vk}_{\mathsf{TSig}}$, jointly generated by all servers via $\mathcal{F}_{\mathsf{TSig}}$: whenever servers agree on a new status of the contract evaluation in $\mathcal{F}_{\mathsf{Ident}}$, this agreement can be settled in $\mathcal{X}_{\mathsf{Lock}}$ with a threshold signature jointly generated via global functionality $\mathcal{F}_{\mathsf{TSig}}$. $\mathcal{X}_{\mathsf{Collateral}}$ is parameterized by $\mathsf{cn}_{\mathsf{Lock}}$ and is activated each time

$\mathcal{F}_{\mathsf{Clock}}$ progresses: it obtains collateral from all participating servers. It observes any recorded cheating servers $\mathcal{J}$ stored in the state of contract instance $\mathsf{cn}_{\mathsf{Lock}}$ and enforces penalties accordingly.

Client enrollment. Clients interact with $\mathcal{X}_{\mathsf{Lock}}$ to enroll a confidential coin it controls to the contract evaluation, and send both the coin commitment opening and numerical input x to an instance of $\mathcal{F}_{\mathsf{Ident}}$. Enrolled coins are removed from the confidential ledger $\mathcal{L}_{\mathsf{Conf}}$ maintained by $\mathcal{X}^{\underline{\mathsf{C}}\mathsf{Ledger}}$ and moved to a dedicated ledger $\mathcal{L}_{\mathsf{Lock}}$ for funds committed to a pending MPC computation in $\mathcal{F}_{\mathsf{Ident}}$.

Clients must also commit to a *output mask* during enrollment, which enables the subsequent redistribution of confidential coins without client interaction in the output phase of the contract evaluation. Here each client with confidential coin input c and numerical input x performs the following:

- Samples $\hat{y} \leftarrow\$ \mathbb{F}$ as a numerical output mask and sends to $\mathcal{F}_{\mathsf{Ident}}$.
- Samples $\hat{\mathbf{w}} \leftarrow\$ \mathbb{F}^{|\mathbb{T}|}$, $\hat{s} \leftarrow\$ \mathbb{F}$, and computes mask commitment $\hat{c} \leftarrow \mathsf{com}(\hat{\mathbf{w}}, \hat{s})$.
- Sends mask commitment $\hat{c}$ to $\mathcal{X}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$.
- Sends mask commitment openings $(\hat{\mathbf{w}}, \hat{s})$ of $\hat{c}$ to $\mathcal{F}_{\mathsf{Ident}}$.

Here clients can also give any auxiliary input, x, needed for the privacy preserving smart contract computation.

Input verification. Upon enrollment of clients, servers must verify that the confidential coin c and mask commitment $\hat{c}$ sent to $\mathcal{X}_{\mathsf{Lock}}$ are consistent with their respective openings $(\bar{\mathbf{v}}, \bar{r})$ and $(\hat{\mathbf{v}}, \hat{r})$ sent to $\mathcal{F}_{\mathsf{Ident}}$ during enrollment. For simplicity of presentation, we illustrate the batched input verification of input confidential coins and their openings assuming a token universe size of $|\mathbb{T}| = 1$, such that $c = g^{\bar{v}} h^{\bar{r}}$. Input verification for output masks $\hat{c}$ and their openings submitted to $\mathcal{F}_{\mathsf{Ident}}$ follow similarly.

$$\begin{tikzcd}
 & \text{Client} \arrow[dl, "c"'] \arrow[dr, "(\bar{v}', \bar{r}')"] & \\
\mathcal{X}_{\mathsf{Lock}} \arrow[rr, dashed, leftrightarrow, "c \overset{?}{=} \mathsf{com}(\bar{v}', \bar{r}')"'] \arrow[dr, "c"'] & & \mathcal{F}_{\mathsf{Ident}} \arrow[dl, "(\bar{v}'^{(i)}, \bar{r}'^{(i)})"] \\
 & \text{Server} &
\end{tikzcd}$$

Each server obtains both confidential coin c from $\mathcal{X}_{\mathsf{Lock}}$ and *additive shares* of submitted openings thereof from $\mathcal{F}_{\mathsf{Ident}}$, namely $(\bar{v}'^{(i)}, \bar{r}'^{(i)})$. We write $\bar{v}'^{(i)} = (\bar{v}+\epsilon)^{(i)}$ and similarly for $\bar{r}'^{(i)}$, where the ϵ denotes the error or discrepancy that the adversary can introduce to $\bar{v}$. We employ a standard technique of evaluating a random linear combination over client inputs to verify consistency.

1. Servers jointly sample $\gamma, \alpha, \beta \leftarrow\$ \mathbb{F}$ and open γ.
2. Each server locally computes the following on the inputs from m clients.
   - $\bar{v}'^{(i)}_{\mathsf{lin}} = \alpha^{(i)} + \gamma\, \bar{v}'^{(i)}_1 + \ldots + \gamma^m\, \bar{v}'^{(i)}_m$ and $r'^{(i)}_{\mathsf{lin}} = \beta^{(i)} + \gamma\, r'^{(i)}_1 + \ldots + \gamma^m\, r'^{(i)}_m$
   - Subsequently, it sends $\bar{v}'^{(i)}_{\mathsf{lin}}$ and $\bar{v}'^{(i)}_{\mathsf{lin}}$ to all other servers.
3. Each server locally reconstructs $\bar{v}'_{\mathsf{lin}} = \prod_{i\in[n]} \bar{v}'^{(i)}_{\mathsf{lin}}$ and $r'_{\mathsf{lin}} = \prod_{i\in[n]} r'^{(i)}_{\mathsf{lin}}$
4. Servers locally verify: $\prod_{i\in[n]} g^{\alpha^{(i)}} h^{\beta^{(i)}} \prod_{j\in[m]} c_j^{\gamma^j} \overset{?}{=} g^{\bar{v}'_{\mathsf{lin}}} h^{r'_{\mathsf{lin}}}$

Note that $\bar{v}'^{(i)}_{\mathsf{lin}}$ and $r'^{(i)}_{\mathsf{lin}}$ are shares held by servers and do not reveal the values of user inputs. We write $\bar{v}'_{\mathsf{lin}} = \alpha + \gamma\,(\bar{v}_1 + \epsilon_{\bar{v}_1}) + \ldots + \gamma^m\,(\bar{v}_m + \epsilon_{\bar{v}_m})$ and similarly for r'_{lin} to expose ϵ's introduced by the adversary. If ϵ values are committed to by the adversary before α, β, γ are sampled, we can interpret $\bar{v}'_{\mathsf{lin}} - \bar{v}_{\mathsf{lin}} = 0$ and $r'_{\mathsf{lin}} - r_{\mathsf{lin}} = 0$ as m - degree polynomials with coefficients chosen by the adversary that are later evaluated at some random coordinate γ: since verification step (4) implies exactly these assertions, the probability for an undetected non-zero error is therefore $m/|\mathbb{G}|$, where m is the number of polynomial roots, by the Schwartz-Zippel Lemma.

Execute. Servers call the **Evaluate** interface on $\mathcal{F}_{\mathsf{Ident}}$ to evaluate circuit g with input gates set to client inputs.

$$([\,x_1\,], [\,\hat{y}_1\,], [\,\bar{\mathbf{v}}_1\,], [\,r_1\,], [\,\hat{\mathbf{w}}_1\,], [\,\hat{s}_1\,]), ..., ([\,x_m\,], [\,\hat{y}_m\,], [\,\bar{\mathbf{v}}_m\,], [\,r_m\,], [\,\hat{\mathbf{w}}_m\,], [\,\hat{s}_m\,])$$

Upon secure evaluation, outputs in form of numerical values and balances are written to the output gates of g: $\big(([\,y_1\,], [\,\bar{\mathbf{w}}_1\,]), ..., ([\,y_m\,], [\,\bar{\mathbf{w}}_m\,])\big)$. Before masking these for opening, the servers then perform a confidential consistency check to ensure the preservation of tokens as shown in Equation (2).

Masked output values are obtained by applying the masking values input by users, $[\,y'_j\,] = [\,y_j\,] + [\,\hat{y}_j\,]$ and similarly for balances, $[\,\bar{\mathbf{w}}'_j\,] = [\,\bar{\mathbf{w}}_j\,] + [\,\hat{\mathbf{w}}_j\,]$ and generating a joint signature $\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{evaled})$ via $\mathcal{F}_{\mathsf{TSig}}$, that is sent to $\mathcal{X}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$. Upon verification, the $\mathcal{X}_{\mathsf{Lock}}$ contract updates the state of protocol execution, reflecting completion of the *Execute* phase.

Open. Servers run **Optimistic Reveal** in $\mathcal{F}_{\mathsf{Ident}}$ to open masked numerical outputs and balances $\big((y'_1, \bar{\mathbf{w}}'_1), ..., (y'_m, \bar{\mathbf{w}}'_m)\big)$. Should all servers agree on the successful completion of the contract evaluation, they jointly sign all *masked* outputs and send these to $\mathcal{X}_{\mathsf{Lock}}$ (on $\mathcal{F}_{\mathsf{Ledger}}$), which then computes the *unmasked* confidential coins for clients with the newly computed distribution as follows. Given the masked output balance $\bar{\mathbf{w}}'$ from $\mathcal{F}_{\mathsf{Ident}}$ and the coin mask $\hat{c}$ sampled by a client in **Enroll**, contract $\mathcal{X}_{\mathsf{Lock}}$ computes

(a) The masked confidential coin: $c^{\mathsf{out}\,\prime} \leftarrow \mathbf{g}^{\bar{\mathbf{w}}'} h^0$
(b) The <u>un</u>masked confidential coin: $c^{\mathsf{out}} \leftarrow c^{\mathsf{out}\,\prime} \cdot \hat{c}^{-1}$

We rewrite (b) as $c^{\mathsf{out}} = \mathbf{g}^{\bar{\mathbf{w}}' - \hat{\mathbf{w}}} h^{-\hat{s}} = \mathsf{com}(\bar{\mathbf{w}}, -\hat{s})$ to expose the unmasking of the output coin without any knowledge of the final balance. $\mathcal{X}_{\mathsf{Lock}}$ subsequently stores unmasked output coin c^{out} in the confidential ledger in $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, thereby settling the output balance distribution read from output gates of contract circuit g. Should $\mathcal{X}_{\mathsf{Lock}}$ successfully verify the signed outputs, $\mathcal{X}_{\mathsf{Collateral}}$ will infer from the state of $\mathcal{X}_{\mathsf{Lock}}$ the completion of a successful round and return the deposited collateral to the servers.

Withdraw. Upon a successful **Open**, the output of the confidential contract evaluation has completed. Each client can obtain their masked output $(y', \bar{\mathbf{w}}')$ from $\mathcal{X}_{\mathsf{Lock}}$ and newly minted c_{out} from $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ anytime following a successful execution of a contract evaluation. Let $\hat{y}$ and $(\hat{\mathbf{w}}, \hat{s})$ be the output masks generated by the client in **Enroll**. The withdrawing client obtains

(a) The numerical output: $y \leftarrow y' - \hat{y}$

(b) The opening of the output coin: $(\bar{\mathbf{w}}, s) \leftarrow (\bar{\mathbf{w}}' - \hat{\mathbf{w}}, -\hat{s})$

Thus, their the tokens are still confidential and that clients can transfer or redeem these using $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in Fig. 10.

Abort. If the protocol aborts prior to the completion of the **Execute** phase, client funds are simply returned by $\mathcal{X}_{\mathsf{Lock}}$ and collateral deposited to $\mathcal{X}_{\mathsf{Collateral}}$ is returned. If servers have agreed upon the completion of **Execute**, honest servers can interact with $\mathcal{F}_{\mathsf{Ident}}$ to either (a) obtain shares that are verifiable and enable reconstruction of the output or (b) identify cheating servers (Figure 7). Thus, $\mathcal{X}_{\mathsf{Lock}}$ as a registered public verifier, can identify cheating servers by either verifying shares with $\mathcal{F}_{\mathsf{Ident}}$, or obtaining the identities of servers $\mathcal{J}$ that refuse to participate in revealing their shares and allowing their verification. Cheating servers lose their collateral held by $\mathcal{X}_{\mathsf{Collateral}}$ which is redistributed to clients.

We present the full protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ which GUC-realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ in Appendix D.2 and prove the following statement.

Theorem 2. *$\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}[\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}[\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{TSig}}$-hybrid model against any PPT-adversary corrupting at most $n-1$ of the n servers $\mathcal{P}$ statically and any minority of $\mathcal{Q}$.*

4 Efficiency

We note that since previous works focus on using zero knowledge proofs and a trusted contract manager, we refrain from directly comparing our efficiency to their works. The closest previous works to ours is the Hawk family [44,4,5]. Unfortunately neither of the works provide an efficiency analysis, making it hard to provide a meaningful comparison. However, we note they all require computation of cryptographic primitives (commitments and ZKPs) in MPC. Thus requiring strictly more MPC computation, *along* with a larger (and hence) slower field of computation, as this field is needed to facilitate *computational* security of the cryptographic primitives they compute in MPC. In the following analysis, we assume Bulletproofs for range proofs and standard Fiat-Shamir Schnorr proofs of knowledge of exponents using elliptic curves. Although neither of these are UC-secure since knowledge extraction requires rewinding, there is evidence [34] that these techniques can be made non-malleable in the algebraic group model. Hence, for the purpose of efficiency we believe it is reasonable to forgo the formal UC security in this section. We use BLS threshold signatures and for simplicity we assume the size of the group used for BLS and commitments is the same, although it will in practice be slightly larger for BLS.

We outline the amount of heavy computations needed for our core protocol in Table 2, *except* what is needed by the underlying MPC computation computing the contract circuit g, reflecting the privacy preserving smart contract $\underline{\mathsf{C}}\mathsf{Contract}$. Concretely we count the amount of group exponentiations when assuming that the Pedersen commitments are realized using elliptic curves, along with pairings assuming BLS [14] has been used for realizing distributed signatures. The table only contains the complexity of executing one instance of

		Init	Execution	Abort
User	**exp**	2	2	0
Server	**exp**	$2 + 2(n-1)$	$6\lvert\mathcal{C}\rvert + 2$	0
	pair	0	$n-1$	0
	mult	0	$z\lvert\mathcal{C}\rvert$	0
SC comp.	**exp**	0	$2\lvert\mathcal{C}\rvert z$	$\lvert\mathcal{C}\rvert$
	pair	0	2	0
SC call space	$\#\mathbb{G}$ elem.	3	$\lvert\mathcal{C}\rvert z$	$O(n\lvert\mathcal{C}\rvert z)$
Comm.	$\#\mathbb{G}$ elem.	$O(n)$	$O(n^2 \cdot z \cdot \lvert\mathcal{C}\rvert)$	$O(n^2 \cdot z \cdot \lvert\mathcal{C}\rvert)$

Table 2: Complexity of our protocol when executing one $\underline{\mathsf{C}}\mathsf{Contract}$, *excluding* the computation of contract circuit g in MPC. We assume $|\mathcal{C}|z > s$ for statistical security parameter s, where z is the amount of input/output for each client in the set of clients $\mathcal{C}$, including the hidden token amount. $n = |\mathcal{P}|$ is the amount of servers and **mult** denotes the number of multiplications in MPC.

$\underline{\mathsf{C}}\mathsf{Contract}$, but we note that execution of multiple contracts is slightly sublinear in the complexity of a single execution. The *Abort* column illustrates the *additional* overhead associated with a cheating party.

	Mint	ConfTransfer	Redeem
User	4	$O(\log(\bar{v}_{\mathsf{max}}) \cdot \log(\log(\bar{v}_{\mathsf{max}})))$	3
SC comp.	3	$O(\log(\bar{v}_{\mathsf{max}}))$	3
SC space	3	$2\log(\bar{v}_{\mathsf{max}}) + 10$	4

Table 3: Complexity of $\underline{\mathsf{C}}\mathsf{Ledger}$ in group exponentiation and amount of group elements stored, when $\bar{v}_{\mathsf{max}}$ is the maximum amount of allowed tokens (Recall $|\mathcal{C}| \cdot \bar{v}_{\mathsf{max}} < |\mathbb{G}|$).

When it comes to our confidential token layer, we outline the complexity in Table 3. We note that the constant in the complexity of *Confidential Transfer* reflects two range proofs over $\log(|\mathbb{G}|/2)$, under the assumption that BulletProofs are used [17]. Although if the domain of the token amounts is further limited from $\mathbb{G}$ to $\bar{v}_{\mathsf{max}} < |\mathbb{G}|/|\mathcal{C}|$ then they can be reduced to range proofs of $[0; \bar{v}_{\mathsf{max}} - 1]$ and thus complexity $O(\bar{v}_{\mathsf{max}} \cdot \log(\bar{v}_{\mathsf{max}}))$.

In both tables the amount of smart contract space is only what needs to be submitted. The persistent space use needed is only $3 + 3|\mathcal{C}|$ group elements, if we assume that the storage used when posting to $\mathcal{X}_{\mathsf{Lock}}$ in *evaluate* and *open* gets overwritten the next time the servers call these methods.

The round complexity for all steps of both the confidential token layer protocols and our core protocol is constant, assuming g has constant multiplicative depth. Otherwise, the computation of g dominates the round complexity.

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Fair two-party computations via bitcoin deposits. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014 Workshops. LNCS, vol. 8438, pp. 105–121. Springer, Heidelberg (Mar 2014). https://doi.org/10.1007/978-3-662-44774-1_8
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.35
3. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Annual international cryptology conference. pp. 324–356. Springer (2017), https://doi.org/10.1007/978-3-319-63688-7_11
4. Banerjee, A., Clear, M., Tewari, H.: zkhawk: Practical private smart contracts from mpc-based hawk. In: 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS). pp. 245–248. IEEE (2021), https://doi.org/10.1109/BRAINS52497.2021.9569822
5. Banerjee, A., Tewari, H.: Multiverse of HawkNess: A Universally-Composable MPC-based Hawk Variant. Cryptology ePrint Archive (2022), https://eprint.iacr.org/2022/421
6. Baum, C., Chiang, J., David, B., Frederiksen, T., Gentile, L.: Sok: Mitigation of front-running in decentralized finance. The 2nd Workshop on Decentralized Finance (DeFi'22) (01 2022)
7. Baum, C., David, B., Dowsley, R.: Insured MPC: Efficient secure computation with financial penalties. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 404–420. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_22
8. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: CRAFT: Composable randomness and almost fairness from time. Cryptology ePrint Archive, Report 2020/784 (2020), https://eprint.iacr.org/2020/784
9. Baum, C., David, B., Frederiksen, T.K.: P2DEX: Privacy-preserving decentralized cryptocurrency exchange. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 21, Part I. LNCS, vol. 12726, pp. 163–194. Springer, Heidelberg (Jun 2021). https://doi.org/10.1007/978-3-030-78372-3_7
10. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
11. Benhamouda, F., Halevi, S., Halevi, T.: Supporting private data on hyperledger fabric with secure multiparty computation. IBM Journal of Research and Development **63**(2/3), 3–1 (2019), https://doi.org/10.1147/JRD.2019.2913621
12. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_24
13. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 410–440. Springer, Heidelberg (Dec 2017). https://doi.org/10.1007/978-3-319-70697-9_15

14. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. Journal of Cryptology **17**(4), 297–319 (Sep 2004). https://doi.org/10.1007/s00145-004-0314-9
15. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: ZEXE: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy. pp. 947–964. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00050
16. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 423–443. Springer, Heidelberg (Feb 2020). https://doi.org/10.1007/978-3-030-51280-4_23
17. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
18. Camenisch, J., Lehmann, A., Lysyanskaya, A., Neven, G.: Memento: How to reconstruct your secrets from a single password in a hostile environment. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 256–275. Springer, Heidelberg (Aug 2014). https://doi.org/10.1007/978-3-662-44381-1_15
19. Campanelli, M., Hall-Andersen, M.: Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In: Suga, Y., Sakurai, K., Ding, X., Sako, K. (eds.) ASIACCS 22. pp. 652–666. ACM Press (May / Jun 2022). https://doi.org/10.1145/3488932.3517424
20. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
21. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001), https://doi.org/10.1109/SFCS.2001.959888
22. Canetti, R.: Universally composable signature, certification, and authentication. In: 17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA. p. 219. IEEE Computer Society (2004). https://doi.org/10.1109/CSFW.2004.24, http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24
23. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography Conference. pp. 61–85. Springer (2007), https://doi.org/10.1007/978-3-540-70936-7_4
24. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P) (2019). https://doi.org/10.1109/EuroSP.2019.00023
25. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 719–728. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134092
26. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA. pp. 364–369. ACM (1986). https://doi.org/10.1145/12130.12168, https://doi.org/10.1145/12130.12168

27. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy. pp. 910–927. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00040
28. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 169–187. Springer, Heidelberg (Feb 2016)
29. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In: European Symposium on Research in Computer Security. pp. 1–18. Springer (2013), https://doi.org/10.1007/978-3-642-40203-6_1
30. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_38
31. David, B., Dowsley, R., Larangeira, M.: Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 500–519. Springer, Heidelberg (Feb / Mar 2018). https://doi.org/10.1007/978-3-662-58387-6_27
32. David, B., Gentile, L., Pourpouneh, M.: FAST: Fair auctions via secret transactions. In: Ateniese, G., Venturi, D. (eds.) ACNS 22. LNCS, vol. 13269, pp. 727–747. Springer, Heidelberg (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_36
33. da Gama, M.B., Cartlidge, J., Polychroniadou, A., Smart, N.P., Alaoui, Y.T.: Kicking-the-bucket: Fast privacy-preserving trading using buckets. Cryptology ePrint Archive (2021), to appear at FC'22. https://eprint.iacr.org/2021/1549
34. Ganesh, C., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 397–426. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_14
35. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), https://doi.org/10.1007/978-3-662-46803-6_9
36. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. Journal of the ACM (JACM) **59**(3), 1–35 (2012), https://doi.org/10.1145/2220357.2220358
37. Jakobsen, T.P., Nielsen, J.B., Orlandi, C.: A framework for outsourcing of secure computation. In: Ahn, G., Oprea, A., Safavi-Naini, R. (eds.) Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, CCSW '14, Scottsdale, Arizona, USA, November 7, 2014. pp. 81–92. ACM (2014). https://doi.org/10.1145/2664168.2664170
38. Kalodner, H.A., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: Enck, W., Felt, A.P. (eds.) USENIX Security 2018. pp. 1353–1370. USENIX Association (Aug 2018)
39. Kanjalkar, S., Zhang, Y., Gandlur, S., Miller, A.: Publicly auditable mpc-as-a-service with succinct verification and universal setup. In: IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna,

Austria, September 6-10, 2021. pp. 386–411. IEEE (2021). https://doi.org/10.1109/EuroSPW54576.2021.00048, https://doi.org/10.1109/EuroSPW54576.2021.00048
40. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Theory of Cryptography Conference. pp. 477–498. Springer (2013), https://doi.org/10.1007/978-3-642-36594-2_27
41. Kerber, T., Kiayias, A., Kohlweiss, M.: KACHINA - foundations of private smart contracts. In: Küsters, R., Naumann, D. (eds.) CSF 2021 Computer Security Foundations Symposium. pp. 1–16. IEEE Computer Society Press (2021). https://doi.org/10.1109/CSF51468.2021.00002
42. Khovratovich, D., Law, J.: Sovrin: digital identities in the blockchain era. Tech. rep., Sovrin Foundation, accessed: 2022-10-13
43. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_25
44. Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy. pp. 839–858. IEEE Computer Society Press (May 2016). https://doi.org/10.1109/SP.2016.55
45. Kumaresan, R., Bentov, I.: Amortizing secure computation with penalties. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 418–429. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978424
46. Kumaresan, R., Moran, T., Bentov, I.: How to use bitcoin to play decentralized poker. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 195–206 (2015), https://doi.org/10.1145/2810103.2813712
47. Kumaresan, R., Vaikuntanathan, V., Vasudevan, P.N.: Improvements to secure computation with penalties. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 406–417. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978421
48. Lee, J., Nikitin, K., Setty, S.T.V.: Replicated state machines without replicated execution. In: 2020 IEEE Symposium on Security and Privacy. pp. 119–134. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00068
49. Maram, D., Malvai, H., Zhang, F., Jean-Louis, N., Frolov, A., Kell, T., Lobban, T., Moy, C., Juels, A., Miller, A.: CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In: 2021 IEEE Symposium on Security and Privacy. pp. 1348–1366. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00038
50. Maxwell, G.: Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt (2016)
51. Nilsson, A., Bideh, P.N., Brorsson, J.: A survey of published attacks on intel SGX. CoRR **abs/2006.13598** (2020), https://arxiv.org/abs/2006.13598
52. Noether, S.: Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Paper 2015/1098 (2015), https://eprint.iacr.org/2015/1098, https://eprint.iacr.org/2015/1098
53. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Report 2021/1530 (2021), https://eprint.iacr.org/2021/1530

54. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991). https://doi.org/10.1007/3-540-46766-1_9
55. Poelstra, A., Back, A., Friedenbach, M., Maxwell, G., Wuille, P.: Confidential assets. In: International Conference on Financial Cryptography and Data Security. pp. 43–63. Springer (2018), https://doi.org/10.1007/3-540-36178-2_26
56. Reistad, T.I., Toft, T.: Linear, constant-rounds bit-decomposition. In: Lee, D.H., Hong, S. (eds.) Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5984, pp. 245–257. Springer (2009). https://doi.org/10.1007/978-3-642-14423-3_17, https://doi.org/10.1007/978-3-642-14423-3_17
57. van Saberhagen, N.: CryptoNote v 2.0. https://web.archive.org/web/20201028121818/https://cryptonote.org/whitepaper.pdf (2013)
58. Sergio_Demian_Lerner: P2ptradex: P2p trading between cryptocurrencies. https://bitcointalk.org/index.php?topic=91843.0 (2012), accessed: 2022-10-13
59. Solomon, R., Almashaqbeh, G.: smartFHE: Privacy-preserving smart contracts from fully homomorphic encryption. Cryptology ePrint Archive, Report 2021/133 (2021), https://eprint.iacr.org/2021/133
60. Steffen, S., Bichsel, B., Baumgartner, R., Vechev, M.: ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1543–1543. IEEE Computer Society (2022), https://files.sri.inf.ethz.ch/website/papers/sp22-zeestar.pdf
61. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.T.: zkay: Specifying and enforcing data privacy in smart contracts. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 1759–1776. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3363222
62. Team, T.S.N.: Secret network: A privacy-preserving secret contract & decentralized application platform. https://scrt.network/graypaper (2022)
63. Xiong, A.L., Chen, B., Zhang, Z., Bünz, B., Fisch, B., Krell, F., Camacho, P.: VERI-ZEXE: Decentralized private computation with universal setup. Cryptology ePrint Archive, Report 2022/802 (2022), https://eprint.iacr.org/2022/802

A Extended related work

Privacy preserving transactions. The notion of confidential tokens was first proposed by Maxwell in [50], and subsequently formalized in [55]. Here, balances are concealed in additively homomorophic Pedersen commitments [54]. A valid transfer consumes and produces new commitments and is accompanied by a non-interactive range-proof [17] that each newly coin balance is below a threshold preventing the minting of tokens. Thus, if input and output commitments sum up, the ledger can verify that the coin supply was preserved. ZCash [10] breaks any relation between spent and newly minted tokens with general zkSNARK's proving both balance ranges and that confidential input coins have not previously been spent. Subsequent works [19] have improved on the complexity of NIZK's required. Zether [16] proposes a scheme which can be implemented on any public ledger with EVM-like smart contract machine, encrypting user balances with additive Elgamal encryption: importantly, for smaller balance ranges, this scheme permits the receiver to obtain the *opening* of privately received coins by brute-forcing an discrete-logarithm. Thus, unlike in other schemes, a sender does not need forward the private coin opening to the receiver. A weaker notion of privacy is achieved by Monero [57,52], which offers k-anonymity for senders.

Privacy preserving smart contracts. Hawk [44] introduces a general notion of smart contract evaluation over *private individual inputs* resulting in *private individual outputs*. The contract is evaluated over private numerical and private financial values, and is settled on a confidential ledger. Correctness and privacy is archived through the use of non-interactive zero-knowledge proofs. Thus, each party learns nothing about the inputs or outputs of other parties, other than what is implied by its own numerical and financial output. Although the notion of privacy here is limited to ledger and inter-party privacy, as a trusted third party is still required to carry out the private computation. This is quite different from what we normally consider private in the cryptographic community. Hence continuous research has been carried out, trying to remove this trusted third party, although it seems to require expensive, general non-interactive-zero-knowledge (NIZK) proofs to be computed inside the MPC function evaluation to permit private settlement on a confidential ledger. A recent line of work has by Banerjee *et al.* [4,5] contributes concrete efficiency improvements in realizing the private smart contract notion of Hawk by distributing the trusted party using MPC and by reducing the complexity of NIZK's which are computed inside the MPC. Recently, Kanjalkar *et al.* [39] present an optimized ZK protocol to be proven inside MPC.

Zexe [15] extends the ZCash model of private transactions with a private, state-less contract model. Bitcoin Script-like contracts are hidden, as are contract inputs, enabling limited contracting functionality with privacy. Furthermore, Zexe requires the execution of a trusted setup phase for *each* application. However, this need was removed in the follow-up VeriZexe [63].

A notion of smart contracts with *data privacy* is proposed in zkay [61]. Here, *private data* is encrypted on the blockchain, and NIZKs are used to prove that any modifications are done correctly. Follow-up work, Zeestar [60] uses additively homomorphic encryption to allow for *limited* private computation on data from multiple owners, without them having to share their private data with each other. SmartFHE [59] on the other hand uses FHE and NIZKPs construct a blockchain with support for privacy preserving transactions and general privacy preserving smart contracts. Their idea is to have each user setup an FHE scheme associated with their account. Every time they use the tokens in their account, they use the FHE scheme to perform the needed computation on their tokens and any auxiliary input and prove in zero-knowledge this was done correctly, before posting everything to the blockchain, for verification by a miner. Unfortunately this is rather inefficient, as simply validating a private transactions takes a miner more than 9 seconds. Furthermore, to compute fully private smart contracts with inputs from multiple parties they are required to expand their encrypted input to be encrypted under all public keys of the clients giving input to the smart contract. Thus requiring online interaction between all the clients with relevant data to the computation. Furthermore, in general this line of work does not explicitly provide privacy between contract participants, as a party which holds the encryption key can trivially observe the contract evaluation in the clear.

Fair MPC with public ledgers. We describe two closely related lines of work that integrate MPC protocols with a ledger functionality to achieve (1) fair MPC protocols, which identify and penalize cheating parties and (2) private smart contracts executed inside a MPC instance which finalize the *confidential* outcome on the ledger.

The first works to utilize the Bitcoin ledger to achieve fairness in lottery games was introduced in [1,2], where cheating parties can abort upon learning the output first but incur a financial penalty without requiring a trusted party to arbitrate. This notion of output fairness was generalized to any secure function evaluation by Bentov *et al.* [12] and to the reactive setting [46]. Subsequent works improve the efficiency of output fairness [45,47,13,11,31], culminating in Insured MPC [7], which formulates a UC-secure MPC functionality with identifiable aborts. Another line of work [43,25] focuses on stronger notions of fairness, identifying aborting parties' prior to the output phase.

P2DEX. At ANCS 2021 Baum *et al.* [9] introduced P2DEX, which extends Insured MPC [7] by allowing for cross-chain communication and privacy preserving smart contract computation. Although their privacy preservation only involves auxiliary input, and *not* the token amounts. Their idea is to first have user send the tokens to a burner address, generated in a distributed manner by a set of servers. These servers then run an Insured MPC protocol which computes a private smart contract based on auxiliary input privately given by the users. Based on the result, appropriate amount of tokens can be transferred from the burner addresses to the users by having the servers threshold sign these transactions. The authors use this to make a system for decentralized cross-chain exchange, preventing *both* miner and operator front-running.

B Extended preliminaries

A model of smart contracts. $\mathcal{F}_{\mathsf{Ledger}}$ parses authenticated messages which can authorize the deployment and activation of smart contracts, each modelled with a state transition function encoded as an arithmetic circuit T of maximum depth d_T, thereby enforcing a notion of bounded termination. Each contract maintains a public state fragment $\gamma \in \{0,1\}^*$ that is updated by circuit T upon the evaluation of each authenticated CallContract message. Each contract also maintains a balance $\bar{\mathbf{w}}$ of $\mathbb{T}$. We sketch the evaluation of a smart contract call with parameters $\mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}}$ authorized by signature verification key vk:

- *Contract identifier* cn selects the contract instance for evaluation.
- *Function selector* fn is an input that identifies the *contract interface* being evaluated, facilitating the logical separation of contract descriptions.

- *Input string* $x \in \{0,1\}^*$ denotes parameters input to circuit T: it is logically evaluated by the contract interface selected by fn.
- *Token balance* $\bar{\mathbf{v}}$ is provided to the contract call and is subtracted from the ledger entry associated with verification key vk.

The circuit T associated with contract instance cn is then evaluated on input $(\nu \,|\, \gamma \,|\, \bar{\mathbf{w}} \,|\, \mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}}, \mathsf{vk})$, where ν denotes $\mathcal{F}_{\mathsf{Clock}}$ round at the time of the call, and γ denotes the contract state stored by $\mathcal{F}_{\mathsf{Ledger}}$. Upon completed evaluation of T, $\mathcal{F}_{\mathsf{Ledger}}$ reads the encoding of a state transition $\mathcal{L}|\gamma|\bar{\mathbf{w}} \rightarrow^{\mathsf{ts}} \mathcal{L}'|\gamma'|\bar{\mathbf{w}}'$ from the output gates of evaluated T, thereby updating ledger, contract state and contract balance. Here, $\mathcal{F}_{\mathsf{Ledger}}$ asserts token supplies over $\mathbb{T}$ are preserved and that non-calling account balances cannot decrease from applying update ts.

We note the presence of call-back gates permitted in contract circuits deployed to $\mathcal{F}_{\mathsf{Ledger}}$, related to a UC-modelling technicality described in more detail in Appendix C.1. Concretely, these gates permit the UC functionality $\mathcal{F}_{\mathsf{Ledger}}$ to forward verification calls to hybrid functionalities $\mathcal{F}_{\mathsf{Ident}}$ and $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ via an honest majority committee $\mathcal{Q}$. Thus, in the hybrid $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$-setting, the simulator maintains the ability to equivocate and efficiently extract inputs from dishonest parties.

Pedersen commitments. Let g, h denote random generators of $\mathbb{G}$ such that nobody knows the discrete logarithm of h base g, i.e., a value w such that $g^w = h$. The Pedersen commitment scheme [54] to an $s \in \mathbb{Z}_p$ is obtained by sampling $t \leftarrow\!\$\, \mathbb{Z}_p$ and computing $\mathsf{com}(s,t) = g^s h^t$. Hence, the commitment $\mathsf{com}(s,t)$ is a value uniformly distributed in $\mathbb{G}$ and opening the commitment requires to reveal the values of s and t. The Pedersen commitments are additively homomorphic, i.e., starting from the commitment to $s_1 \in \mathbb{Z}_p$ and $s_2 \in \mathbb{Z}_p$, it is possible to compute a commitment to $s_1 + s_2 \in \mathbb{Z}_p$, i.e., $\mathsf{com}(s_1, t_1) \circ \mathsf{com}(s_2, t_2) = \mathsf{com}(s + 1 + s_2, t_1 + t_2)$.

(Global) Universal Composability. In this work, the (Global) Universal Composability or (G)UC framework [21,23] is used to analyze security. Due to space constraints, we refer interested readers to the aforementioned works for more details. We generally use $\mathcal{F}$ to denote an ideal functionality and Π for a protocol. We implicitly assume private and authenticated channel between each pair of parties.

Several functionalities in this work allow public verifiability. Following Badertscher *et al.* [3] we dynamically allow the construction of a set of verifiers $\mathcal{V}$ through register and de-register commands. The adversary, $\mathcal{S}$ will always be allowed to obtain the list of registered verifiers. Concretely we implicitly assume all functionalities with public verifiability include the following interfaces (which are omitted in the concrete boxes for simplicity):

Register: Upon receiving (Register, *sid*) from some verifier $\mathcal{V}_i$, set $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_i$ and return (Registered, *sid*, $\mathcal{V}_i$) to $\mathcal{V}_i$.

Deregister: Upon receiving (Deregister,sid) from some verifier $\mathcal{V}_i$, set $\mathcal{V} = \mathcal{V} \setminus \mathcal{V}_i$ and return (Deregistered, *sid*, $\mathcal{V}_i$) to $\mathcal{V}_i$.

Is Registered: Upon receiving (Is-Registered, *sid*) from $\mathcal{V}_i$, return (Is-Registered, *sid*, b) to $\mathcal{V}_i$, where $b = 1$ if $\mathcal{V}_i \in \mathcal{V}$ and $b = 0$ otherwise.

Get Registered: Upon receiving (Get-Registered, *sid*) from the ideal adversary $\mathcal{S}$, the functionality returns (Get-Registered, *sid*, $\mathcal{V}$) to $\mathcal{S}$. The above instructions can also be used by other functionalities to register as a verifier of a publicly verifiable functionality.

Global clock. As some parts of our work are inherently synchronous, we model rounds using a global clock functionality $\mathcal{F}_{\mathsf{Clock}}$ as in [3,40,43]. In Fig. 3 we show the global UC clock functionality, $\mathcal{F}_{\mathsf{Clock}}$, we need, taken verbatim from the work of Baum *et al.* [9]. We note that in the real execution all parties will send messages to, and receive them, from $\mathcal{F}_{\mathsf{Clock}}$. Whereas in the simulated case only the ideal functionality, other global functionalities as well as the corrupted parties will do so. Throughout this work, we will write "update $\mathcal{F}_{\mathsf{Clock}}$" as a short-hand for "send (Update, *sid*) to $\mathcal{F}_{\mathsf{Clock}}$".

Functionality $\mathcal{F}_{\mathsf{Clock}}$

$\mathcal{F}_{\mathsf{Clock}}$ is parameterized by a variable ν, sets $\mathcal{P}$,$\mathcal{F}$ of parties and functionalities respectively. It keeps a Boolean variable d_J for each $J \in \mathcal{P} \cup \mathcal{F}$, a counter ν as well as an additional variable u. All d_J, ν and u are initialized as 0.

Clock update: Upon receiving a message (Update) from $J \in \mathcal{P} \cup \mathcal{F}$:
1. Set $d_J = 1$.
2. If $d_F = 1$ for all $F \in \mathcal{F}$ and $d_P = 1$ for all honest $P \in \mathcal{P}$, then set $u \leftarrow 1$ if it is 0.

Clock read: Upon receiving a message (Read) from any entity:
1. If $u = 1$ then first send (Tick, *sid*) to $\mathcal{S}$. Next set $\nu \leftarrow \nu + 1$, reset d_J to 0 for all $J \in \mathcal{P} \cup \mathcal{F}$ and reset u to 0.
2. Answer the entity with (Read, ν).

Fig. 3: Global UC functionality $\mathcal{F}_{\mathsf{Clock}}$ for the clock.

Signatures. We will implicitly assume access to a global digital signature ideal functionality $\mathcal{F}_{\mathsf{Sig}}$ as defined in [22] (where it is also shown any EUF-CMA signature scheme realizes it), which is used for signing transactions to a ledger. We also use a global UC secure threshold signature scheme which offers identifiable abort. We denote this functionality $\mathcal{F}_{\mathsf{TSig}}$ and define it in Fig. 4 (which is taken verbatim from the work of Baum *et al.* [9]). The functionality allows a set of n parties to collaboratively sign a message m, and allows the adversary to corrupt up to $n-1$ parties without being able to forge signatures. That is, we assume the full-threshold setting. Thus its behaviour matches that of $\mathcal{F}_{\mathsf{Sig}}$, although it additionally allows $\mathcal{S}$ to choose the string of shares that later get combined into a signature. Although under the constraint that $\mathcal{S}$ has to choose both the signature shares and the actual signature, σ, together. Although this allows $\mathcal{S}$ to always make a valid signature, it is never allowed to make an invalid signature in an honest execution of **Share Generation**. Based on the signature shares, the parties can learn σ from **Share Combination**, although if parties have been cheating in **Shares Generation** they will be exposed during **Share Combination**. We observe that the choice of shares binds $\mathcal{S}$ to a certain set of dishonest parties. Note that by assuming both $\mathcal{F}_{\mathsf{Sig}}$ and $\mathcal{F}_{\mathsf{TSig}}$ to be *global* UC functionalities, it allows other UC functionalities, both local and global, to verify signatures on them. This becomes essential to allow interaction with our, global, ledger functionalities.

Functionality $\mathcal{F}_{\mathsf{TSig}}$

$\mathcal{F}_{\mathsf{TSig}}$ is parameterized with an ideal adversary $\mathcal{S}$, a set of signers $\mathcal{P}$ and functionalities $\mathcal{F}$, a verifiers $\mathcal{V}$ (which automatically contains $\mathcal{P}$ and $\mathcal{F}$) and a set of corrupted signers $I \subset \mathcal{P}$. $\mathcal{F}_{\mathsf{TSig}}$ has two internal lists `Sh` and `Sig`.

Key Generation: Upon receiving a message $(keygen)$ from each $\mathcal{P}_i \in \mathcal{P}$ or a functionality $\mathcal{F}_j \in \mathcal{F}$ hand $(keygen)$ to the adversary $\mathcal{S}$. Upon receiving $(verificationkey)\mathsf{vk}$ from $\mathcal{S}$, if $(\cdot, \mathsf{vk})$ was not recorded yet then output $(verificationkey)\mathsf{vk}$ to each $\mathcal{P}_i \in \mathcal{P}$ (or to $\mathcal{F}_j$), and record the pair $(\mathcal{P}, \mathsf{vk})$. If vk was recorded before then output $(Abort)$ to $\mathcal{S}$ and stop.

Share Generation: Upon receiving a message $(sign)m, \mathsf{vk}$ from all honest parties or a functionality $\mathcal{F}_j \in \mathcal{F}$ send $(sign)m$ to $\mathcal{S}$. Upon receiving $(signature)m, \rho, \sigma, J, f$ from $\mathcal{S}$, verify that
- no entry $(m, \rho, J', \mathsf{vk}')$ with $J' \neq J$ is recorded in `Sh`, and
- no entry $(m, \sigma, \mathsf{vk}, 0)$ is recorded in `Sig` if $J = \emptyset$.

If either is, then output an error message to $\mathcal{S}$ and halt. Else, let $f' = 1$ if $J = \emptyset$ and $f' = f$ otherwise, record the entry $(m, \rho, J, \mathsf{vk})$ in `Sh`, $(m, \sigma, \mathsf{vk}, f')$ in `Sig` and return $(shares)m, \rho$.

Share Combination: Upon receiving a message $(combine)m, \rho, \mathsf{vk}$ from any party in $\mathcal{P}$ or functionality $\mathcal{F}_j \in \mathcal{F}$, find $(m, \rho, J, \mathsf{vk})$ in `Sh` and $(m, \sigma, \mathsf{vk}, b)$ in `Sig`. If $J \neq \emptyset$ then return $(Failure)J$. If $J = \emptyset$ return $(combined)m, \sigma, \mathsf{vk}$. If no entry could be found in `Sh` and `Sig` then return $(Not - Generated)$.

Signature Verification: Upon receiving a message $(verify)m, \sigma, \mathsf{vk}'$ from some entity in $\mathcal{V}$, hand $(verify)m, \sigma, \mathsf{vk}'$ to $\mathcal{S}$. Upon receiving $(verified)m, \phi$ from $\mathcal{S}$ do:
1. If $\mathsf{vk}' = \mathsf{vk}$ and $(m, \sigma, \mathsf{vk}, 1) \in$ `Sig`, then set $f = 1$.
2. Else, if $\mathsf{vk}' = \mathsf{vk}$ and $(m, \sigma', \mathsf{vk}, 1) \notin$ `Sig` for any σ', then set $f = 0$ and record the entry $(m, \sigma, \mathsf{vk}, 0)$ in `Sig`.
3. Else, if there is an entry $(m, \sigma, \mathsf{vk}', f') \in$ `Sig` recorded, then let $f = f'$.
4. Else, let $f = \phi$ and record the entry $(m, \sigma, \mathsf{vk}', \phi)$ in `Sig`.

Return $(verified)m, f$.

Fig. 4: Global UC functionality $\mathcal{F}_{\mathsf{TSig}}$ for Threshold Signatures.

Non-interactive zero-knowledge. We us non-interactive zero-knowledge arguments of knowledge, allowing any party to construct a proof that can later be validated by any verifier. We model this in the same way as done by Groth *et* *al.* [36] and formally define the functionality $\mathcal{F}_{\mathsf{NIZK}}$ for this in Fig. 5 in Sec. B. The functionality $\mathcal{F}_{\mathsf{NIZK}}$ allows any party to prove in zero knowledge that they know a witness w for a public statement x such that $(x, w) \in R$ for a NP relation R.

Functionality $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$

$\mathcal{F}_{\mathsf{NIZK}}$ interacts with parties $\mathcal{P} = \{P_1, ..., P_n\}$ and simulator $\mathcal{S}$ and is parameterized with relation R.

Proof: On input $(\textsc{Prove}, sid, x, w)$ from party P, ignore if $(x, w) \notin R$. Send $(\textsc{Prove}, x)$ to $\mathcal{S}$ and wait for answer $(\textsc{Proof}, \pi)$. Upon receiving the answer store (x, π) and send $(\textsc{Proof}, sid, \pi)$ to P.

Verification: On input $(\textsc{Verify}, sid, x, \pi)$ from $\mathcal{V}$, check whether (x, π) is stored. If not send $(\textsc{verify}, x, \pi)$ to $\mathcal{S}$ and wait for an answer $(\textsc{Witness}, w)$. Upon receiving the answer, check whether $(x, w) \in R$ and in that case, store (x, π). If (x, π) has been stored, return $(\textsc{Verification}, sid, 1)$ to $\mathcal{V}$, else return $(\textsc{Verification}, sid, 0)$.

Fig. 5: UC functionality $\mathcal{F}_{\mathsf{NIZK}}$ for Non-interactive Zero-Knowledge

MPC. Secure Multi-Party Computation (MPC) allows a set of mutually distrusting $\mathcal{P} = \{P_1, \ldots, P_n\}$ to compute any efficiently commutable function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ where each party P_i supplied private input x_i and received private output y_i. MPC guarantees that the only thing known to party P_i after the computation is x_i and y_i. Multiple security and computational models exist for this, but in this paper we will assume the arithmetic black box model, where computation is a directed acyclic graph of arithmetic operations over a finite field $\mathbb{F}$, where $|\mathbb{F}| = p \geq 2^s$. We assume the UC-security against a *static*, *active/malicious* adversary, who can corrupt up to $n-1$ parties and who may cause an abort at any point in the computation. We assume an MPC scheme, which is *reactive*, meaning that it is possible to compute $f(\cdot)$,and depending on the output, compute some other function $f'(\cdot)$ on the same input as $f(\cdot)$. This model can for example be realized by the SPDZ protocol [30]. For simplicity we will assume the bracket-notation, where the function to be computed is specified by arithmetic operations on hidden variables. Concretely we assume $[\cdot]$ expresses a value hidden in MPC and on which arithmetic computations can be carried out. I.e. $[x] \cdot [y] + [z]$, expresses the computation $x \cdot y + z$ of values $x, y, z \in \mathbb{F}$.

Outsourced MPC. Typically MPC in the setting we need require a non-constant amount of rounds of communication between *all* pairs of parties (depended on the function to compute). If we have many parties supplying input this can become prohibitively expensive. For this reason we introduce another model of MPC known as *outsourced MPC*. Jakobsen *et al.* [37] shows how to use information theoretic operation in conjunction with *any* MPC scheme as described above, to allow a large set of clients $\mathcal{C} = \{C_1, \ldots, C_m\}$ to supply private input to an MPC computation, executed by a small set of servers $\mathcal{P} = \{P_1, \ldots, P_n\}$, and receive private output. Crucially the clients only need to execute a few lightweight operations, bounded by their amount of inputs and outputs, and only need to communicate with the servers in a constant amount of rounds.

Insured MPC. It has been shown [26] that it is impossible to achieve *fairness* in MPC when more than $n/2$ of the parties are corrupted. By fairness we mean that if one party learns their output of the computation, so does the rest of the parties. This is a problem since the party learning the output may be malicious and thus maybe abort the protocol based on what they learned. Baum *et al.* [7] show how to incentivize the completion of an MPC protocol, in a public verifiable manner, through financial incentives enforced on a public ledger. Specifically they showed this is possible to do, based on any MPC scheme fitting the model discussed above. We combine this incentivized notion of MPC with the outsourced notion of MPC in the functionality $\mathcal{F}_{\mathsf{Ident}}$. Concretely this specifies an out-sourced MPC functionality where clients $\mathcal{C} = \{C_1, \ldots, C_m\}$ supply private input that is computed on in MPC by the servers $\mathcal{P} = \{P_1, \ldots, P_n\}$ and where the output of the computation is verifiably shared between the servers in such a manner that the shares can verified by an external verifier $\mathcal{V}$ after the completion of the protocol to identify any potential malicious behaviour. We refer to appendix B.1 for a detailed description of $\mathcal{F}_{\mathsf{Ident}}$ and its interaction with servers and clients.

B.1 Publicly Verifiable MPC Functionality $\mathcal{F}_{\mathsf{Ident}}$

We adopt $\mathcal{F}_{\mathsf{Ident}}$ from [7,9] but include the following extensions to its interface. Firstly, when realizing $\mathcal{F}_{\mathsf{Ident}}$ with a reactive MPC scheme such as [30,29], we can amend $\mathcal{F}_{\mathsf{Ident}}$ with a reactive interface as in $\mathcal{F}_{\mathsf{Online}}$ from [29], exposing arithmetic operations over secret values, each identified with a unique *vid*, which are *selectively* input or output to the parties.

In addition to a reactive interface, we permit clients to *securely input* values to $\mathcal{F}_{\mathsf{Ident}}$. This is realized with the secure client input protocol from [37], which permits MPC servers to verify the linear MAC of the client input inside a reactive MPC instance. We wrap this secure client input protocol inside $\mathcal{F}_{\mathsf{Ident}}$ (as in the security proof of [37]) to obtain an input interface which can be called by clients.

Theorem 3. *Functionality $\mathcal{F}_{\mathsf{Ident}}$ (in Figure 6) can be realized by a reactive secure computation scheme permitting linear operations for free and the secure client input protocol from [37].*

Proof. (Proof sketch) The original, non-interactive $\mathcal{F}_{\mathsf{Ident}}$ functionality from [7] can be extended with a reactive interface when realized with a reactive MPC scheme [30,29] with minor adaptations of the UC-proof in [7]. The secure client input protocol of [37] is information-theoretically secure, and can be instantiated with *any* MPC scheme with free linear operations, including [30,29], thereby realizing reactive and client input interfaces of $\mathcal{F}_{\mathsf{Ident}}$ in Figure 6.

Functionality $\mathcal{F}_{\mathsf{Ident}}$

For each session, $\mathcal{F}_{\mathsf{Ident}}$ interacts with servers $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, clients $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and also provides an interface to register external verifiers $\mathcal{V}$. $\mathcal{S}$ provides a set $I_{\mathcal{P}} \subset [n]$ of corrupt parties and $I_{\mathcal{C}} \subseteq [m]$ of corrupt clients. $\mathcal{F}_{\mathsf{Ident}}$ only interacts with $\mathcal{P}, \mathcal{C}, \mathcal{V}$ and $\mathcal{S}$[a] of the respective session sid.

Init: Upon first $(\textsc{Init}, sid)$ by all parties in $\mathcal{P}$ set $\mathtt{rev}, \mathtt{ver}, \mathtt{ref} \leftarrow \emptyset$.
Input: Upon first $(\textsc{Input}, sid, x)$ by $\mathcal{C}_j$, forward $(\textsc{Input}, sid, \mathcal{C}_j)$ to $\mathcal{S}$. If $\mathcal{S}$ continues, $\mathcal{F}_{\mathsf{Ident}}$ samples vid, stores (vid, x) and returns $(\textsc{Input}, sid, \mathcal{C}_j, vid)$ to all $\mathcal{P}$.
Evaluate: Upon first $(\textsc{Eval}, sid, g, vid_1, ..., vid_p)$ by all parties $\mathcal{P}$, if $(vid_1, ..., vid_p)$ have been stored internally:
1. Compute $x_{p+1}, ..., x_{p+q} \leftarrow g(x_1, ..., x_p)$, sample $vid_{p+1}, ..., vid_{p+q}$.
2. Store $(vid_{p+1}, x_{p+1}), ..., (vid_{p+q}, x_{p+q})$, return $(vid_{p+1}, ..., vid_{p+q})$ to all parties.

Get Shares: Upon first $(\textsc{GetShare}, sid, vid)$ by $\mathcal{P}_i \in \mathcal{P}$ and if (vid, x) is stored:
1. For $\mathcal{P}_i \in I_{\mathcal{P}}$, let $\mathcal{S}$ provide $s_{vid}^{(i)} \in \mathbb{F}$. For $\mathcal{P}_i \in \overline{I_{\mathcal{P}}}$, let $s_{vid}^{(i)} \overset{\$}{\leftarrow} \mathbb{F}$ s.t. $x = \sum_{i \in [n]} s_{vid}^{(i)}$.
2. Return $(\textsc{Share}, sid, vid, s_{vid}^{(i)})$ to P_i.

Open with identifiable abort: All interfaces below are specific to a $(vid, \cdot)$.
Share: Upon first $(\textsc{Share}, sid, vid)$ by $\mathcal{P}_i \in \mathcal{P}$ and if (vid, x) is stored, sample shares as in **Get Shares** if not previously done and and store locally.
Optimistic Reveal: Upon $(\textsc{Optimist-Open}, sid, vid)$ by each honest $\mathcal{P}_i$ and if **Share** for (vid, x) was run, then send $(\textsc{Output}, sid, vid, x)$ to $\mathcal{S}$. If $\mathcal{S}$ continues, send $(\textsc{Output}, sid, vid, x)$ to each honest $\mathcal{P}_i$, otherwise send $(\textsc{Output}, sid, vid, \perp)$.
Reveal: Upon $(\textsc{Reveal}, sid, vid)$ by $\mathcal{P}_i$, if $i \notin \mathtt{rev}[vid]$ send $(vid, i, s_{vid}^{(i)})$ to $\mathcal{S}$.
1. If $\mathcal{S}$ continues, set $\mathtt{rev}[vid] \leftarrow \mathtt{rev}[vid] \cup \{i\}$, send $(\textsc{Reveal}, sid, i, s_{vid}^{(i)})$ to all $\mathcal{P}$.
2. Else if $\mathcal{S}$ sends $(\textsc{Reveal-Not-Ok}, sid, vid, i, J)$ with $J \subseteq I_{\mathcal{P}}, J \neq \emptyset$, send $(\textsc{Reveal-Fail}, sid, vid, i)$ to all $\mathcal{P}$ and set $\mathtt{ref}[vid] \leftarrow \mathtt{ref}[vid] \cup J$.

Test Reveal: Upon $(\textsc{Test-Reveal}, sid, vid)$ from a party in $\mathcal{P} \cup \mathcal{V}$
1. If $\mathtt{ref}[vid] \neq \emptyset$, return $(\textsc{Reveal-Fail}, sid, vid, \mathtt{ref}[vid])$
2. Otherwise return $(\textsc{Reveal-Fail}, sid, vid, [n] \setminus \mathtt{rev}[vid])$.

Allow Verify: Upon $(\textsc{Start-Verify}, sid, vid, i)$ from party $\mathcal{P}_i \in \mathcal{P}$ set $\mathtt{ver}[vid] \leftarrow \mathtt{ver}[vid] \cup \{i\}$. If $\mathtt{ver}[vid] = [n]$ then deactivate all interfaces for vid except **Test Reveal** and **Verify**.
Verify: Upon $(\textsc{Verify}, sid, vid, z^{(1)}, \ldots, z^{(n)})$ by $\mathcal{V}_i \in \mathcal{V}$ with $z^{(j)} \in \mathbb{F}$:
1. If $\mathtt{ver}[vid] \neq [n]$ then return $(\textsc{Verify-Fail}, sid, vid, [n] \setminus \mathtt{ver}[vid])$.
2. Else if $\mathtt{ver}[vid] = [n]$ and $\mathtt{rev}[vid] \neq [n]$, send to $\mathcal{V}_i$ what **Test Reveal** sends.
3. Else set $\mathtt{ws} \leftarrow \{j \in [n] \mid z^{(j)} \neq s_{vid}^{(j)}\}$ and return $(\textsc{Open-Fail}, sid, vid, \mathtt{ws})$.

[a] Throughout **Init**, **Input**, **Evaluate** and **(Get) Share**, $\mathcal{S}$ can at any point abort, upon which $\mathcal{F}_{\mathsf{Ident:rct}}$ sends $(\textsc{Abort}, \perp)$ to all parties and terminates.

Fig. 6: UC functionality $\mathcal{F}_{\mathsf{Ident}}$ for reactive MPC with Publicly Verifiable Output.

Identifiable aborts during the output phase. We provide an overview of the execution of a generic protocol π in the $\mathcal{F}_{\mathsf{Ident}}$-hybrid setting, where π can either obtain the output of an MPC secure evaluation on private client inputs performed by an $\mathcal{F}_{\mathsf{Ident}}$ instance, or identify cheating parties.

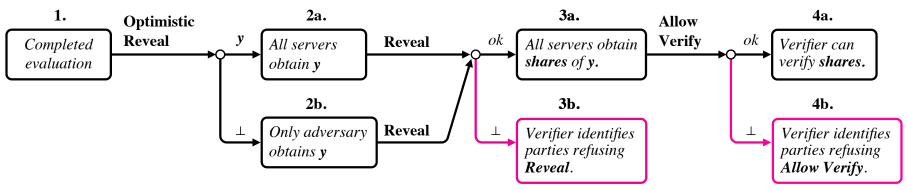

1.
Completed evaluation
Optimistic Reveal
y
⊥
2a.
All servers obtain y
2b.
Only adversary obtains y
Reveal
Reveal
ok
⊥
3a.
All servers obtain shares of y.
3b.
Verifier identifies parties refusing Reveal.
Allow Verify
ok
⊥
4a.
Verifier can verify shares.
4b.
Verifier identifies parties refusing Allow Verify.

Fig. 7: Output phase of $\mathcal{F}_{\mathsf{Ident}}$ with verifiable output.

Following the secure evaluation on the client inputs, honest parties of π perform the following. Upon sending **Optimistic Reveal** to $\mathcal{F}_{\mathsf{ident}}$, honest parties will either (2a) obtain the output $\mathbf{y} = (y_1, ..., y_m)$, where y_i denotes the output for client $C_i \in \mathcal{C}$, or (2b) only the adversary obtains $\mathbf{y}$. In either state (2a)/(2b), the honest parties of π can *always* reach a state of $\mathcal{F}_{\mathsf{Ident}}$ via **Reveal** and **Allow Verify**, in which either (4a) shares $\mathbf{s}^{(i)}$ for each server $P_i \in \mathcal{P}$ are received by all parties, that are verifiable by by $\mathcal{V}$ and from which $\mathbf{y} = \sum_{i \in [n]} \mathbf{s}^{(i)}$ can be reconstructed, or (3b)/(4b) the verifier $\mathcal{V}$ can identify cheating servers.

The interfaces of $\mathcal{F}_{\mathsf{Ident}}$ exposed to the public verifier $\mathcal{V}$ are central to the arbitration of an abort in protocol π. Here, a smart contract playing the role of public verifier $\mathcal{V}$ can identify the set of cheating servers therefore enforce a financial penalty agreed upon prior to the adversary learning $\mathbf{y}$.

C Ledger Functionalities

C.1 Public ledger functionality

In Figure 8 we describe the ideal functionality $\mathcal{F}_{\mathsf{Ledger}}$. It reflects a general public ledger, with the support for transfers of tokens through signatures, along with Turing complete smart contracts, modeled as arithmetic circuits over $\mathbb{F}$. It requires access to the global UC functionalities of $\mathcal{F}_{\mathsf{Clock}}$, for a notion of rounds, and $\mathcal{F}_{\mathsf{(T)Sig}}$ for signature validation.

Beyond its authenticated bulletin board functionality, on which it is based, $\mathcal{F}_{\mathsf{Ledger}}$ parses all the newly received, signed messages and updates its public state accordingly on the first activation of each $\mathcal{F}_{\mathsf{Clock}}$ round. In addition to authenticated messages, we define its public state to include a public ledger over a default token universe, maintaining balances associated with each signature verification key observed in the authenticated message list. Furthermore, $\mathcal{F}_{\mathsf{Ledger}}$ will maintain public state of *smart contracts instances*, each deployed with a transition function encoded as arithmetic circuits.

Interaction with UC functionalities. We permit smart contracts deployed to $\mathcal{F}_{\mathsf{Ledger}}$ to pass messages to external UC functionalities. This is required in order for a smart contract instance to evaluate the verification of proofs generated by a $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ instance or shares output by $\mathcal{F}_{\mathsf{Ident}}$. Interaction in the GUC model is permitted for global functionality $\mathcal{F}_{\mathsf{Ledger}}$ and other global UC functionalities,

Functionality $\mathcal{F}_{\mathsf{Ledger}}$

$\mathcal{F}_{\mathsf{Ledger}}$ interacts with global functionalities $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{TSig}}$. It is parameterized by a token universe $\mathbb{T} = \{\tau_0, ..., \tau_t\}$ and maintains public ledger state $\mathcal{L} : \{0,1\}^* \to \mathbb{Z}^{|\mathbb{T}|}$, public contract states $\mathit{\Gamma} : \mathbb{Z} \to \{0,1\}^*$ and a contract counter $\mathsf{ctr} \in \mathbb{Z}$. $\mathcal{F}_{\mathsf{Ledger}}$ has an initially empty list $\mathcal{M}$ of messages posted to the authenticated bulletin board. Further, it interacts with committee parties $\mathcal{Q} = \{Q_1, ..., Q_q\}$.

Upon each activation $\mathcal{F}_{\mathsf{Ledger}}$ first sends a message $(\textsc{Read}, \mathit{sid})$ to $\mathcal{F}_{\mathsf{Clock}}$. If ν has changed since the last call to $\mathcal{F}_{\mathsf{Clock}}$, then it parses all messages in $\mathcal{M}' \leftarrow \mathcal{M} \setminus \mathcal{M}_{\mathsf{read}}$ in listed order as follows:

Init: Upon parsing $((\textsc{Init}, \mathit{sid}, \mathcal{L}^{\mathsf{init}}), \mathsf{vk}) \in \mathcal{M}'$, set $\mathcal{L} \leftarrow \mathcal{L}^{\mathsf{init}}$, deactivate the parsing of $\textsc{Init}$ messages and activate parsing of $\textsc{Transfer}, \textsc{Deploy}, \textsc{Call}$ messages.

Transfer: Upon parsing $((\textsc{Transfer}, \mathit{sid}, \bar{\mathbf{v}}, \mathsf{vk}_{\mathsf{rcv}}), \mathsf{vk}) \in \mathcal{M}'$, assert $\mathcal{L}[\mathsf{vk}] \geq \bar{\mathbf{v}}$. Set $\mathcal{L}[\mathsf{vk}_{\mathsf{rcv}}] \leftarrow \mathcal{L}[\mathsf{vk}_{\mathsf{rcv}}] + \bar{\mathbf{v}}$ and $\mathcal{L}[\mathsf{vk}] \leftarrow \mathcal{L}[\mathsf{vk}] - \bar{\mathbf{v}}$.

Deploy Contract: Upon parsing $((\textsc{Deploy}, \mathit{sid}, \gamma, T), \mathsf{vk}) \in \mathcal{M}'$, parse initial state $\gamma \in \{0,1\}^*$ and T as an arithmetic circuit over $\mathbb{F}$ of maximum depth d_T. Update $\mathit{\Gamma} \leftarrow \mathit{\Gamma} \cup \{(\mathsf{cn} = \mathsf{ctr}, (\gamma, \bar{\mathbf{0}}, T))\}$ and contract id counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$.

Call Contract: Upon parsing $((\textsc{Call}, \mathit{sid}, (\mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}})), \mathsf{vk}) \in \mathcal{M}'$, assert $\mathcal{L}[\mathsf{vk}] \geq \bar{\mathbf{v}}$. Read $\mathcal{F}_{\mathsf{Clock}}$ round ν, obtain contract state, balances and circuit $(\gamma, \bar{\mathbf{w}}, T) \leftarrow \mathit{\Gamma}[\mathsf{cn}]$ and evaluate circuit T on inputs $(\nu \mid \gamma \mid \bar{\mathbf{w}} \mid \mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}} \mid \mathsf{vk})$, which outputs an encoding of a state transition ts, $\mathcal{L} \mid \gamma \mid \bar{\mathbf{w}} \to^{\mathsf{ts}} \mathcal{L}' \mid \gamma' \mid \bar{\mathbf{w}}'$, updating ledger, contract state and balance.

1. T is permitted "callback" gates which specify:
   a. Fragments of state $\mathcal{L} \mid \mathit{\Gamma}$ to read during evaluation of T.
   b. External functionality $\mathcal{F}$ and message m to send:
      - Forward $(\textsc{ExtCall}, \mathit{sid}, \mathcal{F}, \mathsf{m})$ to all $\mathcal{Q}$ who output this forwarded message.
      - Upon $(\textsc{CallBack}, \mathit{sid}, \mathsf{ret})$ from each $\mathcal{Q}$, write replies to call-back gate; the majority response from $\mathcal{Q}$ is read by the circuit upon continuation.
2. On completed evaluation of T, assert:
   a. Preservation of tokens: $\bar{\mathbf{w}} + \sum_{\mathsf{vk} \in \mathsf{dom}(\mathcal{L})} \mathcal{L}[\mathsf{vk}] = \bar{\mathbf{w}}' + \sum_{\mathsf{vk} \in \mathsf{dom}(\mathcal{L})} \mathcal{L}'[\mathsf{vk}]$
   b. No outflow from non-calling accts: $\forall \mathsf{vk}' \in \mathsf{dom}(\mathcal{L}), \mathsf{vk}' \neq \mathsf{vk} : \mathcal{L}'[\mathsf{vk}'] \geq \mathcal{L}[\mathsf{vk}']$.
3. Set $\mathcal{L} \leftarrow \mathcal{L}'$ and $\mathit{\Gamma}[\mathsf{cn}] \leftarrow (\gamma', \bar{\mathbf{w}}', T)$.

After parsing $\mathcal{M}'$, it sets $\mathcal{M}_{\mathsf{read}} \leftarrow \mathcal{M}$ and sends $(\textsc{Update}, \mathit{sid})$ to $\mathcal{F}_{\mathsf{Clock}}$.

Post: Upon receiving $(\textsc{Post}, \mathit{sid}, m, \mathsf{vk}, \sigma)$ from some entity contact the instance of $\mathcal{F}_{\mathsf{Sig}}$ or $\mathcal{F}_{\mathsf{TSig}}$ belonging to vk. If σ verifies for m and vk then send $(\textsc{Post}, \mathit{sid}, m, \mathsf{vk}, \sigma)$ to $\mathcal{S}$ and append (m, vk) to the list $\mathcal{M}$.

Read: Upon receiving $(\textsc{Read}, \mathit{sid})$ from some entity, return $\mathcal{M}$.

Fig. 8: Functionality $\mathcal{F}_{\mathsf{Ledger}}$ for Public Ledger and Smart Contracts.

such as $\mathcal{F}_{\mathsf{Clock}}$. However, lifting the model of $\mathcal{F}_{\mathsf{Ident}}$ or $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ to global functionalities greatly complicates the definition of any functionality realized in the $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$-hybrid setting, as the simulator can no longer equivocate outputs from $\mathcal{F}_{\mathsf{Ident}}$ without simulating its internal state as a hybrid functionality, and extraction of a global $\mathcal{F}_{\mathsf{NIZK}}^{\mathcal{R}}$ would imply a realization by less efficient constructions.

Although we do not model consensus details with $\mathcal{F}_{\mathsf{Ledger}}$, we argue that such a protocol must ultimately realized in the presence of an honest majority committee. Thus, we adopt this assumption with an honest-majority committee of dummy parties $\mathcal{Q} = \{Q_1, ..., Q_q\}$ interacting with $\mathcal{F}_{\mathsf{Ledger}}$, that *forward* verification calls between $\mathcal{F}_{\mathsf{Ledger}}$ and the environment $\mathcal{Z}$. Concretely, we permit the deployed contract circuits to feature *call-back* gates, which indicate an external functionality and message $(\mathcal{F}, \mathsf{m})$ that is forwarded to $\mathcal{Q}$ by $\mathcal{F}_{\mathsf{Ledger}}$.

- Upon receiving $(\textsc{ExtCall}, \mathit{sid}, \mathcal{F}, \mathsf{m})$ from $\mathcal{F}_{\mathsf{Ledger}}$, an honest party in $\mathcal{Q}$ then returns this message to $\mathcal{Z}$ and waits for a response.
- Upon input $(\textsc{CallBack}, \mathit{sid}, \mathcal{F}, \mathsf{ret})$ from $\mathcal{Z}$ to the same party $Q \in \mathcal{Q}$, it forwards this message to $\mathcal{F}_{\mathsf{Ledger}}$, which writes the majority response to the call-back gate.

The utility of forwarding $(\mathcal{F}, \mathsf{m})$ to the environment via dummy parties $\mathcal{Q}$ becomes immediate in the $\mathcal{F}$, $\mathcal{F}_{\mathsf{Ledger}}$-hybrid setting: here, the parties in the roles of $\mathcal{Q}$, upon receiving $(\textsc{ExtCall}, \mathit{sid}, \mathcal{F}, \mathsf{m})$ from $\mathcal{F}_{\mathsf{Ledger}}$ will call hybrid functionality $\mathcal{F}$ with message m, and return the response to $\mathcal{F}_{\mathsf{Ledger}}$. If $\mathcal{Q}$ maintains an honest majority, we obtain correctness of the verification replies returned to $\mathcal{F}_{\mathsf{Ledger}}$. We emphasize that this is a necessary modelling artifact arising from the constraints of the GUC-framework: in actual realizations we argue the parties in $\mathcal{Q}$ are the same parties which jointly realize the underlying ledger functionality as mining or staking parties.

C.2 Confidential ledger functionality

In Fig. 9 we describe the confidential token ledger functionality, $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ we require in our main construction. $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. It assume access to the $\mathcal{F}_{\mathsf{Ledger}}$ functionality in Fig. 8.

D Protocols

We detail the various protocol realizations of our scheme and their supporting smart contract programs.

D.1 Protocol realizing $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

In Fig. 10 we show how to realize our confidential token functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ from Fig. 9 on any Turing complete ledger, with the help of the smart contract $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ of Fig. 11.

Theorem 1. *Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ GUC-realizes functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$-hybrid model against any PPT-adversary corrupting any minority of committee $\mathcal{Q}$.*

Functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

$\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ interacts with parties $\mathcal{C} = \{C_1, ..., C_p\}$. It is parameterized with token universe $\mathbb{T}$ and max. balance $\bar{\mathbf{v}}_{\mathsf{max}} \in \mathbb{Z}^{|\mathbb{T}|}$ and maintains public ledgers $\mathcal{L}$ and $\mathcal{L}_{\mathsf{Conf}}$, where $\mathcal{L}_{\mathsf{Conf}}$ maps account keys to confidential coins with hidden balances. The functionality is registered at global $\mathcal{F}_{\mathsf{Clock}}$.

Public ledger states are updated at the beginning of each clock round. On each activation, $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ reads $\mathcal{F}_{\mathsf{Clock}}$ and if ν is increased since the last activation, sets $\mathcal{L} \leftarrow \mathcal{L}'$, $\mathcal{L}_{\mathsf{Conf}} \leftarrow \mathcal{L}'_{\mathsf{Conf}}$. Initially, only *GenAcct* and *Init* interfaces are activated.

GenAcct: Upon $(\textsc{GenAcct}, sid)$ from C, forward $(\textsc{GenAcct}, sid, C)$ to $\mathcal{S}$. Upon obtaining fresh vk from $\mathcal{S}$, set $\mathcal{K}[C] \leftarrow \mathcal{K}[C] \cup \{\mathsf{vk}\}$. Return $(\textsc{AcctKey}, sid, \mathsf{vk})$.

Init: Upon receiving $(\textsc{InitLedger}, sid, \mathcal{L}_{\mathsf{Init}}, \mathsf{vk})$ from any $C_i \in \mathcal{C}$, forward to $\mathcal{S}$. Set $\mathcal{L}' \leftarrow \mathcal{L}_{\mathsf{Init}}$ and deactivate *Init* and activate all other interfaces.

Transfer: Upon $(\textsc{Transfer}, \bar{\mathbf{v}}, \mathsf{vk}_{\mathsf{rcv}}, \mathsf{vk})$ from C,
- Assert $\mathsf{vk} \in \mathcal{K}[C]$ and forward message $(\textsc{Transfer}, \bar{\mathbf{v}}, \mathsf{vk}_{\mathsf{rcv}}, \mathsf{vk})$ to $\mathcal{S}$.
- If $\mathcal{L}'[\mathsf{vk}] \geq \bar{\mathbf{v}}$, set $\mathcal{L}'[\mathsf{vk}_{\mathsf{rcv}}] \leftarrow \mathcal{L}'[\mathsf{vk}_{\mathsf{rcv}}] + \bar{\mathbf{v}}$ and $\mathcal{L}'[\mathsf{vk}] \leftarrow \mathcal{L}'[\mathsf{vk}] - \bar{\mathbf{v}}$.

Mint: Upon $(\textsc{Mint}, sid, \bar{\mathbf{v}}, \mathsf{vk})$ from C,
1. Assert $\mathcal{L}'[\mathsf{vk}] \geq \bar{\mathbf{v}}$ and forward $(\textsc{Mint}, sid, \bar{\mathbf{v}}, \mathsf{vk})$ to $\mathcal{S}$ and wait for id from $\mathcal{S}$.
2. Set $\mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}] \cup \{\langle \mathsf{id}, \bar{\mathbf{v}} \rangle\}$ & $\mathcal{L}'[\mathsf{vk}] \leftarrow \mathcal{L}'[\mathsf{vk}] - \bar{\mathbf{v}}$, return $(\textsc{Minted}, sid, \mathsf{id})$.

Confidential Transfer: Upon $(\textsc{ConfTfr}, sid, \mathsf{vk}_{\mathsf{rcv}}, \mathsf{id}_1, \mathsf{id}_2, \bar{\mathbf{v}}'_1, \bar{\mathbf{v}}'_2)$ from C,
1. Assert $\exists \mathsf{vk} \in \mathcal{K}[C] : \langle \mathsf{id}_i, \bar{\mathbf{v}}_i \rangle \in \mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}]$ for $i \in \{1, 2\}$.
2. Assert $\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 = \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2$ and $\bar{\mathbf{v}}_i \leq \bar{\mathbf{v}}_{\mathsf{max}}$ for $i \in \{1, 2\}$.
3. Forward $(\textsc{ConfTfr}, sid, \mathsf{vk}_{\mathsf{rcv}}, \mathsf{id}_1, \mathsf{id}_2)$ to $\mathcal{S}$, and wait for $(\mathsf{id}'_1, \mathsf{id}'_2)$.
4. For $i \in \{1, 2\}$:
   - Set $\mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}] \setminus \{\langle \mathsf{id}_1, \bar{\mathbf{v}}_1 \rangle, \langle \mathsf{id}_2, \bar{\mathbf{v}}_2 \rangle\} \cup \{\langle \mathsf{id}'_2, \bar{\mathbf{v}}'_2 \rangle\}$
   - Set $\mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}_{\mathsf{rcv}}] \leftarrow \mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}_{\mathsf{rcv}}] \cup \{\langle \mathsf{id}'_1, \bar{\mathbf{v}}'_1 \rangle\}$.
5. If $\mathsf{vk}_{\mathsf{rcv}} \in \mathcal{K}[C']$ s.t. $C' \in \mathcal{I}$, send $(\langle \mathsf{id}'_1, \bar{\mathbf{v}}'_1 \rangle)$ to $\mathcal{S}$. Return $(\textsc{Change}, sid, \mathsf{id}'_2)$.

Confidential Receive: Upon $(\textsc{ConfRcv}, sid)$ from C,
- Return $(\textsc{Received}, sid, (\mathsf{vk}_1, \langle \mathsf{id}_1, \bar{\mathbf{v}}_1 \rangle), ..., (\mathsf{vk}_l, \langle \mathsf{id}_l, \bar{\mathbf{v}}_l \rangle))$, containing coins sent to C since the last call, where $\mathsf{vk}_i \in \mathcal{K}[C]$ for $i \in [l]$

Redeem: Upon $(\textsc{ConfRdm}, sid, \mathsf{id})$ from C,
1. Assert $\exists \mathsf{vk} \in \mathcal{K}[C] : \langle \mathsf{id}, \bar{\mathbf{v}} \rangle \in \mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}]$.
2. Remove $\langle \mathsf{id}, \bar{\mathbf{v}} \rangle$ from $\mathcal{L}'_{\mathsf{Conf}}[\mathsf{vk}]$ and set $\mathcal{L}'[\mathsf{vk}] \leftarrow \mathcal{L}'[\mathsf{vk}] + \bar{\mathbf{v}}$.
3. Send $(\textsc{ConfRdm}, sid, \langle \mathsf{id}, \bar{\mathbf{v}} \rangle, \mathsf{vk})$ to $\mathcal{S}$.

GetLedger: Upon $(\textsc{GetLedger}, sid)$, compute sanitized $\mathcal{L}''_{\mathsf{Conf}}$ such that coin balances are removed from $\mathcal{L}_{\mathsf{Conf}}$. Return $\mathcal{L}$, $\mathcal{L}''_{\mathsf{Conf}}$.

Fig. 9: Functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ for Confidential Ledgers.

Proof (Proof of Theorem 1). We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{A}$, hybrid functionalities $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$ and global functionalities $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Sig}}$ such that $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}} \circ \mathcal{S} \approx \Pi_{\underline{\mathsf{C}}\mathsf{Ledger}} \circ \mathcal{A}$ for any PPT environment $\mathcal{Z}$.

Concretely, to create an interaction indistinguishable from a protocol transcript in the composed setting, we construct a simulator $\mathcal{S}$ that generates valid messages for global $\mathcal{F}_{\mathsf{Ledger}}$ from simulated honest client activations and extracts inputs from dishonest messages and forwards these to ideal functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. This ensures consistency of $\mathcal{A}$'s view of $\mathcal{F}_{\mathsf{Ledger}}$ with the state of $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ during the simulated protocol execution.

On an honest GenAcct input, $\mathcal{S}$ generates a fresh signature verification key for the honest client from $\mathcal{F}_{\mathsf{Sig}}$, which it stores. For any subsequent honest input to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ which is forwarded to $\mathcal{S}$, the simulator can generate and post verifying messages global functionality $\mathcal{F}_{\mathsf{Ledger}}$.

For honest InitLedger, Transfer inputs, generating verifying messages to post on $\mathcal{F}_{\mathsf{Ledger}}$ is trivial for $\mathcal{S}$, as it generates and stores signature verification keys for honest clients. On observing dishonest InitLedger, Transfer messages on $\mathcal{F}_{\mathsf{Ledger}}$ and asserting that they are accepted by $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, the simulator can extract all dishonest inputs to forward to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, as these messages are posted to $\mathcal{F}_{\mathsf{Ledger}}$ in cleartext.

On an honest Mint input, $\mathcal{S}$ must generate and send a verifying Call message to $\mathcal{F}_{\mathsf{Ledger}}$ with the minted amount $\bar{\mathbf{v}}$ which activates the deployed $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ contract instance to mint a fresh confidential token. Since $\mathcal{S}$ simulates protocol messages from honest clients, it can generate a valid commitment for $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ itself and store its opening $(\bar{\mathbf{v}}, r)$. With the commitment opening, it obtains a verifying NIZK via $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ proving $\mathcal{R}(\bar{\mathbf{v}}, c; r) = \{c = \mathbf{g}^{\bar{\mathbf{v}}} h^r\}$. For a dishonest mint message observed on $\mathcal{F}_{\mathsf{Ledger}}$ by $\mathcal{S}$, the simulator trivially extracts inputs for $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$: both minted amount and minting account key in the Call message sent to activate minting in the $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ contract instance are observable in cleartext on $\mathcal{F}_{\mathsf{Ledger}}$.

On an honest ConfTransfer input, $\mathcal{S}$ generates valid coin commitments and rangeproofs for a call activating ConfTrfr on the $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ contract instance deployed to $\mathcal{F}_{\mathsf{Ledger}}$. For an *honest sender and honest recipient*, $\mathcal{S}$ needs to generate output coin commitments that are consistent with the chosen input coins for the simulated protocol. Here, $\mathcal{S}$ always possesses the openings of the input coin commitments:

- *Coins previously received from an honest sender* were generated by $\mathcal{S}$ with arbitrary openings previously generated by $\mathcal{S}$: since $\mathcal{S}$ does not learn the transfer amount for confidential transfer between honest users, it generates output coins commitments with *arbitrary balances*, such that the product equality of input and output commitments holds: $\mathbf{g}^{\bar{\mathbf{v}}_1} h^{r_1} \mathbf{g}^{\bar{\mathbf{v}}_2} h^{r_2} = \mathbf{g}^{\bar{\mathbf{v}}'_1} h^{r'_1} \mathbf{g}^{\bar{\mathbf{v}}'_2} h^{r'_2}$. However, since simulated setup functionality $\mathcal{F}_{\mathsf{Setup}}$ samples $s \leftarrow\$\ \mathbb{F}_p$ and outputs $h = g^s$, coins generated by $\mathcal{S}$ can later be *equivocated* to any value.
- *Coins previously received from a dishonest sender* feature openings sent directly to the receiving honest client simulated by $\mathcal{S}$ in the simulated protocol.

Thus, for an honest confidential transfer sending coins to another honest party, $\mathcal{S}$ generates output coin commitments with arbitrary chosen coin balances, stores their openings and obtains verifying rangeproofs via $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$. For an *honest sender and dishonest recipient*, $\mathcal{S}$ learns the transferred amount from $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$, and can generate output coin commitments with correct balances and post these to $\mathcal{F}_{\mathsf{Ledger}}$ (with equivocation of the input coin commitments if necessary). Then, it forwards the coin openings as a simulated protocol message to the dishonest recipient.

Finally for a *dishonest sender and dishonest recipient* $\mathcal{S}$ can extract openings for all coins generated by the dishonest sender since they all have associated NIZK's obtained by sending valid coin openings to the simulated $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ instance. Thus, the simulator can forward the transferred amounts to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. For a *dishonest sender and honest recipient*, the simulated honest recipient obtains transferred coin commitment opening as a protocol message, allowing $\mathcal{S}$ to forward this input to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

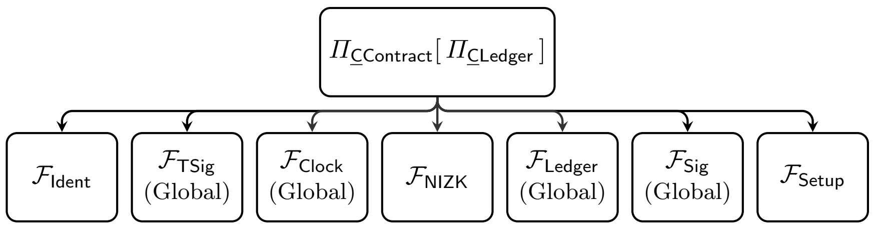
On an honest ConfReceive, the simulator must have previously provided inputs to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ for confidential transfers *initiated* by dishonest parties, as previously described. On a *dishonest confidential receive*, $\mathcal{S}$ will have previously sent the openings of the honestly sent coins to the dishonest recipient as a protocol message, in addition to having generated valid coins and rangeproofs observable on $\mathcal{F}_{\mathsf{Ledger}}$.

On an honest ConfRedeem, if the redeemed coin was originally sent by a dishonest user, $\mathcal{S}$ must have also received its opening as a protocol message , as it simulates the role of the honest user in the protocol execution. Otherwise, the redeemed coin must have been sent by an honest user, and can thus be equivocated by $\mathcal{S}$. Thus, with the equivocated coin opening, $\mathcal{S}$ can produce a verifying NIZK for the honest redeem action in the simulated protocol view. On a *dishonest redeem*, $\mathcal{S}$ observes the redeemed value publicly on $\mathcal{F}_{\mathsf{Ledger}}$, and can thus forward this input to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

As long as the majority of parties in $\mathcal{Q}$ are honest, verification responses from $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ are interpreted correctly by the call-back gate on $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. Thus, the public state of $\mathcal{L}_{\mathsf{Conf}}$ on $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ observed in the simulated protocol view is consistent with the confidential ledger maintained by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

Finally, we note that the *updates* to ledger states induced by client activations are applied at the beginning of each $\mathcal{F}_{\mathsf{Clock}}$ round in both global $\mathcal{F}_{\mathsf{Ledger}}$ and ideal functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. □

D.2 Protocol realizing $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$



$\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}[\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}]$
$\mathcal{F}_{\mathsf{Ident}}$
$\mathcal{F}_{\mathsf{TSig}}$ (Global)
$\mathcal{F}_{\mathsf{Clock}}$ (Global)
$\mathcal{F}_{\mathsf{NIZK}}$
$\mathcal{F}_{\mathsf{Ledger}}$ (Global)
$\mathcal{F}_{\mathsf{Sig}}$ (Global)
$\mathcal{F}_{\mathsf{Setup}}$

In Fig. 12 and 13 we show how to realize our privacy preserving smart contract functionality $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ from Fig. 2 on any Turing complete ledger, with the help of a smart contract with code of $\mathcal{X}_{\mathsf{Lock}}$ of Fig. 14 to manage confidential tokens and $\mathcal{X}_{\mathsf{Collateral}}$ of Fig. 15 to manage underlying collateral. Note that $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ extends $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$, and similarly that contract $\mathcal{X}_{\mathsf{Lock}}$ extends $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$.

Theorem 2. *$\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}[\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ realizes $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}[\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ in the $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Setup}}$, $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{TSig}}$-hybrid model against any PPT-adversary corrupting at most $n-1$ of the n servers $\mathcal{P}$ statically and any minority of $\mathcal{Q}$.*

Proof. (Theorem 2) We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{A}$, hybrid functionalities $\mathcal{F}_{\mathsf{Ident}}$, $\mathcal{F}_{\mathsf{NIZK}}$, and global functionalities $\mathcal{F}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{Sig}}$, $\mathcal{F}_{\mathsf{TSig}}$ such that $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}} \circ \mathcal{S} \approx \Pi_{\underline{\mathsf{C}}\mathsf{Ledger}} \circ \mathcal{A}$ for any PPT environment $\mathcal{Z}$.

Upon an honest Init, the simulator $\mathcal{S}$ simulates the roles of the honest parties in the simulated protocol execution, and jointly generates a threshold signature verification key with the dishonest parties via $\mathcal{F}_{\mathsf{TSig}}$. It simulates **GenAcct** and **InitLedger** as in $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. As $\mathcal{S}$ generates the signature verification key for each honest server, it can call $\mathcal{F}_{\mathsf{Sig}}$ and generate verifying messages for the simulated honest server to post on global $\mathcal{F}_{\mathsf{Ledger}}$, observable by $\mathcal{A}$. Since $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ extends $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$, $\mathcal{S}$ signs messages that initialize contracts $\mathcal{X}_{\mathsf{Lock}}[\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ and $\mathcal{X}_{\mathsf{Collateral}}$, and can authorize collateral deposits to the contract instance $\mathcal{X}_{\mathsf{Collateral}}$ on $\mathcal{F}_{\mathsf{Ledger}}$.

Upon an honest Enroll, the simulator is forwarded the input client coin identifier and account verification key. $\mathcal{S}$ simulates the honest party by generating output masks for which it samples the random openings. Then it determines valid openings for the honest input coin:

- If the honest input coin was previously transferred by a dishonest party, simulator $\mathcal{S}$ can extract its opening from the NIZK range-proof generated via simulated hybrid $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$.
- If the honest input coin was previously transferred by an honest party, $\mathcal{S}$ must have generated the openings itself (See simulator of $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$).

In either case, the simulator sends valid openings of both honest input coins and mask commitments to hybrid $\mathcal{F}_{\mathsf{Ident}}$.

Subsequently, the simulator can simulate a consistent protocol execution of **Verify input** which only aborts if $\mathcal{A}$ provides inputs to simulated hybrid $\mathcal{F}_{\mathsf{Ident}}$ that are inconsistent with the input coin and mask commitments sent to $\mathcal{X}_{\mathsf{Lock}}$ on simulated $\mathcal{F}_{\mathsf{LedgerVM}}$. It simulates the batched sigma protocol to check input consistency in the simulated execution of **Verify input** in $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$. Inconsistency of inputs must arise from cheating by $\mathcal{A}$ and results in an abort. As shown in Section 3.1, the probability that the simulated protocol <u>aborts</u> due to inconsistent inputs whilst the ideal functionality <u>continues</u> is negligible in the group order of the Pedersen commitment scheme.

Upon an honest Evaluate and its successful completion, the simulator will jointly sign eval via $\mathcal{F}_{\mathsf{TSig}}$ with the dishonest parties.

Upon an honest Open, the simulator first observes what $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ outputs, and then will modify the state of the simulated $\mathcal{F}_{\mathsf{Ident}}$ instance, such that the adversary in the simulated protocol observes the masked outputs consistent with what $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ outputs.

Upon an honest Withdraw, the simulator does nothing. At each $\mathcal{F}_{\mathsf{Clock}}$ round during the simulated protocol execution, if the adversary aborts, $\mathcal{S}$ will forward an abort to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$. If a dishonest party cheats during the **Open** phase of the simulated protocol execution it will be identified by the simulated $\mathcal{F}_{\mathsf{Ident}}$ instance, and its identity is forwarded to $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ by $\mathcal{S}$.

$\mathcal{S}$ simulates the honest parties of committee $\mathcal{Q}$ in the simulated protocol execution. As long as the majority of parties in $\mathcal{Q}$ are honest, verification responses from $\mathcal{F}_{\mathsf{Ident}}$ are interpreted correctly by the call-back gate on $\mathcal{X}_{\mathsf{Lock}}$, permitting the cheating parties in the simulated protocol execution to be correctly identified during an abort. □

E Confidential contract extensions

Multi-round confidential contracts. We now demonstrate how our default model of confidential contracts, shown in previous sections, can be extended to a multi-round model, where clients can provide fresh inputs and obtain continuous outputs in a long-running confidential blockchain application.

This is facilitated by our model of confidential contracts, which does not require server-client interaction beyond the *Open* phase, along with the reactive interface of our MPC functionality (Appendix B.1), which permits the selective opening of secrets and indefinite number of circuit evaluations on stored secret values. This allows the set of MPC servers to keep an internal secret shared state, off-chain. Furthermore, since we use *outsourced* MPC [37] and rely on a reactive MPC scheme, any multi-round computation can simply be considered a single *reactive* computation, with interleaved input and output. In fact, clients at most hold a state dependent on their own input and expected outputs through out execution of MPC. Thus whatever confidential state is needed, the MPC servers will simply store this secretly, and collectively, throughout the multiple computations of the private smart contracts with different clients giving input and receiving output.

Consider one confidential contract execution and let, without loss of generality, the confidential state of a client be a tuple consisting of a numerical value and balance: $[\, s_j \,] = ([\, y_i \,], [\, \bar{\mathbf{w}}_i \,])$. Further, let the *confidential contract state* be defined over all confidential client states $\mathsf{st} = ([s_1], ..., [s_m])$, which is stored from the previous contract evaluation round or given as the initial confidential contract state. We define a *confidential contract state transition* that consumes a fresh set of confidential contract inputs $\mathsf{st}^{\mathsf{in}} = ([s_1^{\mathsf{in}}], ..., [s_m^{\mathsf{in}}]) = (([\, x_1^{\mathsf{in}} \,], [\, \bar{\mathbf{v}}_1^{\mathsf{in}} \,]), ..., ([\, x_m^{\mathsf{in}} \,], [\, \bar{\mathbf{v}}_m^{\mathsf{in}} \,]))$, such that the the *current contract circuit* $\mathbf{g}$ is evaluated over both st and $\mathsf{st}^{\mathsf{in}}$ to obtain a new confidential state st' and an encoding of the updated circuit $\mathbf{g}'$ to be evaluated in the next round.

$$\big([\, \mathbf{g}' \,], [\, s_1' \,], ..., [\, s_m' \,]\big) \leftarrow \mathsf{eval}_{\mathbf{g}}\big([s_1^{\mathsf{in}}], ..., [s_m^{\mathsf{in}}], [\, s_1 \,], ..., [\, s_m \,]\big)$$

Upon successful completion of a round evaluation, circuit $\mathbf{g}'$ will be securely opened, stored and evaluated by the servers in the following round. Each client can retrieve its new state by executing *Withdraw*.

However, the set of clients wishing to give input to a confidential contract evaluation might not always be the same. Thus we now argue a simple extension to our $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$ model to permit clients can selectively participate in the **Enroll** phase of a round, or to *skip* a given round by ticking the $\mathcal{F}_{\mathsf{Clock}}$ after calling a **Skip Round** interface.

We propose an output budget for each client corresponding to the number of *unused*, pre-processed output masks: in each round, a client will receive a masked output which can be retrieved from the contract $\mathcal{X}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$, regardless whether it provides a new input and participates in **Enroll**: masked outputs for a specific client are generated in each round until its pre-processed output masks have been consumed. The evaluation of the confidential contract in each round is still evaluated over *all clients* and their secret state $\mathsf{st} = ([s_1], ..., [s_m])$, even if only a subset have provided fresh inputs for a round. A client **Skip** implies evaluating the contract circuit over default input values.

Each participation in an **Enroll** phase of a round permits a client to *restore* its depleted output budget, by generating masks in commitment form and inputting their openings to the MPC instance, which are subsequently verified for consistency in the **Verify Input** protocol phase. Each output mask can be associated with a *fee* paid to the servers executing the MPC: once all output budgets (and associated output masks) are consumed, the multi-round confidential contract can terminate safely.

We observe, that this approach also implies that the expensive **Init** phase only needs to be carried out once, assuming the set of MPC servers don't change and no server actively cheats (i.e. causes the execution of the *abort* phase to an extend where a malicious server is identified at the smart contract level). Thus, servers only need to setup threshold keys, smart contracts and collateral once, but naturally need to reevaluate this setup in case a server is confirmed to cheat and penalized.

Mitigation of token minting. Under full server corruption, it is possible for the adversary to mint confidential balances beyond the supply of underlying tokens wrapped by $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$. This is because in our default protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ shown in Figure 12, any output coin distribution accompanied by a verifying threshold signature (via $\mathcal{F}_{\mathsf{TSig}}$) will be accepted by $\mathcal{X}_{\mathsf{Lock}}$; no coin sum-checks or range-proofs enforce the preservation of confidential token supplies (See Equation 1). This is not publicly detectable, even if $\mathcal{X}_{\mathsf{Lock}}$ implemented a product consistency check over input and output coins with correlated commitment randomness, $\prod_{j\in[m]} c_j = \prod_{j\in[m]} c_j^{\mathsf{out}}$, as hidden balances can exceed $\bar{\mathbf{v}}_{\mathsf{max}} = 2^l - 1$, and p additional units of each token type can be minted.

There are several ways in which this can be mitigated. If we accept client interaction during the output phase, then the client can simply retrieve their output commitments as part of the protocol and subsequently generate range-proofs and a zero-sum proof over input and output coins and post this to the smart contract $\mathcal{X}_{\mathsf{Lock}}$. The smart contract can then validate these proofs and thus ensure that no tokens have been minted or destroyed as part of the private smart contract execution. Unfortunately, this also allows a single malicious client to abort the execution *and* goes against our goal of minimizing client interaction. Instead we suggest an approach based on bit-decomposition of token amounts, along with the masks. Based on the decomposition, the zero-sum property of input and output coin commitments (Equation 1) is ensured by a proof to $\mathcal{X}_{\mathsf{Lock}}$, constructed by the servers *without* the need for computing cryptographic primitives inside the MPC circuit.

More concretely, in the following we show an extension of $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ that prevents the minting of tokens under full server corruption, at the cost of a constant-factor increase in communication complexity. This is based on the bit decomposition approach as in Banerjee *et al.* [4], but greatly improves on the protocol efficiency by *not* requiring any NIZK's and commitments to be generated inside the MPC circuit.

Let l be the number of bits such that each coin balance does not exceed $\bar{v}_{\mathsf{max}} \leq 2^l - 1$. In the **Enroll** phase, clients each generate bit a commitment pair $(c_0 = \mathsf{com}(b_0, s_0), c_1 = \mathsf{com}(b_1, s_1))$ for each bit position $k \in [l]$, such that $b_i = 0 \wedge b_{i-1} = 1$ for random $i \leftarrow\$ \{0, 1\}$. Let π denote the bit permutation sampled by the client for the bit position $k \in [l]$, such that:

$$\pi(k) = \begin{cases} 0 & b_{k,0} = 0 \wedge b_{k,1} = 1 \\ 1 & b_{k,0} = 1 \wedge b_{k,1} = 0 \end{cases}$$

This permutation on the individual bits is later used to mask the bit-decomposed output. Commitment pairs are posted to $\mathcal{F}_{\mathsf{Ledger}}$, together with an efficient sigma proof that commitments are to bit values [35], incurring an additional communication complexity logarithmic in the size of the commitment group order.

During the **Enroll** phase, users input the opening to these bit commitment pair in the permuted order:

$$([\,\bar{v}\,], [\,\bar{r}\,], \{([\,b_{0,k}\,], [\,s_{0,k}\,], [\,b_{1,k}\,], [\,s_{1,k}\,])\}_{k \in [l]})$$

where $c_{\mathsf{in}} = (\bar{v}, \bar{r})$ is the opening to the confidential input coin, and each tuple $(b_{0,k}, s_{0,k}, b_{1,k}, s_{1,k})$ is the opening to the k'th bit commitment pair with permuted bit ordering. We adopt a well-formedness check on bits input to $\mathcal{F}_{\mathsf{Ident}}$ from [33]: servers assert for each $k \in [l]$ bit position, that one of $[\,b_{k,0}\,], [\,b_{k,1}\,]$ holds the value 1 and the other holds the value 0. Concretely, for bit pair $([\,b_0\,], [\,b_1\,])$, servers jointly sample and open $\alpha, \beta, \gamma, \leftarrow\$ \mathbb{F}$, and compute:

$$[\,t\,] = \alpha \cdot ([\,b_0\,] \cdot [\,b_0\,] - [\,b_0\,]) + \beta \cdot ([\,b_1\,] \cdot [\,b_1\,] - [\,b_1\,]) + \gamma \cdot ([\,b_0\,] \cdot [\,b_1\,])$$

$$[\,t'\,] = ([\,b_0\,] + [\,b_1\,])$$

Upon securely opening t and t', servers assert that $t = 0 \wedge t' = 1$. Consistency between all commitments and their openings input to $\mathcal{F}_{\mathsf{Ident}}$ are verified during **Verify input** phase by the servers.

Importantly, the financial output of a client is output in bit-decomposed form, where individual bits are permuted in the ordering as chosen by the clients.

$$(b'_1, ..., b'_l)$$

Let $(b_1, ..., b_l)$ denote the true bit-decomposition of a clients output balance. Then $b'_k = b_k$ if $\pi(k) = 0$ and is bit permuted otherwise, where π denotes the permutation chosen by the client in the enroll phase.

For contract $\mathcal{X}_{\mathsf{Lock}}$ to generate the client output coin from bit commitments submitted during **Enroll**, it computes.

$$c_{\mathsf{out}} = \prod_{k \in [l]} c_{k,i}^{2^k} \text{ where } i = b'_k \in \{0, 1\}$$

Here, note that b'_k is interpreted as selector for bit commitment pair $(c_{k,0}, c_{k,1})$ for bit position k. As both $b'_k = i$ and the bit message of $c_{k,i}$ are permuted by $\pi(k)$, the hidden balance of c_{out} is unmasked. Given the generation of output coins from l bit balance representations, confidential output balances are bounded by $2^l - 1 = \bar{v}_{\mathsf{max}}$.

It remains to prove consistency between input and output commitments to $\mathcal{X}_{\mathsf{Lock}}$ to ensure no token minting occurred. For this, servers compute the commitment randomness for each client output coin and the difference in commitment randomness between the input and output coins.

$$[\, s_{\mathsf{out}} \,] = \sum_{k \in [l]} 2^k \cdot [\, s_{k,i} \,] \text{ where } i = b'_k \in \{0, 1\} \qquad [\, \bar{r}_{\mathsf{diff}} \,] = \sum_{j \in [m]} [\, \bar{r}_{\mathsf{in},j} \,] - [\, s_{\mathsf{out},j} \,]$$

Servers locally compute $h^{\bar{r}^{(i)}_{\mathsf{diff}}}$ over the local share value of $[\, \bar{r}_{\mathsf{diff}} \,]$ and send it to all other servers. Each server then reconstructs $h^{\bar{r}_{\mathsf{diff}}}$ and verifies that sum of confidential input and output balances must be equal and that no tokens are minted (balance over-flow is mitigated by bounding output balances by $2^l - 1$).

$$\prod_{j \in [m]} c_{\mathsf{in},j} = h^{\bar{r}_{\mathsf{diff}}} \cdot \prod_{j \in [m]} c_{\mathsf{out},j}$$

We outline the overhead of this approach to prevent malicious minting when

Table 4: Complexity of the per user overhead by using the stand-alone token minting mitigation approach.

	Exponentiation	MPC mult.
User	$6 \cdot l$	0
Server	$8 \cdot l + 1$	$42.5 \cdot l + 15$
Comm. #$\mathbb{G}$elem.	$O(n \cdot l)$	$O(n^2 \cdot l)$

all servers are corrupted in Tab. 4, when assuming that Schnorr proofs are added for each commitment to allow extraction (though not in UC) and when using the work of Reistad and Toft [56] to do the needed bit decomposition in MPC.

F Applications

In this section we briefly outline some interesting application which privacy preserving smart contracts can help facilitate, along with our scheme can be used to provide privacy preserving side-chains and how it can be extended to allow for privacy preserving cross-chain smart contracts.

F.1 Privacy preserving applications

Several general applications for privacy preserving smart contracts have already been suggested in previous works. We briefly outline some of these here.

Auctions Auctions of digital goods, or digital deeds linked to physical goods, can be constructed simpler and more efficiently than with non-privacy preserving smart contracts. Our solution could be used to implement first and second price auctions securely and privately. Concretely confidential tokens reflecting the maximum bid each user should be transferred to a privacy preserving smart contract along with the good for sale. The smart contract then compares the bids and transfers ownership of the good and handles the payment and refunding, according to code of the smart contract being executed in MPC.

Identity management Decentralized Identity (DID) management is the idea that, by using blockchains, users remain in charge over how their private attributes (certified by an appropriate authority) are used online. Multiple schemes for this has been suggested such as Sovrin [42] or CanDID [49]. However, these schemes generally only consider leveraging the blockchain for storing user's attribute information. However, using privacy preserving smart contracts would allow integration of user-certified attributes in both the web 2 and web 3 space. Concretely the users could give their hidden certified attributes as input the privacy preserving smart contract, which can validate them privately and use the content of these attributes to affect its business logic. For example the attributes can be used to decide the price of an NFT or to validate whether a user is privilege enough to execute certain commands of the contract.
Mixer Our structure can naturally be extended to allow for a mixing functionality. While several other technologies exist for this, we observe that doing this in MPC allows several advantages that can prevent the mixing to be used for money laundering. Concretely we could imagine that KYC (Know Your Customer) information linked to the users' blockchain address must be given and privately validated against deny-lists, to prevent criminals using this service. Even if deny-lists are not in use, linking to an actual identity could also be leveraged to allow a given user to only get privacy on the first x amount of tokens they mix, and after that, information on the token amount will become public.

F.2 Anonymous side-chain

Our solution could also be used to construct privacy preserving side-chains. When no server is trying to cheat, there is technically no need for the MPC servers to post anything related to the specific clients and their input to the blockchain, after the *evaluation* phase. Thus the MPC servers can alone realize a privacy preserving side-chain where they in MPC hold the opening information to the commitments of hidden tokens. Thus users can request transfers to other users in this side-chain, if the servers just use the MPC scheme to keep track of how many tokens each user has. At certain intervals, each user can then just decide to get paid back whatever they hold in the side-chain, by the execution of the *open* and *withdraw* phases. This can be used to enhance the anonymity of hidden transfers, since now *only* the MPC servers know the transaction graph, and yet they do not know the transaction amounts. An interesting observation with this case is also that the side-chain will be faster and cheaper to use than the underlying layer 1 blockchain, since it will only be managed by the MPC servers.

F.3 Cross-chain Exchange

In section we will discuss how to use the ideas of P2DEX [9] to make our scheme capable of doing confidential computation and transactions *across* multiple layer 1 blockchains.

Decentralized exchanges. When it comes to decentralized exchange, multiple approaches exist but generally fall into one of the following families:

P2P Two parties, each with tokens on a chain the other decide, agree on doing an exchange with a certain exchange rate. This is for example the approach used in hash-proofs [58]. This unfortunately requires multiple rounds of on-chain interaction, fees, not to mention the issue of having parties find each other.
Exchange chain A chain contains wrapped tokens pegged to their native counterparts through holding smart contracts on all the native chains. This allows to reduce the cross-chain exchange problem to an on-chain problem, assuming the problem of inter-chain communication has been solved. With an exchange chain in place there are multiple ways of facilitating exchanges, since now the problem is reduced same-chain exchange: **Order book**: In the order book approach all orders (e.g. limit orders) are written to the chain and then matched and carried out by a smart contract. Unfortunately this inherently front-running by miners. **AMM**: An AMM is basically a liquidity holding smart contract, allowing exchange between two different tokens. The smart contract then facilitates exchange between tokens of type A and B, with an exchange rate that ensure that the product of the amount of tokens in the contract, remains constant. Unfortunately AMMs are highly susceptible to front-running by miners, since orders and exchange rates will be known to miners before they get carried out.

While these approaches solve some issues related to decentralized exchange none of these are unfortunately a silver bullet for users who desire both ease of use, decentralization and front-running resistance [6].

P2DEX. P2DEX [9] is a different system for achieving cross chain exchange, although it can be considered a special case of the order book approach. It uses a set of outsourced MPC servers [28,37] who threshold control burner addresses, where the clients transfer the tokens they wish to exchange, to compute order matching based on private input of clients. The servers then use the threshold keys for these addresses to send money out of these burner addresses to the intended recipients.

Adding cross-chain functionality. Like our work, P2DEX also use a set of MPC servers to compute on client's private input. But unlike P2DEX we don't use burner addresses, but instead a holding smart contract Lock, administered by a *single* distributed signing key. But we note that the P2DEX approach will also work with the smart contract based approach. Thus by simply having Lock smart contracts instantiated on multiple blockchains, with different administration keys, these can form the same purpose as the burner addresses in P2DEX. Concretely this can be realized by simply having each client provide a confidential token commitment on each chain they expect to receive some tokens. Note that such a commitment can be of 0 tokens. The MPC servers will then validate all the confidential tokens given to Lock on each of the different chains, through the *verify input* phase. Then one, unified privacy preserving smart contract ConfContract will be executed, which will yield new commitments for each of the relevant clients on each chain. The clients can then use *withdraw* in Lock on each of the different chains to finish the computation and get their confidential tokens on the relevant chains.

This approach could of course also be combined with the mixer idea above, allowing for cross-chain mixers with selective levels of privacy depending on the amounts mixed by a given user.

Doing cross-chain exchange on hidden tokens also has the advantage of allowing parties, with very large amounts of tokens, to carry out an exchange in a slow and continues manner, thus preventing sudden exchange fluctuations. In fact, our system could enforce an upper bound on the amount of tokens to exchange in one round, and automatically split up large orders so they get completed over multiple rounds, instead of just one.

Security. The overall security of this approach follows from P2DEX, although we will also argue that intuitively there is nothing non-trivial to simulation if we adjust our ideal functionalities and protocol to follow this approach. Basically where there is formal cryptography to be proven is in the integration between the different ideal functionalities, in particular when ensuring consistency between the input to outsourced MPC and the commitments transferred to the holding contract. However, using our scheme across multiple chains make no difference in this. The only modelling difference is simply that the ideal blockchain functionalities can no be considered to "wrap" different instances of the same functionality (thus reflecting multiple chains). Such a wrap inherently does not affect secure insofar that it does not contain any logical loopholes.

F.4 Future work

While we construct and prove UC-secure a scheme for decentralized privacy preserving smart contracts, we believe there are multiple paths for future work to explore. An immediate interesting path is to implement and benchmark the system for some of the applications we have discussed. For example, a better formalization of the cross-chain approach, along with an investigation of MPC friendly algorithms for fair matching of exchange orders could allow the realization of a highly secure and private decentralized exchange. For such applications it also becomes important to investigate, the logic of how to use the collateral to punish malicious parties in case they cheat. In particular such that no rational party will end up with a skewed or perverse incentives. In relation to this, it would be interesting to investigate how to integrate Pedersen commitments with MPC in an efficient way, *without* requiring the MPC computation domain to be the same as the Pedersen message space. This could have a great affect on the efficiency if the MPC computation domain is significantly smaller than 256 bit. Currently we require the sharing the opening information of commitments to happen in a P2P manner, off-chain, when transferring hidden tokens. It would be interesting to investigate how to implement hidden tokens in a way that does not require client-to-client communication when doing transfers, while working with the rest our protocol. In relation to this, other ways the overall usability of our system could also be improved is constructing a protocol leveraging other existing results to allow stateless clients. For example through some notion of password authenticated distributed secret sharing [18]. In continuation of this, investigating how to prevent the use of user-supplied masks for each round of execution private smart contract computation, would also give a great impact on the usability of our solution.

Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$

$\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ is run by clients $\mathcal{C}$ and committee $\mathcal{Q}$. The protocol runs in the presence of $\mathcal{F}_{\mathsf{Ledger}}, \mathcal{F}^{R}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Sig}}$ instances. Initially, only accept inputs GenAcct and InitLedger.

GenAcct: Upon $(\textsc{GenAcct}, sid)$, obtain fresh vk from $\mathcal{F}_{\mathsf{Sig}}$. Set key store to $\mathcal{K} \leftarrow \mathcal{K} \cup \{\mathsf{vk}\}$, and return $(\textsc{NewAcct}, sid, \mathsf{vk})$.

InitLedger: Upon $(\textsc{InitLedger}, sid, \mathcal{L}_{\mathsf{Init}}, \mathsf{vk})$, client C parses $\mathcal{L}_{\mathsf{Init}}$ as a map from a set of signature keys to token balances $\mathbb{G} \mapsto (\mathbb{T} \mapsto \mathbb{Z})$ and asserts $\mathsf{vk} \in \mathcal{K}$.

1. C initializes $\mathcal{F}_{\mathsf{Ledger}}$ with $\mathcal{Q}$ and signs $m = (\textsc{Init}, sid, \mathcal{L}_{\mathsf{Init}})$ via $\mathcal{F}_{\mathsf{Sig}}$ with key vk. Send $(\textsc{Post}, sid, m, \mathsf{vk}, \sigma_{\mathsf{vk}}(m))$ to $\mathcal{F}_{\mathsf{Ledger}}$.
2. C compiles $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ to initial contract state and circuit (γ, T). Sign $m = (\textsc{Deploy}, sid, \gamma, T, \mathsf{vk})$ via $\mathcal{F}_{\mathsf{Sig}}$ with vk, and send $(\textsc{Post}, sid, m, \mathsf{vk}, \sigma_{\mathsf{vk}}(m))$ to $\mathcal{F}_{\mathsf{Ledger}}$. Ignore further InitLedger inputs and accept all other inputs.

Transfer: Upon $(\textsc{Transfer}, \bar{\mathbf{v}}, \mathsf{vk}_{\mathsf{rcv}}, \mathsf{vk})$, obtain $\mathcal{L}$ from GetLedger procedure. Assert $\mathsf{vk} \in \mathcal{K}$ and $\mathcal{L}[\mathsf{vk}] \geq \bar{\mathbf{v}}$. Sign $m = (\textsc{Transfer}, sid, \bar{\mathbf{v}}, \mathsf{vk}_{\mathsf{rcv}})$ via $\mathcal{F}_{\mathsf{Sig}}$ with key vk and send $(\textsc{Post}, sid, m, \mathsf{vk}, \sigma_{\mathsf{vk}}(m))$ to $\mathcal{F}_{\mathsf{Ledger}}$.

Mint: On $(\textsc{Mint}, sid, \bar{\mathbf{v}}, \mathsf{vk})$, client C,

1. Assert $\mathsf{vk} \in \mathcal{K}$, obtain state $\mathcal{L}, \Gamma$ from $\mathcal{F}_{\mathsf{Ledger}}$ and assert $\mathcal{L}[\mathsf{vk}] \geq \mathbf{v}$.
2. Sample $r \leftarrow\$\, \mathbb{F}$, compute $c \leftarrow \mathsf{com}(\bar{\mathbf{v}}, r)$ and obtain string π from $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ where $\mathcal{R}(c, \bar{\mathbf{v}}; r) = \{c = \mathsf{com}(\bar{\mathbf{v}}, r)\}$.
3. Sign $(\textsc{Call}, sid, (\mathsf{cn}, f(\textsc{Mint}), (c, \pi), \bar{\mathbf{v}}))$ via $\mathcal{F}_{\mathsf{Sig}}$ with vk and post to $\mathcal{F}_{\mathsf{Ledger}}$.
4. Set wallet $\mathcal{W}[\mathsf{vk}] \leftarrow \mathcal{W}[\mathsf{vk}] \cup \{\langle \mathsf{id} = c, (\bar{\mathbf{v}}, r)\rangle\}$ and return $(\textsc{Minted}, sid, \mathsf{id})$.

ConfTransfer: On $(\textsc{ConfTrfr}, sid, \mathcal{C}_{\mathsf{rcv}}, \mathsf{vk}_{\mathsf{rcv}}, \{\mathsf{id}_i, \bar{\mathbf{v}}'_i\}_{i \in \{1,2\}})$, client C:

1. Assert $\exists \mathsf{vk}_{\mathsf{src}} \in \mathsf{dom}(\mathcal{W}) : (\mathsf{id}_i, (\bar{\mathbf{v}}_i, r_i)) \in \mathcal{W}[\mathsf{vk}_{\mathsf{src}}]$, $\bar{\mathbf{v}}_1 + \bar{\mathbf{v}}_2 = \bar{\mathbf{v}}'_1 + \bar{\mathbf{v}}'_2$, $\bar{\mathbf{v}}'_i \leq \bar{\mathbf{v}}_{\mathsf{max}}$.
2. For $i \in \{1, 2\}$, sample $r'_i \leftarrow\$\, \mathbb{F}$ such that $\sum_{i \in \{1,2\}} r'_i = \sum_{i \in \{1,2\}} r_i$ and compute $c'_i = \mathsf{com}(\bar{\mathbf{v}}'_i, r'_i)$ and π_i via $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ that proves $\mathcal{R}(c'_i; \bar{\mathbf{v}}'_i, r'_i) = \{\bar{\mathbf{v}}'_i \leq \bar{\mathbf{v}}_{\mathsf{max}}\}$
3. Sign and post $(\textsc{Call}, sid, (\mathsf{cn}, f(\textsc{ConfTransfer}), x, 0^t), \mathsf{vk}_{\mathsf{src}})$ to $\mathcal{F}_{\mathsf{Ledger}}$, where $x = (\{c_i, c'_i, \pi_i\}_{i \in \{1,2\}}, \mathsf{vk}_{\mathsf{rcv}})$.
4. Send $(\textsc{ConfTrfr}, sid, \bar{\mathbf{v}}'_1, r'_1, \mathsf{vk}_{\mathsf{src}})$ to $\mathcal{C}_{\mathsf{rcv}}$, which stores it.
5. Set $\mathcal{W}[\mathsf{vk}_{\mathsf{src}}] \leftarrow \mathcal{W}[\mathsf{vk}_{\mathsf{src}}] \cup (\mathsf{id}'_2 = c'_2, (\bar{\mathbf{v}}'_2, r'_2))$ and return $(\textsc{Change}, sid, \mathsf{id}'_2)$.

ConfReceive: On $(\textsc{ConfReceive}, sid)$ client C:

1. For $\mathsf{vk} \in \mathsf{dom}(\mathcal{W})$, retrieve $\{(\bar{\mathbf{v}}_i, r_i, \mathsf{vk}_i)\}_{i \in [l]}$ received from clients since the last ConfReceive input and $\mathcal{L}_{\mathsf{Conf}}$ from $\mathcal{F}_{\mathsf{Ledger}}$.
2. For $(\bar{\mathbf{v}}, r, \mathsf{vk}) \in \{(\bar{\mathbf{v}}, r, \mathsf{vk}_i)\}_{i \in [l]}$, compute $c = \mathsf{com}(\bar{\mathbf{v}}, r)$ and assert $c \in \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}]$.
   - If satisfied, add $(\mathsf{id} = c, (\bar{\mathbf{v}}, r))$ to $\mathcal{W}[\mathsf{vk}]$.
3. Returns $(\textsc{Received}, (\mathsf{vk}_1, \langle \mathsf{id}_1, \bar{\mathbf{v}}_1\rangle), ..., (\mathsf{vk}_l, \langle \mathsf{id}'_l, \bar{\mathbf{v}}'_l\rangle))$ for l' received transfers.

Redeem: On $(\textsc{ConfRdm}, sid, \mathsf{id})$ client C,

1. If $\exists(\mathsf{vk}, \bar{\mathbf{v}}, r) : (\mathsf{id}, (\bar{\mathbf{v}}, r)) \in \mathcal{W}[\mathsf{vk}]$, where $\mathsf{id} = \mathsf{com}(\bar{\mathbf{v}}, r)$.
2. Compute π via $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ which proves $\mathcal{R}(c, \bar{\mathbf{v}}; r) = \{c = \mathsf{com}(\bar{\mathbf{v}}, r)\}$.
3. Sign and post $(\textsc{Call}, sid, (\mathsf{cn}, f(\textsc{Redeem}), (\bar{\mathbf{v}}, c, \pi), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.

GetLedger: Upon $(\textsc{GetLedger}, sid)$, client C obtains $(\mathcal{L}, \Gamma)$ and contract id cn from $\mathcal{F}_{\mathsf{Ledger}}$, reads $(\gamma, \mathbf{w}, T) \leftarrow \Gamma[\mathsf{cn}]$, and parses γ as $(\mathcal{L}_{\mathsf{Conf}}, \bar{\mathbf{m}})$. C outputs $(\textsc{Ledger}, sid, \mathcal{L}, \mathcal{L}_{\mathsf{Conf}})$.

ExtCall: Upon $(\textsc{ExtCall}, sid, \mathcal{F}, \mathsf{m})$ received from $\mathcal{F}_{\mathsf{Ledger}}$, party $Q \in \mathcal{Q}$ forwards m to hybrid instance $\mathcal{F}$ and waits. Upon response ret from $\mathcal{F}$, party Q forwards $(\textsc{CallBack}, sid, \mathsf{ret})$ to $\mathcal{F}_{\mathsf{Ledger}}$.

Fig. 10: Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ UC-securely realizing $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

Program $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

On input $(\nu \,|\, \gamma \,|\, \bar{\mathbf{w}} \,|\, \mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}} \,|\, \mathsf{vk})$, parses function selector fn and execute function routine with input string $x \in \{0,1\}^*$, parsed according to function descriptions below. Further, parse contract state γ as $\mathcal{L}_{\mathsf{Conf}}$, where $\mathcal{L}_{\mathsf{Conf}} : \mathbb{G}^{\mathsf{vrk}} \to \{\mathbb{G}^{\mathsf{com}}, ...\}$. $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ is parameterized with committee $\mathcal{Q}$.

Mint: parse x as (c, π) where $c \in \mathbb{G}$ and $\pi \in \{0,1\}^*$.
1. Send call to $\mathcal{Q}$ with msg for $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ to verify that π proves $\mathcal{R}(\bar{\mathbf{v}}, c; r) = \{c = \mathbf{g}^{\bar{\mathbf{v}}} h^r\}$.
2. Set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \cup \{c\}$, $\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \bar{\mathbf{v}}$ and $\mathcal{L}[\mathsf{vk}] \leftarrow \mathcal{L}[\mathsf{vk}] - \bar{\mathbf{v}}$.
3. Output updated $(\mathcal{L}, \gamma' = \mathcal{L}_{\mathsf{Conf}}, \bar{\mathbf{w}})$.

ConfTransfer: parse x as $(c_1, c_2, c_1', c_2', \pi_1, \pi_2, \mathsf{vk}_{\mathsf{rcv}})$ where $c_i, c_i' \in \mathbb{G}$, $\pi_i \in \{0,1\}^*$ for $i \in \{1,2\}$ and $\mathsf{vk}_{\mathsf{rcv}} \in \mathbb{G}$.
1. Assert $\{c_1, c_2\} \in \mathcal{L}^{\mathsf{Conf}}[\mathsf{vk}]$ and that $c_1 \cdot c_2 = c_1' \cdot c_2'$ holds.
2. Send call to $\mathcal{Q}$ with msg for $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ to verify $\forall i \in \{1,2\}$: π_i proves $\mathcal{R}(c_i'; \bar{\mathbf{v}}_i', r_i') = \{\bar{\mathbf{v}}_i' \leq \bar{\mathbf{v}}_{\mathsf{max}} \wedge c_i' = \mathbf{g}^{\bar{\mathbf{v}}_i'} h^{r_i'}\}$.
3. Set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \backslash \{c_1, c_2\} \cup \{c_2'\}$ and $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_{\mathsf{rcv}}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_{\mathsf{rcv}}] \cup \{c_1'\}$.
4. Output updated $(\mathcal{L}, \gamma' = \mathcal{L}_{\mathsf{Conf}}, \bar{\mathbf{w}})$.

Redeem: parse x as $(\bar{\mathbf{v}}, c, \pi)$, $c \in \mathbb{G}$ and $\pi \in \{0,1\}^*$.
1. Assert $c \in \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}]$ and $\bar{\mathbf{w}} \geq \bar{\mathbf{v}}$.
2. Send call to $\mathcal{Q}$ with msg for $\mathcal{F}^{\mathcal{R}}_{\mathsf{NIZK}}$ to verify π proves $\mathcal{R}(\bar{\mathbf{v}}, c; r) = \{c = \mathbf{g}^{\bar{\mathbf{v}}} h^r\}$.
3. Set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \backslash \{c\}$, $\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} - \bar{\mathbf{v}}$.
4. Output updated $(\mathcal{L}, \gamma' = \mathcal{L}_{\mathsf{Conf}}, \bar{\mathbf{w}})$.

Fig. 11: The smart contract code $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ for confidential tokens.

I/II: Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$, extends $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$

All clients and servers are registered with $\mathcal{F}_{\mathsf{Clock}}$.

Init: On $(\textsc{Init}, sid, g)$ server $P \in \mathcal{P}$,
1. Runs **GenAcct** in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ to generate signature verification key vk, sends to $\mathcal{P}$.
2. Runs **InitLedger** in $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$ to initialize $\mathcal{F}_{\mathsf{Ledger}}$; here, $P \in \mathcal{P}$ obtains fresh vk.
3. Jointly samples key $\mathsf{vk}_{\mathsf{TSig}}$ via $\mathcal{F}_{\mathsf{TSig}}$ with $\mathcal{P}$.
4. Deploys $\mathcal{X}_{\mathsf{Lock}}[\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}]$ and $\mathcal{X}_{\mathsf{Collateral}}$ to $\mathcal{F}_{\mathsf{Ledger}}$.
   a. Obtains contract instance id's $\mathsf{cn}_{\mathsf{Lock}} = \mathsf{cn}_{\underline{\mathsf{C}}\mathsf{Ledger}}$ and $\mathsf{cn}_{\mathsf{Coll}}$ from $\mathcal{F}_{\mathsf{Ledger}}$.
   b. Signs and sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Init}), (\mathsf{vk}_{\mathsf{TSig}}), 0^{|\mathbb{T}|}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.
   c. Sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Coll}}, f(\textsc{Deposit}), (\mathsf{vk}_{\mathsf{TSig}}), \bar{\mathbf{v}}_{\mathsf{Coll}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.
5. Initializes $\mathcal{F}_{\mathsf{Ident}}$, asserts circuit depth of $\mathsf{depth}(g) \leq d_T$ and stores it.
6. Updates $\mathcal{F}_{\mathsf{Clock}}$.

Enroll: Upon input $(\textsc{Enroll}, sid, x, \mathsf{id}, \mathsf{vk})$, client $C \in \mathcal{C}$:
1. Asserts $\exists(\mathsf{id}, (\bar{\mathbf{v}}, \bar{r})) \in \mathcal{W}[\mathsf{vk}]$ and $\mathsf{cn}_{\mathsf{Lock}}$, $\mathsf{cn}_{\mathsf{Coll}}$ are in `enroll`/`coll`.
2. Generate output masks:
   a. Samples and stores $\hat{y}, \hat{\mathbf{w}} = (\hat{w}_1, ..., \hat{w}_{|\mathbb{T}|}), \hat{r} \leftarrow\$ \mathbb{F}$.
   b. Computes and stores $\hat{c} \leftarrow \mathsf{com}(\hat{\mathbf{w}}, \hat{s})$.
3. Sends client input and output masks $(x, (\bar{\mathbf{v}}, \bar{r}), (\hat{\mathbf{w}}, \hat{s}))$ to $\mathcal{F}_{\mathsf{Ident}}$.
4. Sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Enroll}), (c = \mathsf{com}(\bar{\mathbf{v}}, \bar{r}), \hat{c}), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.
5. Removes $(\mathsf{id}, (\bar{\mathbf{v}}, \bar{r}))$ from $\mathcal{W}[\mathsf{vk}]$ and updates $\mathcal{F}_{\mathsf{Clock}}$.

Verify input: Upon input $(\textsc{Execute}, sid)$, if $\mathcal{F}_{\mathsf{Clock}}$ has progressed since last activation and $\mathsf{cn}_{\mathsf{Lock}}$, $\mathsf{cn}_{\mathsf{Coll}}$ are in `enroll`<u>`ed`</u> and `coll` respectively, server $P_i \in \mathcal{P}$ performs:
1. P_i obtains client input coins and masks $\{(c_1, \hat{c}_1), ..., (c_m, \hat{c}_m)\}$ from $\mathcal{F}_{\mathsf{Ledger}}$.
2. For verification of client inputs $\{(\bar{\mathbf{v}}_j, \bar{r}_j, c_{j,})\}_{j \in [m]}$, P_i performs:
   a. Servers interact with $\mathcal{F}_{\mathsf{Ident}}$ and call following interfaces:
   - **Evaluate**: $[\,\bar{\mathbf{a}}\,], [\,\bar{b}\,], [\,\gamma\,] \leftarrow \mathsf{rand}()$[a]
   - **Open** $\gamma \leftarrow [\,\gamma\,]$.
   - **Get Shares**: $\bar{\mathbf{a}}^{(i)} = (\bar{a}_1^{(i)}, ..., \bar{a}_{|\mathbb{T}|}^{(i)}), \bar{b}^{(i)}, \{\bar{\mathbf{v}}_j^{(i)} = (\bar{v}_{j,1}^{(i)}, ..., \bar{v}_{j,|\mathbb{T}|}^{(i)}), \bar{r}_j^{(i)}\}_{j \in [m]}$

   a. Local computation of the following and sends resulting shares to all $\mathcal{P}$:
   - $\bar{c}_{\mathbf{a},b}^{(i)} \leftarrow \mathsf{com}(\bar{\mathbf{a}}^{(i)}, \bar{b}^{(i)})$, $\bar{v}_t^{(i)\prime} \leftarrow \bar{a}_t^{(i)} + \sum_{j \in [m]} \bar{v}_{j,t}^{(i)} (\gamma^{(i)})^j$ for $t \in [|\mathbb{T}|]$
   - $\bar{r}^{(i)\prime} \leftarrow \bar{b}^{(i)} + \sum_{j \in [m]} \bar{r}_j^{(i)} (\gamma^{(i)})^j$

   b. Each P_i reconstructs $(\bar{\mathbf{v}}' = \bar{v}_1', ..., \bar{v}_{|\mathbb{T}|}', \bar{r}')$, from shares and
   - Asserts: $\prod_{i \in [n]} \bar{c}_{\mathbf{a},b}^{(i)} \cdot \prod_{j \in [m]} (c_{j,\mathsf{in}})^{\gamma^j} = \mathbf{g}^{\bar{\mathbf{v}}'} h^{\bar{r}'}$
3. Servers repeats for the batch verification of client masks $\{(\hat{\mathbf{w}}_j, \hat{s}_j, \hat{c}_j)\}_{j \in [m]}$.
4. Server P_i updates $\mathcal{F}_{\mathsf{Clock}}$.

[a] e.g. XOR circuit evaluated on random inputs.

Fig. 12: Part 1 - Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ UC-securely realizing $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$.

II/II: Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$, extends $\Pi_{\underline{\mathsf{C}}\mathsf{Ledger}}$

Evaluate: After *verify input* and if $\mathcal{F}_{\mathsf{Clock}}$ has progressed, $\mathsf{cn}_{\mathsf{Lock}}$ is in `enroll`$\underline{\texttt{ed}}$, each server $P_i \in \mathcal{P}$ performs:

1. It interacts with following interfaces of $\mathcal{F}_{\mathsf{Ident}}$.
   a. Runs **Evaluate** on circuit g with secret client inputs $\{(x_j, \bar{\mathbf{v}}_j)\}_{j\in[m]}$.
   b. Runs **Evaluate** to apply masks over output gate values of circuit g:
      - For each $j \in [m]$: $([\,y'_j\,], [\,\bar{\mathbf{w}}'_j\,]) \leftarrow ([\,y_j\,], [\,\bar{\mathbf{w}}_j\,]) + ([\,\hat{y}_j\,], [\,\hat{\mathbf{w}}_j\,])$
   c. Runs **Share** for obtain shares of $\{([\,y'_j\,], [\,\bar{\mathbf{w}}'_j\,])\}_{j\in[m]}$ from $\mathcal{F}_{\mathsf{Ident}}$.
2. Jointly signs $\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{eval})$ with $\mathcal{P}$ via $\mathcal{F}_{\mathsf{TSig}}$: if $\mathcal{F}_{\mathsf{TSig}}$ aborts, runs **abort**.
3. Sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Lock}), (\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{eval})), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$, updates $\mathcal{F}_{\mathsf{Clock}}$.

Open: Upon running *evaluate* and $\mathcal{F}_{\mathsf{Clock}}$ has progressed, each server $P_i \in \mathcal{P}$:

1. Runs **Optimistic Reveal** in $\mathcal{F}_{\mathsf{Ident}}$ for masked outputs $\mathsf{mout} = \{(y'_j, \bar{\mathbf{w}}'_j)\}_{j\in[m]}$.
2. Jointly signs $\mathsf{sig} = \sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{mout})$ via $\mathcal{F}_{\mathsf{TSig}}$: if *abort* is returned, run **abort**.
3. Sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Settle}), (\mathsf{mout}, \mathsf{sig}), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$, updates $\mathcal{F}_{\mathsf{Clock}}$.

Withdraw: Upon $(\textsc{Withdraw}, sid)$, each client $C_j \in \mathcal{C}$ performs:

1. Retrieves all masked outputs $\{(y'_{j,1}, \bar{\mathbf{w}}'_{j,1}), ..., (y'_{j,l}, \bar{\mathbf{w}}'_{j,l})\}$ added to $\mathsf{cn}_{\mathsf{Lock}}$ on $\mathcal{F}_{\mathsf{Ledger}}$ since last *withdraw* activation.
2. For each retrieved masked output, reads corresponding mask values $(\hat{y}_j, \hat{\mathbf{w}}_j, \hat{s}_j)$ stored locally and computes $y_j = y'_j - \hat{y}_j$, $\bar{\mathbf{w}}_j \leftarrow (\bar{\mathbf{w}}'_j - \hat{\mathbf{w}}_j)$.
   - Samples id'_j and set $\mathcal{W}[\mathsf{vk}_j] \leftarrow \mathcal{W}[\mathsf{vk}_j] \cup \{(\mathsf{id}'_j, (\bar{\mathbf{w}}_j, -\hat{s}_j))\}$.
3. Returns $(y_{j,1}, \langle \mathsf{id}'_{j,1}, \bar{\mathbf{w}}_{j,1}\rangle), ..., (y_{j,l}, \langle \mathsf{id}'_{j,l}, \bar{\mathbf{w}}_{j,l}\rangle)$.

Abort: Upon receiving $(\textsc{Abort}, sid)$, each $P_i \in \mathcal{P}$ ticks $\mathcal{F}_{\mathsf{Clock}}$:

1. If $\mathsf{cn}_{\mathsf{Lock}}$ is in state `enroll` or `enroll`$\underline{\texttt{ed}}$,
   signs and sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Abort}), \{0, \bar{\mathbf{0}}\}_{j\in[m]}), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.
2. Else if $\mathsf{cn}_{\mathsf{Lock}}$ is in state `lock`, run **Reveal** and **Allow Verify** with $\mathcal{F}_{\mathsf{Ident}}$.
   - If $\mathcal{F}_{\mathsf{Ident}}$ returns cheating servers $\mathcal{J}$
     signs and sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Abort}), \{0, \bar{\mathbf{0}}\}_{j\in[m]}), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$.
   - Else server P_i obtains $\{y_j^{(i)}, \bar{\mathbf{w}}_j^{(i)}\}_{j\in[m]}$ from **Reveal**, signs and
     sends $(\textsc{Call}, sid, (\mathsf{cn}_{\mathsf{Lock}}, f(\textsc{Abort}), \{y_j^{(i)}, \bar{\mathbf{w}}_j^{(i)}\}_{j\in[m]}), \bar{\mathbf{0}}), \mathsf{vk})$ to $\mathcal{F}_{\mathsf{Ledger}}$
3. Updates $\mathcal{F}_{\mathsf{Clock}}$ and *terminates.*

Fig. 13: Part 2 - Protocol $\Pi_{\underline{\mathsf{C}}\mathsf{Contract}}$ UC-securely realizing $\mathcal{F}_{\underline{\mathsf{C}}\mathsf{Contract}}$.

Program $\mathcal{X}_{\mathsf{Lock}}$, extends $\mathcal{X}_{\underline{\mathsf{C}}\mathsf{Ledger}}$

On input $(\nu \,|\, \gamma \,|\, \bar{\mathbf{w}} \,|\, \mathsf{cn}, \mathsf{fn}, x, \bar{\mathbf{v}}, \mathsf{vk})$, $\mathcal{X}_{\mathsf{Lock}}$ parses γ as $(\mathcal{L}_{\mathsf{Conf}}, \mathcal{L}_{\mathsf{Lock}}, \mathcal{M}, \mathcal{J}, \mathsf{st})$, where $\mathcal{L}_{\mathsf{Lock}}$ is a ledger of locked coins, $\mathcal{M}$ is a map of account verification keys to coin commitment masks, $\mathcal{J}$ is set of cheating servers and $\mathsf{st} \in \{\mathtt{enroll}, \mathtt{enroll}\underline{\mathtt{ed}}, \mathtt{lock}\}$ captures the phase the confidential coin lock is currently in.

Init lock: parse x as $\mathsf{vk}_{\mathsf{TSig}}$, set state to $\mathtt{enroll}$.

Enroll: parse x as $(c, \hat{c}) \in \mathbb{G}^{\mathsf{com}} \times \mathbb{G}^{\mathsf{com}}$.
1. Assert state is $\mathtt{enroll}$ and $c \in \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}]$. Set $\mathcal{L}_{\mathsf{Lock}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Lock}}[\mathsf{vk}] \cup \{c\}$, $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \setminus \{c\}$ and $\mathcal{M}[\mathsf{vk}] \leftarrow \hat{c}$.
2. If round ν has progressed since last state transition to $\mathtt{enroll}$, set st to $\mathtt{enroll}\underline{\mathtt{ed}}$. Return updated $(\mathcal{L}, \gamma' = (\mathcal{L}_{\mathsf{Conf}}, \mathcal{L}_{\mathsf{Lock}}, \mathcal{M}, \mathcal{J}, \mathsf{st}), \bar{\mathbf{w}})$.

Evaluated: parse x as $\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{eval}) \in \{0,1\}^*$.
1. Assert state st is $\mathtt{enroll}\underline{\mathtt{ed}}$, verify $\sigma_{\mathsf{vk}_{\mathsf{TSig}}}(\mathsf{eval})$ via $\mathcal{F}_{\mathsf{TSig}}$.
2. If no *abort* is returned and two $\mathcal{F}_{\mathsf{Clock}}$ rounds (observed via ν) have progressed since last state transition to $\mathtt{enroll}\underline{\mathtt{ed}}$: set st to $\mathtt{lock}$.
3. Return updated $(\mathcal{L}, \gamma' = (\mathcal{L}_{\mathsf{Conf}}, \mathcal{L}_{\mathsf{Lock}}, \mathcal{M}, \mathcal{J}, \mathsf{st}), \bar{\mathbf{w}})$.

Settle: parse x as $\sigma_{\mathsf{vk}_{\mathsf{Lock}}}(\{y_j, \bar{\mathbf{v}}_j\}_{j \in [m]})$.
1. Assert state is $\mathtt{lock}$, verify $\sigma_{\mathsf{vk}_{\mathsf{Lock}}}(\{y_j, \bar{\mathbf{v}}_j\}_{j \in [m]})$ via $\mathcal{F}_{\mathsf{TSig}}$.
2. If no *abort* is returned and $\mathcal{F}_{\mathsf{Clock}}$ round has progressed, run $\mathtt{payout}(\{\bar{\mathbf{v}}_j\}_{j \in [m]})$.

Abort: parse x as $\{y_j^{(i)}, \bar{\mathbf{v}}_j^{(i)}\}_{j \in [m]}$.
1. If state is $\mathtt{enroll}$ or $\mathtt{enroll}\underline{\mathtt{ed}}$, run $\mathtt{reimburse}$.
2. Else if state is $\mathtt{lock}$,
   - Send (Test-Reveal, *sid*) and (Verify, *sid*, $\{y_j^{(i)}, \bar{\mathbf{v}}_j^{(i)}\}_{j \in [m]}$) to $\mathcal{F}_{\mathsf{Ident}}$.
   - If no *abort* is returned and $\mathcal{F}_{\mathsf{Clock}}$ round has progressed, reconstruct $\{\bar{\mathbf{v}}_j\}_{j \in [m]}$ when all shares received from $\mathcal{P}$ and run $\mathtt{payout}(\{\bar{\mathbf{v}}_j\}_{j \in [m]})$.
   - Else if *abort* is returned from $\mathcal{F}_{\mathsf{Ident}}$ or shares are missing, record cheating servers $\mathcal{J}$ and run $\mathtt{reimburse}(\mathcal{J})$.

$\mathtt{payout}(\{\bar{\mathbf{v}}_j\}_{j \in [m]})$: for each C_j's output $\bar{\mathbf{v}}_j \in \{\bar{\mathbf{v}}_j\}_{j \in [m]}$,
1. $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}_j] \cup \{c_j \cdot \hat{c}_j^{-1}\}$, where $c_j = \mathbf{g}^{\bar{\mathbf{v}}_j} h^0$, $\hat{c}_j = \mathcal{M}[\mathsf{vk}_j]$. $\mathcal{L}_{\mathsf{Lock}} \leftarrow \emptyset$, $\mathcal{M} \leftarrow \emptyset$. Return updated $(\mathcal{L}, \gamma' = (\mathcal{L}_{\mathsf{Conf}}, \mathcal{L}_{\mathsf{Lock}}, \mathcal{M}, \mathcal{J}, \mathtt{enroll}), \bar{\mathbf{w}})$.

$\mathtt{reimburse}(\mathcal{J})$: for $\mathsf{vk} \in \mathsf{dom}(\mathcal{L}_{\mathsf{Lock}})$,
1. Set $\mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \leftarrow \mathcal{L}_{\mathsf{Conf}}[\mathsf{vk}] \cup \mathcal{L}_{\mathsf{Lock}}[\mathsf{vk}]$.
2. Set $\mathcal{L}_{\mathsf{Lock}} \leftarrow \emptyset$, $\mathcal{M} \leftarrow \emptyset$. Return updated $(\mathcal{L}, \gamma' = (\mathcal{L}_{\mathsf{Conf}}, \mathcal{L}_{\mathsf{Lock}}, \mathcal{M}, \mathcal{J}, \mathtt{enroll}), \bar{\mathbf{w}})$.

Fig. 14: The smart contract code $\mathcal{X}_{\mathsf{Lock}}$ for extended confidential token functionality.

Program $\mathcal{X}_{\mathsf{Collateral}}$

Parameterized by signature verification keys $\{\mathsf{vk}_1, ..., \mathsf{vk}_n\}$ associated with servers $\mathcal{P} = \{P_1, ..., P_n\}$, contract identifier $\mathsf{cn}_{\mathsf{Lock}}$ and collateral threshold $\bar{\mathbf{v}}_{\mathsf{coll}}$.

Deposit collateral:
1. Assert local state is `deposit`, $\mathsf{cn}_{\mathsf{Lock}}$ state is `enroll`, and $\bar{\mathbf{v}}_{\mathsf{in}} \geq \bar{\mathbf{v}}_{\mathsf{coll}}$.
2. If collateral received by accounts associated with vk for $i \in [n]$, set state to `coll`.

Round activation: If $\mathcal{F}_{\mathsf{Clock}}$ round has progressed since update to `coll`.
1. If $\mathsf{cn}_{\mathsf{Lock}}$ is in `deposit`, return collateral and set state to `deposit`.
2. Else if $\mathsf{cn}_{\mathsf{Lock}}$ is in `abort` with cheating $\mathcal{J} \subseteq \mathcal{P}$, distribute $\mathcal{J}$'s collateral to $\mathcal{C}'$. Return collateral of honest servers. Set state to `deposit`.

Fig. 15: The smart contract code $\mathcal{X}_{\mathsf{Collateral}}$.

Fuzzy Order Matching: Differentially Private Market Mechanisms with MPC

Contribution

- Co-author.

Remarks

Preliminary results from this work have been presented at the *3rd Workshop on Decentralized Finance* at Financial Cryptography and Data Security 2023. The manuscript in this thesis contains the main results; implementation and performance benchmarks in MPC are work-in-progress.

Fuzzy Order Matching: Differentially Private Market Mechanisms with MPC

James Hsin-yu Chiang
Technical University of Denmark, Denmark

Bernardo David
IT University of Copenhagen, Denmark

Mariana Gama
imec-COSIC, KU Leuven, Belgium

Christian Janos Lebeda
IT University of Copenhagen, Denmark
Basic Algorithms Research Copenhagen, Denmark

Abstract

We present the first *differentially private* market mechanisms that formally mitigate information leakage from all trading activity of a user, and instantiate it with an MPC engine, with the goal of mitigating front-running and protecting the privacy of long-running strategies in both traditional dark pools and decentralized finance.

Towards this goal, we first extend the notion of differential privacy to the setting of n clients and a trusted curator; in each round, clients provide private inputs during the input phase, upon which the trusted curator evaluates a function over submitted inputs. Then, the trusted curator outputs private outputs to each client. We propose *round-differential privacy* to protect the honest client transcript from leaking to the output of corrupted clients; here, protecting client inputs as not sufficient, as round-differential privacy must also protect against *correlations* between honest and adversarial outputs; in fact, this leakage between outputs is common in economic functionalities which allocate resources across participants. To the best of our knowledge, this setting was not previously considered by the differential privacy community and may be of independent interest.

We show that traditional market mechanisms in the dark-pool setting do not satisfy round-differential privacy and thus cannot formally guarantee pre- and post-trade privacy; a matched order by a corrupted client always reveals the presence of an opposing order, potentially that of the honest client. In response, we propose novel, round-differentially private market mechanisms which guarantee both pre- and post-trade privacy for the trader by satisfying round-differential privacy in the trusted curator model. To achieve round-differential privacy, *fuzzy order matching* is performed; this ensures privacy for the honest party, but also requires a privacy-preserving liquidity provider mechanism to compensate for the potential liquidity mismatch, as orders are no longer guaranteed to execute pairwise. We propose application-specific secure multi-party computation (MPC) protocols to realize our proposed market mechanisms and implement these in the Scale-Mamba Framework using Shamir Secret Sharing based MPC. We demonstrate practical trading throughput with minimal overhead induced by differential privacy sub-routines.

Keywords and phrases Differential Privacy, Secure Multi-party Computation, Dark Pools, Decentralized Finance

1 Introduction

The term front-running originates from the notion of "getting in front" of pending trades. A party anticipating a large buy order may purchase the same asset first, as the pending large buy order will likely drive up the price of the asset; the front-running party can then sell the asset at a higher price following the execution of the large buy order. Front-running occurs whenever submitted trade orders that have yet to be executed are observable by the front-running adversary. In traditional finance, the presence of pending orders may be public or inferred from market order books. In decentralized finance, pending transactions are publicly gossiped across a peer-to-peer network. In both settings, front-running is prevalent.

In traditional finance, Dark Pool venues [23] promise the private execution of trades. Here, clients submit private orders to the venue operator, who then computes the execution of trades without leaking pending orders submitted by clients; **pre-trade privacy** ensures that pending orders remain private, whilst **post-trade privacy** protects the privacy of the trade execution. Yet, we observe that post-trade privacy guarantees are impossible in classical order-matching algorithms; a trade execution always implies a counter-party, thus revealing the presence of another trade in the opposing direction (Lemmas 4 and 6). In venues with low trade volume, such inferences may lead to practical attacks.

This motivates our investigation of market mechanisms with formal, differential privacy guarantees for the entire transcript of interactions between trader and dark pool venue. We propose the following main contributions.

(1) Round-differential privacy. We formalize a novel notion of round-differential privacy (Def. 7) applicable to the **trusted curator model** (Sec. 3); here, a set of clients interact with a trusted curator over private and secure communication channels. Interaction between n clients and curator occurs round-wise, where the adversary can corrupt up to $n-1$ clients, and thereby inject inputs and observe outputs received by corrupted clients from the curator; the trusted curator evaluates a mechanism $\mathbf{M}(\mathbf{x})$ over all privately submitted client inputs ($\mathbf{x} = (x_1, ..., x_n)$) and returns private outputs to each client ($\mathbf{y} = (y_1, ..., y_n)$) resulting from the evaluation of $\mathbf{M}$. This stands in contrast to the classic setting of differential privacy where an analyst submits a query evaluated on a private database and the query result is not protected. Thus, our notion of round-differential privacy consists of two related properties.

Input-differential privacy (Def. 3) ensures that the privacy of submitted inputs in each interaction round with the trusted curator is protected. This implies that the sensitivity of the corrupted output distribution to any change in the honest input is bounded by chosen security parameters.

Correlated-output-differential privacy (Def. 7) protects honest outputs from being inferred by corrupted, correlated outputs; in many economic applications, all outputs are correlated, as they may represent an allocation of assets or resources; if the total aggregate funds output is known, an adversary corrupting $n-1$ clients can trivially infer the funds privately output to the single honest client by observing its own outputs.

Round-differential privacy permits trader to execute long-running strategies in a privacy-preserving manner. A common strategy is the Time-weighted Average Price (TWAP)[1] trade, where a larger trade volume is scheduled as smaller trades over time to minimize price impact. If the periodic execution of such smaller trades is detected early, the remaining trade schedule can be anticipated and front-run. Such leakage is prevented with round-differential privacy; each submitted trade and subsequent trade execution remains (differentially) private.

(2) Fuzzy order matching. We propose round-differentially private market mechanisms in the trusted curator model which improve on traditional dark pools designs by enabling the fair execution of long-running trade strategies. Below, we provide a high-level description applicable to both of our proposed fair market mechanisms DP-volume-match (Sections 4.1 and 5.1) and DP-double-auction (Sections 4.2 and 5.2);

Input phase: Traders privately submit limit orders[2] to the venue operator. Our auction

[1] `https://en.wikipedia.org/wiki/Time-weighted_average_price`
[2] In the DeFi setting, confidential ledger deposits accompany each trade order as in [4].

mechanism requires a market making *liquidity provider* to compensate for the noise added to trade matching.

Auction phase: A deterministic, optimal order matching is performed; such a matching will leak the inclusion or exclusion of a single trade request in the outputs and is not differentially private. The actual "trade", "no-trade" outcome for each order is determined by *sampling* a Bernoulli distribution biased towards the deterministically computed, optimal matching; we call this **fuzzy order matching**. However, since trades are filled or not filled based on independently sampling trade outcomes, there is no guarantee that each executed trade is matched with an equivalent volume in opposing direction; therefore, a liquidity deficit may occur. Here, market makers make up for liquidity deficits. To prevent market makers from learning about the traded volume of a single user (output privacy) from their updated liquidity balances, a random, yet bounded amount of market maker liquidity is *frozen* to obtain (ε, δ)-correlated-output differential privacy[3].

Output phase: Traders observe whether or not their order was fulfilled and market makers observe a noisy update to the liquidity balance. The trade output *distribution* is ε-indistinguishable with respect to the inclusion or exclusion of a single submitted order. The liquidity balance distribution is (ε, δ)-indistinguishable with respect to a single trade outcome.

(3) Practical fuzzy order matching in MPC. We propose custom MPC protocols for efficient execution of DP-volume-match (Sec. 5.1) and DP-double-auction (Sec. 5.2), and implement these in the Scale-Mamba framework [3] with Shamir Secret Sharing based MPC; history has shown that dark pool venue operators frequently exploit confidential order flow information [18, 19, 17], thus motivating us to demonstrate practical feasibility of distributing round-differential private market mechanisms across MPC committees in lieu of a trusted operator.

Finally, we emphasize that our fair market mechanism designs are applicable to both traditional dark pool venue operators and decentralized finance. Our fair markets can be instantiated in privacy-preserving smart contract frameworks realized by a MPC committee and privacy-preserving ledger, most recently demonstrated by Baum et al. in [4] with minimal complexity overhead; here, trade execution is settled in private on a public ledger.

1.1 Related Work

Differentially private markets. Chitra *et al.* [10] propose a *Uniform Random Execution* algorithm which permutes and splits submitted trades in a randomized manner. We note that [10] does not offer output or *post-trade* privacy; all executed trades are seen by the adversary. Thus, this approach does not contribute to our goal of protecting the privacy of long-running trader strategies performed over multiple rounds.

Dark pool markets. Recent proposals [6, 7, 11, 12] have convincingly demonstrated that the role distribution of the dark pool operator can be instantiated in practice with multi-party computation (MPC) to prevent abuse of private order information. Still, these works also do not consider the entirety of information flow leaking from all honest trader activity; (1) adversarial outputs reveal information about privately submitted honest inputs (Lemma 4) and (2) outputs are correlated, such that an adversary also obtains information about honest outputs (Lemma 6). In the decentralized finance setting, homomorphic encryption has been

[3] We note that the frozen liquidity is returned to the market makers following n auction periods, where n is sufficiently large to prevent front-running in practice.

proposed to aggregate orders obliviously [22]; however, since all inputs are encrypted to the same public key, any subsequent decryption to reveal the aggregated order will leak privacy of any single trade, if all but one client has been corrupted.

Differential privacy and MPC. Whilst differentially private mechanisms have been implemented in MPC, these works do not consider privacy over the full, individual transcript in the trusted curator model (§3), where clients submit private inputs and receive private outputs. Instead, the MPC output is a single query result computed over inputs from a private database. Here, the returned query is not considered private. The main use-case is generating differentially private machine learning models over private data with MPC [20, 1, 24, 21].

2 Preliminaries

Differential privacy. Differential privacy was introduced in [13] as a technique for quantifying the privacy guarantees of a mechanism. A central concept is the definition of neighbouring datasets which are denoted $x \sim x'$. Intuitively, this definition is used to capture the information we want to protect. Typically x and x' are identical except for the data about one individual. We formally define neighbouring inputs in our setting of the trusted curator in Section 3. Differential privacy is a restriction on how much the output distribution of a mechanism can change between any neighbouring input datasets.

▶ **Definition 1** ((ε, δ)-DP). *A randomized mechanism $\mathcal{M}$ satisfies (ε, δ)-differential privacy if for all pairs of neighbouring datasets $x \sim x'$ and all sets of outputs $\mathcal{S}$ we have:*

$$\Pr[\mathcal{M}(x) \in \mathcal{S}] \leq \exp(\varepsilon) \cdot \Pr[\mathcal{M}(x') \in \mathcal{S}] + \delta$$

Multi-party computation. Multi-party computation is a cryptographic technique that allows a set of n mistrustful parties to calculate a function of their own private inputs without revealing them. We consider an MPC protocol based on Shamir secret sharing, where a secret value s is shared by giving each party i the evaluation $f(i)$ of a polynomial f of degree t and coefficients in $\mathbb{F}_p$ such that $f(0) = s$. The protocol assumes a honest majority, i.e., $t < n/2$, and it is actively secure with a abort, meaning that a malicious party deviating from the protocol is caught with overwhelming probability and the honest parties abort the protocol when this happens. In this work, we use Scale-Mamba [3], a framework that implements various MPC protocols in the preprocessing model. In this methodology, the computation has a preprocessing phase where input independent data is generated. This data is then used in the input dependent online phase, where the desired computation over private inputs is performed.

3 Differential privacy in the trusted curator model

3.1 The trusted curator

We first define our proposed notions of privacy in the "trusted curator" model, which can then seamlessly be applied to the setting of secure multi-party computation. The trusted curator C interacts with parties $P_1, ..., P_n$, which are assumed to have established private, authenticated communication links with the trusted curator. Interaction proceeds in *rounds*, each consisting of the following phases.

1. **Input phase** All parties send their individual inputs to the trusted curator C, which obtains the input set $x_1, ..., x_n$ from parties $P_1, ..., P_n$ respectively.

2. **Evaluation phase** Upon receiving all inputs, the trusted curator locally computes a known algorithm M over inputs received in the input phase: namely, $\mathbf{y} \leftarrow \mathbf{M}(\mathbf{x})$ where $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{y} = (y_1, ..., y_n)$. Further, curator C is assumed to have access to to randomness to evaluate randomized algorithms.
3. **Output phase** The trusted curator privately sends each output element y_i in $\mathbf{y}$ to party P_i, and enters the input phase again.

Client corruption. The adversary $\mathcal{A}$ can statically corrupt up to $n-1$ clients, upon which it decides what inputs the corrupted clients submits in each interaction round. The adversary observes the output for each corrupted client returned from the trusted curator, but cannot corrupt the curator itself. We denote the adversarial output view from a round evaluating mechanism $\mathbf{M}$ on round inputs $\mathbf{x}$ as $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$.

Public outputs. We permit the trusted curator to also return public outputs; these are considered part of the adversarial output view $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$.

Privacy against the network adversary. We assume that the *physical presence* of a party in each round is observable by the network adversary. Since obfuscating the active participation across the network may be challenging, we assume parties to be physically online and to participate in each round, but permit them to submit **dummy inputs**, allowing for passive participation and obfuscating the *logical presence* of a party in a given round. Without dummy inputs, the physical presence of a party will always leak the presence of a logical input contributed by a party to the computation by the trusted curator; in the setting of privacy-preserving markets, for example, the network adversary would learn that a party is submitting some trade in a given round.

Further, we assume that parties can anonymously submit inputs to the trusted curator via techniques such as mixnets [9, 8], thereby hiding their identity from the network adversary. In practice, parties can delegate the physical interaction with the trusted curator model in each round to trusted servers, and only need to come online when they wish to forward a valid, non-dummy input.

3.2 Differential privacy for inputs

In the standard setting of differential privacy, an analyst performs a query on a private database and the result of the query is released to the analyst; a differentially private query bounds how much the analyst output distribution changes, upon editing an entry in the private database.

We adapt the classic notion of differential privacy to the setting of the trusted curator. Instead of protecting private database entries, we first wish to protect inputs submitted by honest clients. Thus, the following definition of neighbouring input vectors follows directly from the standard definition of neighbouring databases under the add/remove relation in the classic setting.

▶ **Definition 2 (neighboring input vectors).** *Let input vectors $\mathbf{x} = (x_1, ..., x_n)$ and $\mathbf{x}' = (x'_1, ..., x'_n)$ of equal length be such that the following holds true;*

$\exists i \in [n] : x_j = x'_j$ *for all* $j \neq i$

For a randomized algorithm $\mathbf{M}$ evaluated on input vector $\mathbf{x}$, let $\mathbf{M}^{\mathcal{A}}(\mathbf{x}) = \{Y_j\}_{j \in \mathcal{A}}$ denote output distributions observed by corrupted clients. Then, the following definition follows directly from the standard notion [14] of differential privacy where we consider the input vector as the private database on which the query $\mathbf{M}$ is performed and the adversary obtains the output view $\mathbf{M}^{\mathcal{A}}(\mathbf{x})$ of all corrupted parties.

▶ **Definition 3** (**(ε, δ)-input-DP**). *For an evaluation of (ε, δ)-input differentially private algorithm $\mathbf{M}$ in the trusted curator model over neighboring private input vectors $\mathbf{x} \sim \mathbf{x}'$, the following must hold for any adversarially observable output event $\mathcal{S}^{\mathcal{A}}$.*

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}') \in \mathcal{S}^{\mathcal{A}} \,] + \delta$$

As we will see in Section 3.3, input-differential privacy is necessary, but insufficient to protect both in- and output of an honest client in the trusted curator round. Whilst Definition 3 protects the privacy of a user input, it does not not guarantee that the honest output remains private. This motivates *correlated-output* differential privacy, introduced in subsequent section Section 3.3. Again, the standard setting of differential privacy does not consider the privacy of the query output, as there is only a single query result which is released publicly or to the adversarial analyst.

▶ **Lemma 4.** *Dark Pools violate (ε, δ)-input differential privacy.*

Proof. (Sketch) A dark pool venue operator can be idealized as a trusted curator which privately receives trade orders from clients. Upon evaluating the market algorithm in private, it privately outputs trade executions to clients. Assume the corrupted client submitting a sell order observes that its trade order is executed. Any change in the honest counter-party's privately submitted buy order may cancel the matching of this order pair, observable to adversary with probability 1, thereby violating Definition 3. ◀

Adversarially chosen inputs Note that input differential privacy in Definition 3 naturally protects against *chosen input attacks*; informally, such an attack permits the adversary to change its inputs and observe induced effects on its output distributions to learn something about honest inputs. However, note that (ε, δ)-input-DP applies equal privacy guarantees to *any input* submitted to the trusted curator. Thus, for an appropriately chosen privacy parameters, a chosen input attack on (ε, δ)-input-DP mechanism will not reveal meaningful information to the adversary, as its chosen input perturbation will not induce a sufficiently observable effect on its output distributions.

3.3 Differential privacy for correlated outputs

In contrast to prior work, where a single output is returned from a differentially-private mechanism, we must protect the privacy of outputs that are returned from trusted curator model to individual clients over private channels. Even if input-differential privacy protects honest inputs, the individual outputs returned to clients may still be strongly correlated, potentially allowing honest outputs to be inferred from corrupted ones.

▶ **Definition 5** (**(ε, δ)-correlated-output-DP**). *For an evaluation of (ε, δ)-correlated output differentially private algorithm $\mathbf{M}$ in the trusted curator model over fixed input vector $\mathbf{x}$, the following must hold for any adversarial output event $\mathcal{S}^{\mathcal{A}}$ and any honest output event $\mathcal{S}^{h}$.*

$$\Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^{h}(\mathbf{x}) \in \mathcal{S}^{h} \,] \leq \exp(\varepsilon) \cdot \Pr[\, \mathbf{M}^{\mathcal{A}}(\mathbf{x}) \in \mathcal{S}^{\mathcal{A}} \mid \mathbf{M}^{h}(\mathbf{x}) \notin \mathcal{S}^{h} \,] + \delta$$

Definition 5 is interpreted as follows; for any set of inputs and two different honest output events ($\mathbf{M}^{i}(\mathbf{x}) \in \mathcal{S}^{i}$ vs. $\mathbf{M}^{i}(\mathbf{x}) \notin \mathcal{S}^{i}$), the output distribution of the adversary remains (ε, δ)-similar. In other words, any change in the honest output can only have a bounded effect on the adversarially observable output.

We highlight an immediate consequence of Definition 5 for economic applications; a correlated-output-DP mechanism cannot distribute funds to all clients where the sum of output funds is known or public; an adversary corrupting $n-1$ clients can trivially infer the funds privately output to the single honest client by just observing its own outputs. Thus;

▶ **Lemma 6.** *Economic mechanisms evaluated in the trusted curator model which allocate a fixed supply of "assets" over client outputs violate (ε, δ)-correlated output differential privacy.*

Overcoming this is not straight-forwards, as a financial application cannot be allowed to arbitrarily mint or create funds out of thin air. We overcome these constraints by performing *fuzzy matching* of orders and temporarily freezing funds to achieve correlated-output differential privacy in DP-volume-match (Section 4.1) and DP-double-auction (Section 4.2).

Applications with correlated outputs. We argue there exist many applications in the trusted curator setting which require correlated outputs; most related to this work are economic applications which govern the private allocation of finite resources, which include auctions, markets, financial derivatives and other economic contracts.

3.4 Single-round & Multi-round privacy

Since (ε, δ)-input-DP and (ε, δ)-correlated-output-DP protect different parts of the honest round transcript, we must formally consider two separate privacy budgets which are consumed with each interaction round in the trusted curator model. We define differential privacy for *each interaction round* with the trusted curator as follows.

▶ **Definition 7 (Round-DP).** *The evaluation of a mechanism that satisfies $(\varepsilon^{in}, \delta^{in})$-input-DP and $(\varepsilon^{out}, \delta^{out})$-correlated-output-DP is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-round differentially private.*

Definition 7 implies that the privacy of input and outputs may be parameterized *independently*. Indeed, this permits the trade-off between utility and privacy for different parts of the honest transcript to be decided separately; the input to an evaluation round may require a higher degree of privacy than the returned output or vice versa.

Multi-round privacy In standard differential privacy protecting a private database, m instances of (ε, δ)-differentially private queries taken together are $(\varepsilon_1 + \ldots + \varepsilon_m, \delta_1 + \ldots + \delta_m)$ differentially private. However, in the trusted curator model, the curator accepts fresh inputs in each interaction round, allowing us to consider each round input as disjoint, private data. We define multi-round-differential privacy as the sensitivity of the adversarial output view over m rounds to a change in a single input of a single round $r \in [m]$.

▶ **Definition 8 (m-round-DP).** *Let the adversarial output view over m interaction rounds between n-clients and the trusted curator be given as $\mathbf{M}_1^{\mathcal{A}}(\mathbf{x}_1), \ldots, \mathbf{M}_m^{\mathcal{A}}(\mathbf{x}_m)$. Then, we define this m-round transcript as;*

$$\mathbf{M}_{mul}^{\mathcal{A}}(\bar{\mathbf{x}}) = (\mathbf{M}_1^{\mathcal{A}}(\mathbf{x}_1), \ldots, \mathbf{M}_m^{\mathcal{A}}(\mathbf{x}_m)) \quad \textit{where} \quad \bar{\mathbf{x}} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$$

Multi-round inputs $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$ are neighboring, if there exists unique round $j \in [m]$, such that $\mathbf{x}_j \in \bar{\mathbf{x}}$ and $\mathbf{x}'_j \in \bar{\mathbf{x}}'$ are neighbouring (Definition 2) and $\mathbf{x}_k = \mathbf{x}'_k$ for all other rounds $k \neq j$. Then, let we denote an m-round output event for the adversarial and honest client as $\mathcal{S}_{mul}^{\mathcal{A}} = \mathcal{S}_1^{\mathcal{A}}, \ldots, \mathcal{S}_m^{\mathcal{A}}$ and $\mathcal{S}_{mul}^{h} = \mathcal{S}_1^{h}, \ldots, \mathcal{S}_m^{h}$ respectively.

The m-round interaction is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-$m$-round differentially private if for neighbouring $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$, any adversarial m-round event $\mathcal{S}_{mul}^{\mathcal{A}}$ and honest m-round event $\mathcal{S}_{mul}^{h}$, the *following holds;*

$$\begin{aligned}
&\Pr[\, \mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul} \,] \\
\leq \exp(\varepsilon^{in}) \cdot &\Pr[\, \mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}') \in \mathcal{S}^{\mathcal{A}}_{mul} \,] + \delta^{in} \qquad (a) \\
&\Pr[\, \mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul} \mid \mathbf{M}^{h}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{h}_{mul} \,] \\
\leq \exp(\varepsilon^{out}) \cdot &\Pr[\, \mathbf{M}^{\mathcal{A}}_{mul}(\bar{\mathbf{x}}) \in \mathcal{S}^{\mathcal{A}}_{mul} \mid \mathbf{M}^{h}_{mul}(\bar{\mathbf{x}}) \notin \mathcal{S}^{h}_{mul} \,] + \delta^{out} \qquad (b)
\end{aligned}$$

Concretely, m-round-DP bounds the sensitivity of the adversarial output distribution over m-rounds to both (a) a change in the honest users input in the round $j \in [m]$ round and (b) a change in the honest users output in round $j \in [m]$.

The following theorem relates single-round-DP (def. 7) with m-round-DP (def. 8).

▶ **Theorem 9** (**m-round composition**). *Let there be m consecutive interaction rounds with n clients and the trusted curator. In each round, the trusted curator evaluate round-specific algorithms $\mathbf{M}_1, ..., \mathbf{M}_m$ that are $(\varepsilon^{in}_1, \delta^{in}_1)$-$(\varepsilon^{out}_1, \delta^{out}_1)$, ... , $(\varepsilon^{in}_m, \delta^{in}_m)$-$(\varepsilon^{out}_m, \delta^{out}_m)$ round differentially private and evaluated on round-specific input vectors $\mathbf{x}_1, \dots, \mathbf{x}_m$, each consisting of $|\mathbf{x}_j| = n$ elements. Then, the m-round evaluation is*

$$\big(\max_{j\in[m]} \varepsilon^{in}_j \ , \ \max_{j\in[m]} \delta^{in}_j\big) - \big(\max_{j\in[m]} \varepsilon^{out}_j \ , \ \max_{j\in[m]} \delta^{out}_j\big)$$

m-round differentially private.

Proof. (Theorem 9) In an m-round interaction with n clients and the trusted curator evaluating round-specific algorithms $\mathbf{M}_1, ..., \mathbf{M}_m$, clients provide fresh set of inputs $\mathbf{x}_j$ in each round $j \in [m]$. Definition 8 bounds the sensitivity of the full, adversarial output over m-rounds to any change in (a) honest input or (b) honest output in a single round $j \in [m]$. However, each evaluation of $\mathbf{M}_i$ is independent; in each round, the curator only computes on freshly submitted inputs only. Thus, any change in adversarial output distribution induced by (a) or (b) will be only be observable in round j of the adversarial transcript.

Since $\mathbf{M}_j$ for round $j \in [m]$ is $(\varepsilon^{in}_j, \delta^{in}_j)$-$(\varepsilon^{out}_j, \delta^{out}_j)$-round differentially private, it follows that each algorithm $\mathbf{M}_j$ for any round $j \in [m]$ is also

$$(\varepsilon^{in}_{max}, \delta^{in}_{max}) - (\varepsilon^{out}_{max}, \delta^{out}_{max}) = \big(\max_{j\in[m]} \varepsilon^{in}_j \ , \ \max_{j\in[m]} \delta^{in}_j\big) - \big(\max_{j\in[m]} \varepsilon^{out}_j \ , \ \max_{j\in[m]} \delta^{out}_j\big)$$

round differentially private. Then, m-round differentially privacy (Definition 8) is satisfied, since a change to (a) an honest input or (b) output in round $j \in [m]$ will induce an (a) $(\varepsilon^{in}_{max}, \delta^{in}_{max})$ or (b) $(\varepsilon^{out}_{max}, \delta^{out}_{max})$ bounded effect on adversarial output $\mathbf{M}^{\mathcal{A}}_j$ in round j only. ◀

4 Fuzzy Order Matching

We propose round-differentially-private, periodic market mechanisms in the trusted curator model for (1) volume matching of orders, where a batch of buy and sell orders are matched at a given exchange rate determined at an external reference market, and (2) double auctions, where buy and sell orders also feature price limits, such that a clearing price must first be computed for each round before orders can be matched. Following a gentle introduction, each algorithm is formally proven to satisfy round-differential-privacy introduced in Definition 7.

To realize any meaningful notion of privacy in practice, we distribute the trusted curator by means of secure multi-party computation (MPC). The overhead added the use of MPC can, however, result in a prohibitively low order throughput when evaluating a naively built algorithm. Thus, to obtain market mechanisms with efficiency suitable for real-world use, our proposed MPC algorithms in Section 5 are designed to avoid expensive MPC operations.

4.1 Differentially Private Volume Matching

In volume matching, the exchange rate is pre-determined by an external reference rate. A trader only chooses to submit a sell, buy or to abstain from the round with a dummy order.

We introduce a $(\varepsilon^{\mathsf{in}}, 0)$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$ round-differentially private volume matching algorithm named **DP-Volume-match**, overcoming the privacy limitations of the traditional dark pool setting, where a matching of buy and sell orders implies leaking an order execution to the counter-party trade and violating both input- and correlated-output differential privacy (Lemmas 4 and 6). We overcome this privacy hurdle in classical order matching with two randomized sub-routines;

[1] Fuzzy order matching. In the first phase of DP-volume-match, orders are matched in a "fuzzy" manner; following a preliminary, deterministic matching step which maximizes number of trades, each final trade output (trade/no-trade) is sampled independently from a distribution parameterized by $\varepsilon^{\mathsf{in}}$ and biased towards the preliminary matching result; we adapt this technique from the standard randomized response mechanism [14], that is both $(\varepsilon^{\mathsf{in}}, 0)$-input and $(0, 0)$-correlated output differentially private; the latter property arises naturally from the independent sampling of outputs which occurs in randomized response. Randomized, fuzzy matching of orders also implies that the final aggregate exchange of tokens may not sum to zero; in any given round, the total buy volume may not equal the total sell volume.

[1] Liquidity compensation. We therefore introduce a **liquidity provider**, which compensates for the in mismatch between buy and sell volume; to ensure that the (corrupt) liquidity provider's output is correlated-output differentially private, a randomized amount of its liquidity is frozen; the differential privacy guarantees of DP-volume-match hold over m rounds until the frozen funds are returned to the liquidity provider. We argue that in practice, it is acceptable to guarantee multi-round privacy for a bounded number of rounds. We detail the different steps of DP-volume-match next and refer to the full algorithm implemented in MPC in Figure 3 for details.

1. Fuzzy order matching
   a. Deterministic matching
   b. Randomized response over matches
2. Liquidity compensation
   a. Liquidity compensation for sampled trades
   b. Randomized liquidity freezing

DP-Volume-match orders. Let a valid, privately submitted trade order be the tuple (b, s, id), where b and s represent buy and sell bits respectively, and id is the trader identifier. Thus, let $(b, s) \in [(1, 0), (0, 1), (0, 0)]$ represent a buy, sell and dummy order respectively. We fix buy and sell unit volumes such that a single sell and buy order always match.

[1a] Deterministic matching (L2 in Figure 3). Let the number of orders sent to the trusted curator by n clients be $\mathbf{x} = \{(b, s, \mathsf{id}_1), ..., (b, s, \mathsf{id}_n)\}$. Then, the maximum possible number of matches between buy $(b, s) = (1, 0)$ and sell $(b, s) = (0, 1)$ orders is computed, which is simply the smaller of the number of buy b and sell s orders. Let the result of the deterministic matching phase be the bit array $\mathsf{match} = (\mathsf{match}_1, ..., \mathsf{match}_n)$, where bit match_i indicates if the i'th submitted order was matched (1) or not (0). Once the total number of preliminary matched pairs is computed, they are assigned randomly to the non-dummy orders; dummy orders are never matched.

[1b] Randomized response over matches (L5 in Figure 3). Here, we apply the standard randomized response mechanism [14] to determine whether a *trade* or *no-trade* is returned to the trader who submitted a valid trade order; for each bit in array match where $\mathsf{match}_i = 1$, the probability of the final trade_i bit equaling 1 or 0 is given by;

$$\begin{aligned}
&\Pr[\,\mathsf{trade}_i = 1 \,|\, \mathsf{match}_i = 1\,] = e^{\varepsilon}(1+e^{\varepsilon})^{-1} \qquad (1)\\
&\Pr[\,\mathsf{trade}_i = 0 \,|\, \mathsf{match}_i = 1\,] = (1+e^{\varepsilon})^{-1}
\end{aligned}$$

Conversely, for each bit $\mathsf{match}_i = 0$ in match and party i did not submit a dummy order, the probability of the final trade_i outcome being sampled as 1 or 0 is given by;

$$\begin{aligned}
&\Pr[\,\mathsf{trade}_i = 1 \,|\, \mathsf{match}_i = 0\,] = (1+e^{\varepsilon})^{-1} \qquad (2)\\
&\Pr[\,\mathsf{trade}_i = 0 \,|\, \mathsf{match}_i = 0\,] = e^{\varepsilon}(1+e^{\varepsilon})^{-1}
\end{aligned}$$

Thus, for parties submitting valid, non-dummy trades, the final trading results in array $\mathsf{trade} = [\mathsf{trade}_1, ..., \mathsf{trade}_n]$ is obtained from independently sampling from distributions Eq. 1 or 2 according to the match array from the deterministic matching subroutine [1a].

Trader outputs are given by the array $\mathbf{y} = [(b_1^{\mathsf{out}}, s_1^{\mathsf{out}}, \mathsf{id}_1), ..., (b_n^{\mathsf{out}}, s_n^{\mathsf{out}}, \mathsf{id}_n)]$, where each entry $((b_i^{\mathsf{out}}, s_i^{\mathsf{out}}, \mathsf{id}_i))$ returned to party i indicates whether a buy $((b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (1,0))$, sell $((b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (0,1))$ or no trade $((b_i^{\mathsf{out}}, s_i^{\mathsf{out}}) = (0,0))$ was executed; $\mathbf{y}$ is determined from input vector $\mathbf{x}$ and final trade outcome vector $\mathsf{trade} = [\mathsf{trade}_1, ..., \mathsf{trade}_n]$ from step [2b],

We emphasize that a trade can only be executed if a non-dummy order was submitted at the beginning of the round, and in the same direction (sell or buy) as intended by the trader. Dummy orders always return $((b^{\mathsf{out}}, s^{\mathsf{out}}) = (0,0))$ as output; the fuzzy matching is only applied to valid, non-dummy orders only, and thus the trading "interface" remains the same as in traditional volume matching algorithms.

[2a] Liquidity compensation for sampled trades (L6 in Figure 3). Fuzzy matching of orders via randomized response implies that trades output from step [1b] in DP-volume-match do not match precisely; for the output vector $\mathbf{y} = [(b_1^{\mathsf{out}}, s_1^{\mathsf{out}}, \mathsf{id}_1), ..., (b_n^{\mathsf{out}}, s_n^{\mathsf{out}}, \mathsf{id}_n)]$, the following can occur;

$$\sum_{i\in[n]} s_i^{\mathsf{out}} \neq \sum_{i\in[n]} b_i^{\mathsf{out}}$$

Since sells and buys may not cancel out, we introduce the presence of a **liquidity provider**, which compensates for this mismatch in traded liquidity. Then, the amount of the numeraire asset (Δ_0) and risky asset (Δ_1) provided $(\Delta_{j\in\{0,1\}} < 0)$ or received $(\Delta_{j\in\{0,1\}} > 0)$ by the liquidity provider is given as;

$$\Delta_0 = -\big(\sum_{i\in[n]} s_i^{\mathsf{out}} - b_i^{\mathsf{out}}\big) \qquad \Delta_1 = \big(\sum_{i\in[n]} s_i^{\mathsf{out}} - b_i^{\mathsf{out}}\big) \qquad (3)$$

The liquidity providers compensates for this liquidity imbalance resulting from fuzzy matching; its initial balances $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$ are updated to $(x_0^{\mathsf{liq}} + \Delta_0, x_1^{\mathsf{liq}} + \Delta_1)$; however, note that any change in the honest user's trade execution will affect Δ_0, Δ_1 with probability 1, observable by the corrupted liquidity provider and violating correlated-output differential privacy (Definition 5); *relaxing* the correlation between the final exchange of assets and the update in funds observed by the liquidity provider can only imply the *minting* or *removal* of funds in the round outputs.

We propose a compromise, which is a randomized mechanism to *freeze liquidity*, protecting the privacy of traders for the m-round duration that the liquidity remains frozen. Our algorithm DP-volume-match refrains from minting, preserving the integrity of the underlying asset types.

[2b] Randomized liquidity freezing (L8 in Figure 3). The liquidity provider inputs reserves $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$ of numeraire and risky asset pair (τ_0, τ_1) to a given round, and is returned updated reserve balances $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = (x_0^{\mathsf{liq}} + \Delta_0 - \rho_0, x_1^{\mathsf{liq}} + \Delta_1 - \rho_1)$, where (ρ_0, ρ_1) is the volume of (τ_0, τ_1) frozen in the given round. In practice, we assume that frozen amounts are eventually returned to the liquidity provider after m rounds, where m is chosen to be sufficiently long to protect a trading strategy executed over multiple rounds.

We describe the distribution from which (ρ_0, ρ_1) are sampled, given by probability mass function P_{frz} in Equation (4); this distribution is parameterized by a maximum amount of frozen liquidity $\rho_t^{\mathsf{max}} \geq 1$ for $t \in \{0, 1\}$ in the round, and correlated-output differential privacy parameters $\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}}$.

$$\forall t \in \{0,1\}: \tag{4}$$

$$P_{\mathsf{frz}}(\rho_t) = \begin{cases} (1-\sqrt{1-\delta})/2 \cdot \exp(\varepsilon^{\mathsf{out}} \cdot \rho_t) & \rho_t \in [\,0 : \lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil\,] \\ (1-\sqrt{1-\delta})/2 \cdot \exp(\varepsilon^{\mathsf{out}} \cdot (\rho_t^{\mathsf{max}} - \rho_t)) & \rho_t \in [\,\lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil + 1 : \rho^{\mathsf{max}}\,] \\ 0 & \text{otherwise} \end{cases}$$

The sensitivity of $\rho_0 + \Delta_0$ and $\rho_1 + \Delta_1$ to the execution of a single trade is ± 1. Distribution frz allocates probability mass across multiples of unit trade volume; neighbouring freezing events ρ_t and $\rho_t \pm 1$ are allocated probabilities which differ by factor $\exp(\varepsilon^{\mathsf{out}})$. Since we limit the amount of frozen tokens to the range $[0 : \rho_t^{\mathsf{max}}]$, we must accept a non-zero δ^{out} probability of violating $(\varepsilon^{\mathsf{out}})$-correlated-output differential privacy (See Lemma 12).

Parameterization of P_{frz}. The freeze distribution is paramaterized by $(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$ and ρ_0^{max} and ρ^{max}, but we note that these parameters cannot be chosen independently. For total probability mass in Equation (4) must sum to 1, parameters $(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$ and ρ_t^{max} must satisfy the following relation, which follows directly from the aggregate probability mass of P_{frz}.

$$\sum_{\rho_t} P_{\mathsf{frz}}(\rho_t) = \sum_{\rho_t \in \mathsf{L}} (1-\sqrt{1-\delta})/2 \cdot \exp(\varepsilon \cdot \rho_t) + \sum_{\rho_t \in \mathsf{R}} (1-\sqrt{1-\delta})/2 \cdot \exp(\varepsilon \cdot (\rho_{\mathsf{max}} - \rho_t)) = 1$$

$$\text{where} \quad \mathsf{L} = [\,0 : \lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil\,] \quad \mathsf{R} = [\,\lceil \frac{\rho^{\mathsf{max}}-1}{2} \rceil + 1 : \rho^{\mathsf{max}}\,]$$

▶ **Theorem 10.** *DP-Volume-matching is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-$m$-round differentially private.*

Proof. (Theorem 10) Follows from Lemma 11 and Lemma 12, stated and proven below, for the m rounds during which frozen liquidity provider reserves are secretly frozen. ◀

▶ **Lemma 11.** *DP-Volume-matching is (ε, δ)-input differentially private.*

Proof. (Lemma 11) Let $\mathbf{x}, \mathbf{x}'$ be neighboring vectors of trade orders (Definition 2) submitted to the DP-volume matching algorithm. Note that the number of submitted orders n is always even, thus unique pairs can be formed for all submitted orders, such that some pairs will be in opposing directions and matched during the [1a] determinstic matching phase; for such a pair at index pair (h, j), where $h \neq j$, the matching bits output from [1a] for trade orders

h, j will be $\mathsf{match}_h = \mathsf{match}_j = 1$. Since they are in opposing directions, $(b_i, s_i) = (1, 0)$, and $(b_j, s_j) = (0, 1)$.

We observe that any change in an input can, in the worst case, increment, or decrement the number of matched order pairs in opposing direction output from [1a] of DP-volume match; in case (a), the honest user changes the direction of its trade order, thus potentially leaving the previously matched order unmatched. in case (b) the honest user changes its order to or from a dummy-order. Here, the impact on the counter-party order is the same. Thus, the sensitivity of the deterministic matching [1a] outcome $\mathsf{match} = [\mathsf{match}_1, ..., \mathsf{match}_n]$ to a change from $\mathbf{x}$ to $\mathbf{x}'$ is a flip in both bits of pair match_h and match_j $(h \neq j)$. We argue input-differential privacy for transcript of traders and liquidity provider separately.

For the adversarial **trader output view**, we must argue that the probability of any trade outcome is at most (ε, δ)-sensitive to the change in the honest traders order; note, that the adversary corrupting all but the honest trader will observe a change in output distribution for one of the corrupted parties; let this party index be j. Then, recall, a flip in the deterministic matching outcome match_j from [1a] implies a change in distribution (eq. 1, 2) from which trade_j is sampled in randomized response step [1b] of DP-volume match. Thus, the following holds for the trade output view of the adversary;

$$\prod_{i \in \mathcal{A}} \frac{\Pr[\, \mathsf{trade}_i = b_i \mid \mathbf{x}\,]}{\Pr[\, \mathsf{trade}_i = b_i \mid \mathbf{x}'\,]} = \frac{\Pr[\, \mathsf{trade}_j = b_j \mid \mathbf{x}\,]}{\Pr[\, \mathsf{trade}_j = b_j \mid \mathbf{x}'\,]} = \frac{e^{\varepsilon} \cdot (1 + e^{\varepsilon})}{(1 + e^{\varepsilon})} = e^{\varepsilon}$$

Here, the first equality holds, since only trade_j at a single adversarial index is affected by the honest input change; all other corrupted outputs observe the same distribution. The second equality follows directly from eq. 1 & 2.

For the adversarial **liquidity provider output view**, it suffices to argue that liquidity mismatch (Δ_0, Δ_1) defined in eq. 3 and determined in step [2a] of DP-volume-match preserves (ε, δ)-input differential privacy; any post-processing preserves this property, including the liquidity freezing sub-routine [2b] in DP-volume match.

Consider our pair $(\mathsf{match}_h, \mathsf{match}_j)$ again, which sees both bits flipped upon a change in the single change in input $(\mathbf{x} \to \mathbf{x}')$. Recall that Δ_0, Δ_1 indicate the liquidity required to compensate for unmatched orders in $\mathsf{trade} = [\mathsf{trade}_1, ..., \mathsf{trade}_n]$ sampled in step [1b] of DP-volume match. Δ_0 and Δ_1 are incremented/decremented when an (un)matched pair $(\mathsf{match}_h, \mathsf{match}_j) \in \{(1,1), (0,0)\}$ in [1a] obtains a sampled outcome in [1b] such that only one of the two orders is executed, e.g. $(\mathsf{trade}_h, \mathsf{trade}_j) \in \{(0,1), (1,0)\}$. Δ_0 and Δ_1 are not incremented/decremented in the case of $(\mathsf{trade}_h, \mathsf{trade}_j) \in \{(0,0), (1,1)\}$, as no additional liquidity imbalance is contributed by order pair (h, j). Thus, we are interested in events where order pair (h, j) do (not) contribute to the liquidity imbalance Δ_0, Δ_1.

The event probabilities of $(\mathsf{trade}_h, \mathsf{trade}_j)$ sampled in [1b] conditioned on input vector $\mathbf{x}$, which induces $(\mathsf{match}_h, \mathsf{match}_j) = (1, 1)$ in step [1a] are given by,

$$\begin{aligned}
&\Pr[\, (\mathsf{trade}_i, \mathsf{trade}_j) = (0,0) \mid \mathbf{x}\,] = (1 + e^{\varepsilon})^{-1}(1 + e^{\varepsilon})^{-1} \\
&\Pr[\, (\mathsf{trade}_i, \mathsf{trade}_j) = (0,1) \mid \mathbf{x}\,] = (1 + e^{\varepsilon})^{-1} e^{\varepsilon}(1 + e^{\varepsilon})^{-1} \\
&\Pr[\, (\mathsf{trade}_i, \mathsf{trade}_j) = (1,0) \mid \mathbf{x}\,] = e^{\varepsilon}(1 + e^{\varepsilon})^{-1}(1 + e^{\varepsilon})^{-1} \\
&\Pr[\, (\mathsf{trade}_i, \mathsf{trade}_j) = (1,1) \mid \mathbf{x}\,] = e^{\varepsilon}(1 + e^{\varepsilon})^{-1} e^{\varepsilon}(1 + e^{\varepsilon})^{-1}
\end{aligned}$$

and the probabilities of the same events conditioned on input vector $\mathbf{x}'$, which induces

$(\mathsf{match}_h, \mathsf{match}_j) = (0,0)$ in step [1a] are given by

$$\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (0,0) \mid \mathbf{x}'\,] = e^{\varepsilon}(1+e^{\varepsilon})^{-1} e^{\varepsilon}(1+e^{\varepsilon})^{-1}$$
$$\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (0,1) \mid \mathbf{x}'\,] = e^{\varepsilon}(1+e^{\varepsilon})^{-1}(1+e^{\varepsilon})^{-1}$$
$$\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (1,0) \mid \mathbf{x}'\,] = (1+e^{\varepsilon})^{-1} e^{\varepsilon}(1+e^{\varepsilon})^{-1}$$
$$\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (1,1) \mid \mathbf{x}'\,] = (1+e^{\varepsilon})^{-1}(1+e^{\varepsilon})^{-1}$$

Thus, the event probability where pair (h, j) contributes no liquidity imbalance to Δ_0, Δ_1 is not sensitive to an change in input.

$$\frac{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) \in \{(1,1),(0,0)\} \mid \mathbf{x}\,]}{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) \in \{(1,1),(0,0)\} \mid \mathbf{x}'\,]} = 1 \leq \exp(\varepsilon)$$

Further, the event probability where pair (h, j) does contribute liquidity imbalance in a specific direction to Δ_0, Δ_1 is also not sensitive to a change in input.

$$\frac{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (1,0) \mid \mathbf{x}\,]}{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (1,0) \mid \mathbf{x}'\,]} = \frac{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (0,1) \mid \mathbf{x}\,]}{\Pr[\,(\mathsf{trade}_i, \mathsf{trade}_j) = (0,1) \mid \mathbf{x}'\,]} = 1 \leq \exp(\varepsilon)$$

◀

▶ **Lemma 12.** *DP-Volume-matching is ε, δ-correlated-output differentially private.*

Proof. (Lemma 12) As per Definition 5, we must demonstrate that adversary output event probabilities is ε, δ-sensitive to a change in the honest user's output; the adversarial output view is composed of corrupted trader outputs and the view of the corrupted liquidity provider.

Note that inputs are fixed in correlated-output differential privacy. Thus, the output $\mathsf{match} = [\mathsf{match}_1, ..., \mathsf{match}_n]$ of deterministic matching in step [1a] remains unaffected; the distribution of trader outputs also remain unchanged in step [1b]. Correlated-output differential privacy holds trivially for the adversarial **trader output view**.

The **corrupted liquidity provider** provides reserves $(x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}})$ to compensate for liquidity mismatch (Δ_0, Δ_1 in eq. 3) resulting from randomized response in step [1b]. The sensitivity of Δ_0, Δ_1 to a change in the honest users trade output sampled in the randomized response step [1b] of DP-volume match is $|1|$ for both Δ_0 and Δ_1, as the trade outcome for pair $(\mathsf{trade}_h, \mathsf{trade}_j) \in \{(1,1),(0,0)\}$ is changed to $(\mathsf{trade}_h, \mathsf{trade}_j) \in \{(1,0),(0,1)\}$ or vice versa.

Consider the output event that the liquidity provider observes, namely that its reserves are updated to $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = \big(x_0^{\mathsf{liq}} + \Delta_0 - \rho_0\ ,\ x_1^{\mathsf{liq}} + \Delta_1 - \rho_1\big)$ following the liquidity freezing step [2b] of DP-volume-match. We analyze how the probability of a given event or reserve update observed by the liquidity provider is affected by the change of the honest output, which induces $(\Delta_0, \Delta) \to (\Delta_0 \pm 1, \Delta \mp 1)$ in step [2a]. If the liquidity amount frozen is in step [2b] is incremented/decremented in the opposing direction, namely $(\rho_0, \rho_1) \to (\rho_0 \mp 1, \rho_1 \pm 1)$, the observed reserve update remains unchanged. We express following two events which are indistinguishable by the adversary;

(a) $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = \big(x_0^{\mathsf{liq}} + \Delta_0 - \rho_0\ ,\ x_1^{\mathsf{liq}} + \Delta_1 - \rho_1\big)$
(b) $(y_0^{\mathsf{liq}}, y_1^{\mathsf{liq}}) = \big(x_0^{\mathsf{liq}} + (\Delta_0 \pm 1) - (\rho_0 \mp 1)\ ,\ x_1^{\mathsf{liq}} + (\Delta_1 \mp 1) - (\rho_0 \pm 1)\big)$

Then, we relate the probabilities of these two events for liquidity freezing range $(0 : \rho_{\mathsf{max}})$;

$$\forall \rho_0, \rho_1 \in (0 : \rho_{\mathsf{max}}) : \tag{5}$$

$$\frac{\Pr[\,\big(x_0^{\mathsf{liq}} + \Delta_0 - \rho_0\ ,\ x_1^{\mathsf{liq}} + \Delta_1 - \rho_1\big)\,]}{\Pr[\,\big(x_0^{\mathsf{liq}} + (\Delta_0 \pm 1) - (\rho_0 \mp 1)\ ,\ x_1^{\mathsf{liq}} + (\Delta_1 \mp 1) - (\rho_0 \pm 1)\big)\,]} \leq \exp(\varepsilon) \tag{6}$$

The inequality holds, because sampling probabilities for ρ_t and $\rho_t \pm 1$ differs by e^{ε}, as defined in P_{frz} (eq. 4). However, it does not hold for $\rho_t \in \{0, \rho_t^{\mathsf{max}}\}$; here, the probability of sampling $\rho_t \pm 1$ can be 0, violating correlated-output differential privacy. Concretely, observe for P_{frz} in eq. 4, $\Pr[\rho_t = 0 - 1] = \Pr[\rho_t = \mathsf{max} + 1] = 0$. We show that the probability of that $\rho_t \in \{0, \rho_{\mathsf{max}}\}$ for any $t \in \{0, 1\}$ is bounded by δ.

Consider the probability that $\rho_t \in \{0, \rho^{\mathsf{max}}\}$ is sampled is $P_{\mathsf{frz}}(0) + P_{\mathsf{frz}}(\rho_{\mathsf{max}})$ (eq. 4). Then, the probability that neither of the sampled frozen amounts equals $\{0, \rho^{\mathsf{max}}\}$ is given below;

$$\begin{aligned}
\Pr[\,\rho_0, \rho_1 \notin \{0, \rho^{\mathsf{max}}\}\,] &= \big(1 - (P_{\mathsf{frz}}(0) + P_{\mathsf{frz}}(\rho_{\mathsf{max}}))\big)^2 \\
&= \big(1 - (\frac{1 - \sqrt{1-\delta}}{2} + \frac{1 - \sqrt{1-\delta}}{2})\big)^2 \\
&= \big(1 - (1 - \sqrt{1-\delta})\big)^2 \\
&= \big(\sqrt{1-\delta}\big)^2 = 1 - \delta
\end{aligned} \tag{7}$$

◀

4.2 Differentially Private Double Auctions

We propose a $(\varepsilon^{\mathsf{in}}, 0)$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-round differentially private double auction algorithm, called **DP-double-auction** for the trusted curator setting. Here, we introduce an initial sub-routine to compute a $(\varepsilon^{\mathsf{in}}, 0)$-input-differentially private **clearing price** from trade orders input to the round. Subsequently, the DP-volume-match (§4.1) is performed on the subset of trade orders with price limits consistent with the clearing price.

For each round, we assume a discrete price range $\mathbf{r} = [r_1, ..., r_l]$; here, the discrete price maximizing the number of order matches is initially selected; to preserve input differential privacy, a subsequent exponential mechanism is applied to determine the publicly output clearing price.

DP-double-auction orders. Inputs submitted to DP-double-auction are in form $\mathbf{x} = [(\mathbf{w}_1, \mathsf{dir}_1, \mathsf{id}_1), ..., (\mathbf{w}_n, \mathsf{dir}_n, \mathsf{id}_n)]$, where each valid trade order $(\mathbf{w}_i, \mathsf{dir}_i, \mathsf{id}_i)$, contains a bit w_{ij} in array $\mathbf{w}_i = [w_{i1}, ..., w_{il}]$ that indicates whether user i is willing to buy ($\mathsf{dir}_i = 0$) or sell ($\mathsf{dir}_i = 1$) at price $r_j \in \mathbf{r}$.

Differentially private clearing price. In spirit of the differentially private mechanisms introduce thus far, DP-double-auction first computes a deterministic clearing price, and then applies a randomized, differentially private mechanism to determine the final, $(\varepsilon^{\mathsf{in}}, \delta^{\mathsf{in}})$-input-differentially private clearing price. For each discrete price index $j \in [l]$, we aggregate trade orders willing to sell or buy at price $r_j \in \mathbf{r}$. Let S_j denote all sell orders willing to sell at price $r_j \in [\mathbf{r}]$ and B_j denote all buy orders willing to buy at price $r_j \in [\mathbf{r}]$. Then, the number of matched pairs at price r_j is given by $u_j = \min(B_j, S_j)$, resulting in $\mathbf{u_x} = \{u_1, ..., u_j\}$. Here, we interpret $(\mathbf{u_x} : \mathbb{Z}^{\leq l} \to \mathbb{Z}^{\leq n/2})$ as the **utility function** for sampling the final clearing price with the exponential mechanism.

Exponential mechanism. The exponential mechanism first introduced by McSherry and Talwar [16], realizes a probability distribution over a range of events, for which the mechanism designer can express a *utility* score function $\mathbf{u_x}$ applicable to each event; thus, events can be allocated probability mass proportional to $\exp\big(\varepsilon \cdot \mathbf{u_x}(r)/(2\Delta u)\big)$, where Δu denotes the sensitivity of the utility function to a change to a single round input. In other words, the mechanism designer can influence the probability distribution over events by indicating which have higher utility.

In DP-double-auction, the senstivity of $\mathbf{u}_x(j)$ at any discrete price index $j \in [l]$ is simply 1; thus $\Delta u = 1$. A change in an honest input from valid to a dummy order affects at most one match per price, as does a change to the price limit of the order. Therefore,

$$\Delta \mathbf{u} = \max_{j \in [l]} \max_{\mathbf{x},\mathbf{x}'} \mid \mathbf{u}_{\mathbf{x}}(j) - \mathbf{u}_{\mathbf{x}'}(j) \mid \leq 1$$

Then, the probability distribution over which the clearing price is sampled is given by the exponential mechanism parameterized by utility function $\mathbf{u}_{\mathbf{x}}$ determined by submitted trade orders $\mathbf{x}$. Thus, the probability of each discrete price $r_j \in \mathbf{r}$ is given by;

$$\Pr[j] = \frac{\exp(\varepsilon \cdot \mathbf{u}_{\mathbf{x}}(j)/2)}{\sum_{j \in [|\mathbf{r}|]} \exp(\varepsilon \cdot \mathbf{u}_{\mathbf{x}}(j)/2)} \tag{8}$$

Since the exponential mechanism is $(\varepsilon_1^{\mathsf{in}}, 0)$-differentially private over all inputs ([16]), we consume $\varepsilon_1^{\mathsf{in}}$ of our input privacy budget when outputting the clearing price computed over $\mathbf{x}$, leaving another $\varepsilon_2^{\mathsf{in}}$ for the subsequent DP-volume-match at price r, such that $\varepsilon^{\mathsf{in}} = \varepsilon_1^{\mathsf{in}} + \varepsilon_2^{\mathsf{in}}$.

DP-Volume matching at clearing price. We subsequently apply DP-volume-match from Section 4.1 at sampled clearing price from the preceding exponential mechanism step, returning the trade outputs from DP-volume-match privately to each trading client and frozen liquidity amounts to the liquidity provider.

▶ **Theorem 13.** *DP-double auction is $(\varepsilon^{in}, \delta^{in})$-$(\varepsilon^{out}, \delta^{out})$-$m$-round differentially private.*

Proof. (Theorem 13) DP-double auction receives private trade orders and subsequently releases (1) a public clearing price $r \in \mathbf{r}$ from an $(\varepsilon_1^{\mathsf{in}})$-input-differentially private exponential mechanism, and (2) private trade outputs to each of the n clients from a $(\varepsilon_2^{\mathsf{in}}, 0)$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-round-differentially private DP-volume-match mechanism.

To establish round-differential privacy, we must argue correlated-output differential privacy for (1). This holds trivially, as trader outputs are determined by the DP-volume matching subroutine, where all trade outcomes are sampled independently from the clearing price. Thus, DP-double auction is $(\varepsilon_1^{\mathsf{in}} + \varepsilon_2^{\mathsf{in}}, \delta^{\mathsf{in}})$-$(\varepsilon^{\mathsf{out}}, \delta^{\mathsf{out}})$-round differentially private. m-round differential privacy is implied as long as liquidity frozen in each DP-volume-match execution remains secretly locked. ◀

5 Fuzzy Order Matching with MPC

To obtain a fair market in practice, we instantiate the trusted curator with a set of MPC parties who execute the market mechanisms described in Sections 4.1 and 4.2 using an MPC protocol. In this section, we present a formal description of the proposed market mechanisms, as well as some textual explanation for the steps of the algorithms where some care is required to ensure the efficiency of the MPC execution.

5.1 DP-Volume matching with MPC

The formal description of the volume matching algorithm is presented in Figures 2 and 3. Note that both the order format and the InputCheckVM protocol in Figure 2, as well as the first 3 steps of the MatchVol protocol in Figure 3 are identical to the ones in the Bucket Match algorithm from [11].

Randomness sampling. The randomness required for both the randomized matching response and the frozen liquidity is independent of the input orders, and can thus be generated ahead of time by running the procedure NoiseGen in Figure 3 together with the preprocessing phase of the MPC algorithm. The sampling, described in Figure 1, is performed using the inverse transform sampling method, similarly to what is proposed in [15]. To sample a random value from a distribution given by probability mass function P, we start by taking the corresponding cumulative distribution function, $F_X(x) = \sum_{x_i \le x} P(X = x_i)$. We then sample a secret shared value $\langle z \rangle \in (0, 1]$ uniformly at random, which can be derived from a randomly generated integer as shown in [15]. The desired distribution can now be obtained by taking the first x such that $F_X(x)$ is greater or equal to $\langle z \rangle$.

```
Sample: On input $P$, the probability distribution of a discrete random variable $X$ that may
    take $k$ different values $x_1, ...x_k$:
1. Sample $\langle z \rangle \in (0, 1]$ uniformly at random.
2. For all $i$: $F_i \leftarrow \sum_{j=1}^{i} P(X = x_j)$
3. For all $i$: $\langle c_i \rangle \leftarrow (F_i \ge \langle z \rangle)$.
4. For all $i$: $c_i \leftarrow \mathsf{Open}(\langle c_i \rangle)$.
5. Return $x_j$ for the lowest $j$ such that $c_j = 1$.
```

Figure 1 Randomness sampling with MPC.

Input format check. Since the orders are secret shared among the MPC parties, a format verification step is required. I.e., we need to check that every order i is such that $(b_i, s_i) \in \{(1, 0), (0, 1), (0, 0)\}$. To do so, we run the InputCheckVM protocol in Figure 2, where orders without the correct format are rejected.

```
InputCheckVM: On input $\mathbf{x}' = [x'_1, ..., x'_n]$, where $x'_i = (\langle b_i \rangle, \langle s_i \rangle, \langle \mathsf{id}_i \rangle)$ and $b_i, s_i, \mathsf{id}_i \in \mathbb{F}_p$:
    Check validity of inputs bits: $(0, 0) \vee (0, 1) \vee (1, 0)$.
1. Sample $\alpha_i, \beta_i, \gamma_i$ uniformly at random.
2. $\langle t_i \rangle \leftarrow \alpha_i \cdot (\langle b_i \rangle \cdot \langle b_i \rangle - \langle b_i \rangle) + \beta_i \cdot (\langle s_i \rangle \cdot \langle s_i \rangle - \langle s_i \rangle) + \gamma_i \cdot (\langle b_i \rangle \cdot \langle s_i \rangle)$
3. $t_i \leftarrow \mathsf{Open}(\langle t_i \rangle)$
4. If $t_i = 0$ then add $x'_i$ to a list $\mathbf{x}$, otherwise reject $x'_i$.
5. Return $\mathbf{x}$.
```

Figure 2 Input correctness check for the volume matching (from [11], Figure 3).

Fuzzy order matching. To achieve the desired differential privacy guarantees, we avoid revealing any of the secret shared values throughout the computation. This is unlike the Bucket Match mechanism from [11], where the direction with the most total volume was revealed, and the matching procedure was simplified by opening successful orders as soon as they were matched. As a result, we obtain a more complex procedure, described in GetMatches in Figure 3. Here, we calculate the cumulative total volume for each i and for each direction, thus obtaining $\langle \sigma_i^b \rangle$ and $\langle \sigma_i^s \rangle$ (note that we need to perform the calculations in both directions to hide which direction has more total volume). We then compare $\langle u \rangle$ (the total matched volume in each direction) with the cumulative volume at each index i, and accept every order i until $\langle u \rangle$ is exceeded. A randomized response over the matches is obtained by using the randomness $\langle \pi_i \rangle$ sampled during the preprocessing.

Liquidity compensation This phase of the algorithm is identical to the protocol in the clear as described in Section 4.1, except that we are now operating over secret shared values.

```
NoiseGen: Use Sample from Figure 1 to compute the noise for steps 5 and 9:
  - For all $i$: $\langle\pi_i\rangle \leftarrow \mathsf{Sample}(P_{\mathsf{rr}})$ (def. in Eq. 1).
  - $\langle\rho_0\rangle, \langle\rho_1\rangle \leftarrow \mathsf{Sample}(P_{\mathsf{frz}})$ (def. in Eq. 4)
MatchVol: On input $\mathbf{x}'$, $\mathbf{x}^{\mathsf{liq}}$, submitted by $(\mathcal{P}^{\mathsf{trd}}_1, ..., \mathcal{P}^{\mathsf{trd}}_n)$ and $\mathcal{P}^{\mathsf{liq}}$, respectively, where $\mathbf{x}' = [x'_1, ..., x'_n]$, $x'_i = (\langle b_i\rangle, \langle s_i\rangle, \langle \mathsf{id}_i\rangle)$, $\mathbf{x}^{\mathsf{liq}} = (\langle x^{\mathsf{liq}}_0\rangle, \langle x^{\mathsf{liq}}_1\rangle)$ and $b_i, s_i, \mathsf{id}_i, x^{\mathsf{liq}}_0, x^{\mathsf{liq}}_1 \in \mathbb{F}_p$:
1. Let $\mathbf{x} \leftarrow \mathsf{InputCheckVM}(\mathbf{x}')$
  Step [1a] Deterministic matching of buy & sell orders
2. For all $i$: $\langle B\rangle \leftarrow \langle B\rangle + \langle b_i\rangle$, and $\langle S\rangle \leftarrow \langle S\rangle + \langle s_i\rangle$
3. Let $\langle c\rangle \leftarrow (\langle S\rangle > \langle B\rangle)$ and $\langle u\rangle \leftarrow \langle c\rangle \cdot \langle B\rangle + (1 - \langle c\rangle) \cdot \langle S\rangle$.
4. $\mathsf{match} \leftarrow \mathsf{GetMatches}(\mathbf{x}, \langle c\rangle, \langle u\rangle, \langle B\rangle, \langle S\rangle)$
  Step [1b] Randomized response over order matches
5. For all $i$:
  - Let $\langle \mathsf{trade}_i\rangle \leftarrow \langle\pi_i\rangle \cdot \langle \mathsf{match}_i\rangle + (1 - \langle\pi_i\rangle) \cdot (1 - \langle \mathsf{match}_i\rangle)$
  - Let $\langle b^{\mathsf{out}}_i\rangle \leftarrow \langle s_i\rangle \cdot \langle \mathsf{trade}_i\rangle + \langle b_i\rangle \cdot (1 - \langle \mathsf{trade}_i\rangle)$
  - Let $\langle s^{\mathsf{out}}_i\rangle \leftarrow \langle b_i\rangle \cdot \langle \mathsf{trade}_i\rangle + \langle s_i\rangle \cdot (1 - \langle \mathsf{trade}_i\rangle)$
  - Add $y_i = (\langle b^{\mathsf{out}}_i\rangle, \langle s^{\mathsf{out}}_i\rangle, \langle \mathsf{id}_i\rangle)$ to the output list $\mathbf{y}$.
  Step [2a] Liquidity compensation for sampled trades
6. For all $i$: $\langle o^b\rangle \leftarrow \langle o^b\rangle + \langle b^{\mathsf{out}}_i\rangle$, and $\langle o^s\rangle \leftarrow \langle o^s\rangle + \langle s^{\mathsf{out}}_i\rangle$
7. Let $\langle\Delta_0\rangle \leftarrow (\langle o^s\rangle - \langle o^b\rangle)$ and $\langle\Delta_1\rangle \leftarrow -\langle\Delta_0\rangle$
  Step [2b] Randomized liquidity freezing
8. $\mathbf{y}^{\mathsf{liq}} = (\langle y^{\mathsf{liq}}_0\rangle, \langle y^{\mathsf{liq}}_1\rangle) \leftarrow (\langle y^{\mathsf{liq}}_0\rangle + \langle\Delta_0\rangle - \langle\rho_0\rangle\,, \langle y^{\mathsf{liq}}_1\rangle + \langle\Delta_1\rangle - \langle\rho_1\rangle)$
9. $\mathsf{FreezeLiq}(\langle\rho_0\rangle, \langle\rho_1\rangle)$, return $\mathbf{y} = [y_1, ..., y_n]$, $\mathbf{y}^{\mathsf{liq}}$ to $(\mathcal{P}^{\mathsf{trd}}_1, ..., \mathcal{P}^{\mathsf{trd}}_n)$ and $\mathcal{P}^{\mathsf{liq}}$, respectively.
Subroutines invoked by MatchVolume
GetMatches: On input $(\mathbf{x}, \langle c\rangle, \langle u\rangle, \langle B\rangle, \langle S\rangle)$:
1. For all $i$: $\langle \mathsf{big}_i\rangle \leftarrow \langle c\rangle \cdot \langle s_i\rangle + (1 - \langle c\rangle) \cdot \langle b_i\rangle$.
2. For all $i$, let $\langle\sigma^b_i\rangle \leftarrow \sum_{h=1}^{i}\langle b_h\rangle$ and $\langle\sigma^s_i\rangle \leftarrow \sum_{h=1}^{i}\langle s_h\rangle$.
3. For all $i$, let $\langle\sigma'_i\rangle \leftarrow \langle c\rangle \cdot \langle\sigma^s_i\rangle + (1 - \langle c\rangle) \cdot \langle\sigma^b_i\rangle$
4. For all $i$, let $\langle \mathsf{match}'_i\rangle \leftarrow (\langle\sigma'_i\rangle \leq \langle u\rangle) \cdot \langle \mathsf{big}_i\rangle$
5. For all $i$: $\langle \mathsf{match}_i\rangle \leftarrow (1 - \langle c\rangle) \cdot \langle s_i\rangle + \langle c\rangle \cdot \langle b_i\rangle + \langle \mathsf{match}'_i\rangle$
6. Return $\mathsf{match} = [\langle \mathsf{match}_1\rangle, ..., \langle \mathsf{match}_n\rangle]$
FreezeLiq: On input $(\langle\rho_0\rangle, \langle\rho_1\rangle)$:
  1. Update $(\langle\rho^{\mathsf{frz}}_0\rangle, \langle\rho^{\mathsf{frz}}_1\rangle) \leftarrow (\langle\rho^{\mathsf{frz}}_0\rangle, \langle\rho^{\mathsf{frz}}_1\rangle) + (\langle\rho_0\rangle, \langle\rho_1\rangle)$.
```

Figure 3 ε, δ-DP unit volume matching with MPC

5.2 DP double auction with MPC

The formal description of the double auction algorithm is presented in Figures 4 and 5.

Input format check. The correctness of the inputs is verified by the procedure InputCheckDA in Figure 4, which checks that $\langle \mathsf{dir}_i\rangle$ as well as every $\langle w_{ij}\rangle$ are bits and rejects order i if that is not the case. For the orders in the correct format, this procedure additionally converts $\langle w_{ij}\rangle$ and $\langle \mathsf{dir}_i\rangle$ into a sequence of pairs $(\langle b_{ij}\rangle, \langle s_{ij}\rangle)$, where $\langle b_{ij}\rangle$ and $\langle s_{ij}\rangle$ represents whether order i is a buy, sell or a dummy for price j (i.e., $\langle b_{ij}\rangle$ and $\langle s_{ij}\rangle$ have the same meaning as in Section 5.1, but are now associated with a specific price r_j).

```
InputCheckDA: On input $\mathbf{x}' = [x'_1, ..., x'_n]$, where $x'_i = (\mathbf{w}_i, \langle \mathsf{dir}_i \rangle, \langle \mathsf{id}_i \rangle)$, $\mathbf{w}_i = [\langle w_{i1} \rangle, ..., \langle w_{il} \rangle]$
    and $w_{ij}, \mathsf{dir}_i, \mathsf{id}_i \in \mathbb{F}_p$:
    Check all inputs are bits.
1.  For all $j$: $\alpha_{ij} \leftarrow \mathcal{F}_{\mathsf{Rand}}()$.
2.  $\beta_i \leftarrow \mathcal{F}_{\mathsf{Rand}}()$.
3.  $\langle t_i \rangle \leftarrow \alpha_{i1} \cdot (\langle w_{i1} \rangle \cdot \langle w_{i1} \rangle - \langle w_{i1} \rangle) + ... + \alpha_{il} \cdot (\langle w_{il} \rangle \cdot \langle w_{il} \rangle - \langle w_{il} \rangle)$
4.  $\langle t_i \rangle \leftarrow \langle t_i \rangle + \beta_i \cdot (\langle \mathsf{dir}_i \rangle \cdot \langle \mathsf{dir}_i \rangle - \langle \mathsf{dir}_i \rangle)$
5.  $t_i \leftarrow \mathsf{Open}(\langle t_i \rangle)$
6.  If $t_i \neq 0$ then reject $\mathbf{x}'_i$. Otherwise, continue to the next step.
7.  For all $j$, let $\langle b_{ij} \rangle = \langle w_{ij} \rangle \cdot (1 - \langle \mathsf{dir}_i \rangle)$ and $\langle s_{ij} \rangle = \langle w_{ij} \rangle \cdot \langle \mathsf{dir}_i \rangle$.
8.  Add $x_i = (\langle b_{i1} \rangle, \langle s_{i1} \rangle, ..., \langle b_{il} \rangle, \langle s_{il} \rangle, \langle \mathsf{id}_i \rangle)$ to a list $\mathbf{x}$.
9.  Return $\mathbf{x}$.
```

Figure 4 Input correctness check for the double auction.

Exponential mechanism. We obliviously determine how many orders can be matched at each price point by calculating $\langle u_j \rangle$ for each price r_j the same way as $\langle u \rangle$ was calculated in the volume matching protocol. The exponential mechanism can now be used to select the best trading price. The probability $\Pr[j]$ associated with price point r_j depends on the corresponding utility value $\langle u_j \rangle$, and since the utility must remain private, the calculated probabilities $\Pr[j]$ will also be secret shared values. While this does not affect the sampling procedure (the algorithm in Figure 1 remains unchanged if the probability mass function is private), computing each $\Pr[j]$ will require the expensive evaluation of a secure exponentiation.

To avoid exponentiation and efficiently compute the selection probabilities with MPC, we use the techniques proposed in [5]. Firstly, instead of considering the selection probabilities as given in Eq. 8, we reduce the complexity by calculating unnormalized probabilities $\langle W_j \rangle$ (called *weights*), where $\langle W_j \rangle = \exp(\varepsilon \cdot \langle u_j \rangle / 2)$. The clearing price sampling can later be performed using these weights by multiplying $\langle z \rangle$ with $\sum_{j=1}^{l} \exp(\varepsilon \cdot \langle u_j \rangle / 2)$, as shown in the SampleWeight procedure in Figure 5. Secondly, there are two possible solutions for computing the weights according to the value of ε (note that ε is public and fixed beforehand):

(i) For $\varepsilon = 2 \cdot \ln(2)$, we get $\langle W_j \rangle = 2^{\langle u_j \rangle}$. This value can be directly written as $(\langle 0 \rangle, \langle 0 \rangle, \langle 2 \rangle, \langle u_j \rangle)$ by using the floating-point notation introduced in [2]. With this notation, a secret shared floating-point value $\langle f \rangle$ is represented as a tuple $(\langle s \rangle, \langle o \rangle, \langle v \rangle, \langle p \rangle)$ with $f = (1 - 2 \cdot s) \cdot (1 - o) \cdot v \cdot 2^p$, where s is the sign bit (set to 1 when f is negative), o is the zero bit (set to 1 when f is zero), v is the mantissa and p the exponent.

(ii) For $\varepsilon = 2 \cdot \ln(2)/2^d$, where $d \in \mathbb{Z}$, we get $\langle W_j \rangle = 2^{\langle u_j \rangle / 2^d} = 2^{\lfloor \langle u_j \rangle / 2^d \rfloor} \cdot 2^{(\langle u_j \rangle \mod 2^d)/2^d}$. The weight $\langle W_j \rangle$ can thus be obtained by calculating the exponentiation with base 2 on the integer part of $\langle u_j \rangle / 2^d$, and multiplying it by a corrective term $2^{(\langle u_j \rangle \mod 2^d)/2^d}$ which takes one out of 2^d possible values depending on u_j. The 2^d possible terms are publicly pre-computed and the correct one is obliviously selected using $\langle u_j \rangle$.

There is an additional procedure in [5] for calculating the weights for arbitrary values of ε. This procedure relies on the decomposability of the considered utility function, meaning that clients can locally calculate the weights associated with their own inputs and these can later be combined to obtain a correct global weight using MPC. Since our utility function is not decomposable, this method is not applicable. An alternative for computing the weights for arbitrary ε would be to first publicly calculate all the possible weights according to the amount of submitted orders, and then obliviously select the correct weight for each price point. This would however imply several secure comparisons and become inefficient for large amounts of submitted orders and available price points. It is therefore preferable to choose ε according to the formats in (i) or (ii), which already provide considerable flexibility.

```
FindPrice: On input $\mathbf{x}'$, $\mathbf{x}^{\mathsf{liq}}$, submitted by $(\mathcal{P}_1^{\mathsf{trd}}, ..., \mathcal{P}_n^{\mathsf{trd}})$ and $\mathcal{P}^{\mathsf{liq}}$, respectively, where
    $\mathbf{x}' = [x'_1, ..., x'_n]$, $x'_i = (\mathbf{w}_i, \langle \mathsf{dir}_i \rangle, \langle \mathsf{id}_i \rangle)$, $\mathbf{w}_i = [\langle w_{i1} \rangle, ..., \langle w_{il} \rangle]$, $\mathbf{x}^{\mathsf{liq}} = (\langle x_0^{\mathsf{liq}} \rangle, \langle x_1^{\mathsf{liq}} \rangle)$ and
    $w_{ij}, \mathsf{dir}_i, \mathsf{id}_i, x_0^{\mathsf{liq}}, x_1^{\mathsf{liq}} \in \mathbb{F}_p$, as well as a list of prices $\mathbf{r} = [r_1, ..., r_l]$:
1. Let $\mathbf{x} \leftarrow \mathsf{InputCheckDA}(\mathbf{x}')$
   Determine best price & call volume matching algorithm
2. For all $j$: $\langle B_j \rangle \leftarrow \langle B_j \rangle + \langle b_{1j} \rangle + ... + \langle b_{nj} \rangle$, and $\langle S_j \rangle \leftarrow \langle S_j \rangle + \langle s_{1j} \rangle + ... + \langle s_{nj} \rangle$.
3. For all $j$, let $\langle c_j \rangle \leftarrow (\langle S_j \rangle > \langle B_j \rangle)$ and $\langle u_j \rangle \leftarrow \langle c_j \rangle \cdot \langle B_j \rangle + (1 - \langle c_j \rangle) \cdot \langle S_j \rangle$.
4. Calculate weights $\langle W_1 \rangle, ..., \langle W_l \rangle$ using Algorithm 3 from [5] on input $\langle u_1 \rangle, ..., \langle u_l \rangle$.
5. $R \leftarrow \mathsf{SampleWeight}(\langle W_1 \rangle, ..., \langle W_l \rangle)$.
6. Set $\mathbf{x}_i^{\mathsf{match}} = [\langle b_{iR} \rangle, \langle s_{iR} \rangle, \langle \mathsf{id}_i \rangle]$
7. Execute MatchVol from Figure 3 from step 4 with inputs $\mathbf{x}^{\mathsf{match}} = [x_1^{\mathsf{match}}, ..., x_n^{\mathsf{match}}]$, $\mathbf{x}^{\mathsf{liq}}$, $\langle c_R \rangle$
   and $\langle u_R \rangle$.
Subroutine invoked by FindPrice
SampleWeight: On input a list of weights $\langle W_1 \rangle, ..., \langle W_l \rangle$ associated with prices $r_1, ..., r_l$:
1. For all $j$: $\langle F_j \rangle \leftarrow \sum_{h=1}^{j} \langle W_h \rangle$
2. Sample $\langle z' \rangle \in (0, 1]$ uniformly at random and let $\langle z \rangle \leftarrow \langle z' \rangle \cdot \langle F_l \rangle$.
3. For all $j$: $\langle c_j \rangle \leftarrow (\langle F_j \rangle \geq \langle z \rangle)$.
4. For all $j$: $c_j \leftarrow \mathsf{Open}(\langle c_j \rangle)$.
5. Return $r_j$ for the lowest $j$ such that $c_j = 1$.
```

Figure 5 ε, δ-DP unit double auction with MPC

After a price is selected, we can run the volume matching algorithm in Figure 3, starting from step 4 of the MatchVol procedure. Note that since we do not know which orders accept the selected price, every order submitted to the double auction will also be considered when subsequently executing the volume matching. Orders that did not accept the select price will appear as dummies during the matching.

5.3 Experiments

To benchmark the performance of our MPC algorithms, we implemented and executed them using Scale-Mamba [3] with Shamir secret sharing between 3 parties. All the parties are run on identical machines with an Intel i-9900 CPU and 128GB of RAM. The ping time between all the machines is 1.003 ms.

References

1 Abbas Acar, Z Berkay Celik, Hidayet Aksu, A Selcuk Uluagac, and Patrick McDaniel. Achieving secure and differentially private computations in multiparty settings. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*, pages 49–59. IEEE, 2017. `https://doi.org/10.1109/PAC.2017.12`.

2 Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA*, 2013.

3 Abdelrahaman Aly, Kelong Cong, Daniele Cozzo, Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Oliver Scherer, Peter Scholl, Nigel P. Smart, Titouan Tanguy, and Tim Wood. SCALE-MAMBA v1.12: Documentation, 2021. URL: `https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf`.

4 Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient Privacy Preserving Smart Contracts. *Cryptology ePrint Archive*, 2022. `https://eprint.iacr.org/2022/1435`.

5 Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private median. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2147–2164. USENIX Association, August 2020. URL: `https://www.usenix.org/conference/usenixsecurity20/presentation/boehler`.

6 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. MPC joins the dark side. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 148–159, 2019. `https://doi.org/10.1145/3321705.3329809`.

7 John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Multi-party computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross. *Intelligent Systems in Accounting, Finance and Management*, 28(4):239–267, 2021. `https://doi.org/10.1002/isaf.1502`.

8 David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri De Ruiter, and Alan T Sherman. cmix: Mixing with minimal real-time asymmetric cryptographic operations. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 557–578. Springer, 2017. `https://doi.org/10.1007/978-3-319-61204-1_28`.

9 David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. `https://www.doi.org/10.1145/358549.358563`.

10 Tarun Chitra, Guillermo Angeris, and Alex Evans. Differential privacy in constant function market makers. *Cryptology ePrint Archive*, 2021. `https://eprint.iacr.org/2021/1101`.

11 Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P Smart, and Younes Talibi Alaoui. Kicking-the-bucket: Fast privacy-preserving trading using buckets. *Cryptology ePrint Archive*, 2021. To appear at FC'22. `https://eprint.iacr.org/2021/1549`.

12 Mariana Botelho da Gama, John Cartlidge, Nigel P. Smart, and Younes Talibi Alaoui. All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation. Cryptology ePrint Archive, Paper 2022/923, 2022. `https://eprint.iacr.org/2022/923`. URL: `https://eprint.iacr.org/2022/923`.

13 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006. `https://doi.org/10.1007/11681878_14`.

14 Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. `http://dx.doi.org/10.1561/0400000042`.

15 Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially private data aggregation with optimal utility. ACSAC '14, page 316–325, New York, NY, USA, 2014. Association for Computing Machinery. `doi:10.1145/2664243.2664263`.

16 Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103. IEEE, 2007. `https://doi.org/10.1109/FOCS.2007.66`.

17 United States of America before the Securities and Exchange Commission. In the matter of itg inc. and alternet securities, inc., exchange act release no. 75672. `https://www.sec.gov/litigation/admin/2015/33-9887.pdf`, 12 Aug 2015.

18 United States of America before the Securities and Exchange Commission. In the matter of pipeline trading systems llc, et al., exchange act release no. 65609. `https://www.sec.gov/litigation/admin/2011/33-9271.pdf`, 24 Oct 2011.

19 United States of America before the Securities and Exchange Commission. In the matter of liquidnet, inc., exchange act release no. 72339. `https://www.sec.gov/litigation/admin/2014/33-9596.pdf`, 6 Jun 2014.

20 Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. *Advances in neural information processing systems*, 23, 2010. https://proceedings.neurips.cc/paper_files/paper/2010/file/0d0fd7c6e093f7b804fa0150b875b868-Paper.pdf.

21 Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. Training differentially private models with secure multiparty computation. *arXiv preprint arXiv:2202.02625*, 2022. https://arxiv.org/abs/2202.02625.

22 Penumbra. ZSwap documentation. https://protocol.penumbra.zone/main/zswap.html, 2023.

23 Monica Petrescu and Michael Wedow. Dark pools in european equity markets: emergence, competition and implications. *ECB Occasional Paper*, (193), 2017. https://doi.org/10.2866/555710.

24 Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021. https://doi.org/10.1145/3418290.

SoK: Privacy-Enhancing Technologies in Finance

Contribution

- Co-author.

Remarks

Under submission. An invited talk on this work was given at the *3rd Workshop on Decentralized Finance* at Financial Cryptography and Data Security 2023.

SoK: Privacy-Enhancing Technologies in Finance

Carsten Baum[1], James Hsin-yu Chiang[1], Bernardo David[2], Tore Kasper Frederiksen[3],

[1] Technical University of Denmark
`cabau@dtu.dk`[⋆], `jachiang@ucla.edu`
[2] IT University of Copenhagen
`bernardo@bmdavid.com`
[3] No affiliation
`jot2re@gmail.com`[⋆⋆]

Abstract. Financial applications have historically required strong security guarantees. These can be achieved in a digital world via cryptographic tools but have traditionally been employed to provide authenticity and privacy for data exchanged between clients and financial institutions over insecure networks (*e.g.* the Internet). However, the recent advent of cryptocurrencies and smart contract platforms, based on blockchains, allowed financial transactions to be carried out over a public ledger, instead of keeping such transactions exclusive to private institutions. This introduced a new challenge: Allowing any third party to verify the validity of financial operations by means of public records on a blockchain, while keeping sensitive data private. Advanced cryptographic techniques such as Zero Knowledge (ZK) proofs rose to prominence as a solution to this challenge, allowing for the owner of sensitive information (*e.g.* the identities of users involved in an operation) to provide unforgeable evidence that a certain operation has been correctly executed without revealing said sensitive data. Moreover, once the Fintech community discovered the power of such advanced techniques, it also became clear that performing arbitrary computation on private data by means of secure Multiparty Computation (MPC), and related techniques like Fully Homomorphic Encryption (FHE), would allow more powerful financial applications, also in traditional finance, involving sensitive data from multiple sources.
In this survey, we present an overview of the main Privacy-Enhancing Technologies (PETs) available in the state of the art of current advanced cryptographic research and how they can be used to address challenges in both traditional and decentralized finance. In particular, we consider the following classes of applications: 1. Identity Management, KYC & AML; 2. Legal; 3. Digital Asset Custody; and 4. Markets & Settlement. We examine how ZK proofs, MPC and related PETs have been used to tackle challenges in each of these applications. Finally, we propose

[⋆] Part of the work are supported by Partisia. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of Partisia.

[⋆⋆] This work was carried while working at Protocol Labs.

future applications of PETs as Fintech solutions to currently unsolved issues. While we present a broad overview, we focus mainly on those applications that require privacy preserving computation on data from multiple parties.

1 Introduction

Modern Cryptography and Traditional Finance. Due to their sensitive nature, financial applications require strong security guarantees. Clearly, it is necessary to ensure authenticity and integrity of any financial operation, *i.e.* guaranteeing that the operation has been ordered by an entity authorized to do so and that this order has not been tampered with. Moreover, it is also necessary to achieve *privacy*, *i.e.* preventing attackers from obtaining sensitive information related to financial operations (*e.g. the identities of entities involved in a transaction and/or the value of that transaction*). In the digital realm, these security guarantees can be achieved even in the presence of powerful adversaries who control communication networks (*e.g.* the Internet) by means of digital signatures (to ensure authenticity) and encryption (to ensure privacy).

A Decentralized Conundrum: The meteoric rise of decentralized financial applications based on cryptocurrencies and smart contracts hosted on blockchain platforms brought to light a whole new set of challenges. While traditional financial applications are hosted and executed by financial institutions in a centralized manner, the decentralized nature of blockchain-based applications requires all operations to be verifiable by third parties by means of publicly available records. If only simple cryptographic primitives are employed, this means that sensitive data that was once internally handled by financial institutions must now be exposed on the blockchain in order to perform a financial application. For example, Bitcoin requires revealing the sender and the receiver of a financial token that is transferred, so that a transfer transaction is considered valid if and only if the rightful owner of the token signs it.

Privacy-Enhancing Technologies (PETs) to the Rescue: While sacrificing privacy to achieve decentralization may be acceptable in some situations, most financial operations involving companies and private citizens cannot be conducted in this manner due to a number of reasons (*e.g.* protecting business interests and complying to local/international regulations). In order to solve this issue, the cryptocurrency community turned to Privacy-Enhancing Technologies (PETs) that allow for achieving the same authenticity and privacy guarantees as in traditional centralized financial applications while providing the public verifiability guarantees needed in decentralized blockchain platforms. In particular, many of the first proposals towards this goal involved using a technology called Zero Knowledge (ZK) proof systems: a method that allows the owner of sensitive data to prove a statement about this data without having to reveal it. For example, in the token transfer transaction example, a ZK proof allows a user to prove that an "encrypted" transfer transaction has been signed by the rightful owner of the token, without revealing neither the owner's nor the receiver's identity.

From Decentralized to Traditional Finance: The vast usefulness of advanced PETs in blockchain applications also sparked an interest in deploying similar solutions for traditional financial applications. The aforementioned ZK proof technology has also been used in innovative solutions to the Know Your Client (KYC) and Anti Money Laundering (AML) problems commonly encountered in the banking industry. As in the case of privacy preserving cryptocurrency transactions, in many scenarios an entity wants to prove that they comply with KYC/AML regulations without revealing their identity nor their sensitive data. For example, a client can prove to a third party service provider that their identity has been verified by their bank and that they are authorized to use a certain service and perform operations up to a certain financial volume, while keeping their identity, and other attributes (*e.g.* the list of operations they are allowed to perform) private.

PETs for the Masses - or - From ZK to MPC: The ZK proof technology lends itself extremely well to applications that require a single entity to publicly prove a statement about its private data, *e.g.* the KYC/AML or cryptocurrency examples above. However, it is limited by the fact that the entity who generates a ZK proof must necessarily know all the private information about which the statement is proven. This is a serious limitation in two cases: 1. applications that must process sensitive data provided by multiple entities; 2. applications where certain data (*e.g.* cryptographic secret keys) are far too valuable to be stored on a single device, which leaks the data if its security is compromised. Fortunately, these limitations can be addressed by means of secure Multiparty Computation (MPC) protocols, which allow a set of entities to jointly execute an arbitrary program that computes on an encrypted version of their private data and only reveals the output of this computation.

For example, specific-purpose MPC protocols have long been used for sealed-bid auctions among entities who do not trust each other, nor a third party auctioneer. In this case, the parties provide as input "encrypted" versions of their bids and jointly compute a program that determines the winner of the auction, without revealing the value of the bids or any other information. In the context of blockchain-based cryptocurrencies, MPC protocols have also been successfully deployed for protecting secret signature keys used to authorizing/authenticating transactions. In this case, many entities locally stores a "share" of the signing key that does not reveal any information about the key itself unless all shares are united. When a transaction must be signed, all these entities use MPC to jointly execute a program that takes as input all the signing key shares, reconstructs the true key and computes the signature on the given transaction, while only revealing the resulting signature and nothing else. Since knowledge of the key is split among many entities, an attacker now has to compromise many, potentially all, entities instead of a single server.

1.1 Systematizing Privacy-enhancing Technologies in Finance

The goal of this SoK is to provide an overview of financial applications that can benefit from PETs. As summarized in Figure 1, we consider 5 main classes of financial applications, which are each addressed in the specific section indicated next to the application class name. At a high level, these applications can be potentially facilitated by the PETs indicated in the **PET** column of Figure 1, which are introduced in detail in Section 2. In particular, we focus on financial applications that require handling private data from multiple entities, as ZK proof technology for financial applications has been extensively addressed in previous surveys (*e.g.* [BKB20,AS22,AZ19,PNP20]). While we aim at providing a general overview of PETs for financial applications covering broad ranges of both PETs and applications, we do not intend to provide an exhaustive review of the PET literature. For each class of applications, we strive to survey the works that introduced the most relevant insights and groundbreaking results, since it would be infeasible to cover every single optimization of each PET that would be relevant for each application.

	PET (§2)
(§3) Identity, KYC & AML	MPC, ZK
(§4) Legal	MPC, ZK
(§5) Digital Asset Custody	TSS, MPC
(§6) Markets & Settlement	(F)HE, MPC, ZK
(§7) Future applications	PSI, DP, MPC

Fig. 1. PET stack for financial applications. TSS = Threshold Secret Sharing, DP = Differential Privacy, (F)HE=(Fully) Homomorphic Encryption, PSI = Private Set Intersection, MPC = Multiparty Computation, ZK = Zero Knowledge proofs. See Section 2 for a discussion of each concept.

The financial applications we cover and the respective relevant PETs are summarized as follows:

Identity, KYC & AML (Section 3): Identity management is a classical problem that has the added challenges of Know Your Client (KYC) and Anti Money Laundering (AML) regulations in the financial sector. PETs can be used in these applications to provide robust identity management with privacy preserving methods for enforcing KYC & AML regulations in both decentralized and traditional scenarios.

Legal Procedures (Section 4): Many legal procedures require evidence to be presented in court. However, in many cases the evidence or even the relevant law/regulation must be kept private. PETs allow for such legal procedures to be conducted without sacrificing neither privacy nor auditability (*i.e.* the ability of any entity to verify that a legal procedure has been properly executed). Furthermore, due to the novelty of PETs it is not always clear how they fit within existing legal frameworks and how they might help and provide utility while fulfilling privacy regulations such as GDPR.

Digital Asset Custody (Section 5): Digital assets such as cryptocurrencies are usually transferred by means of a digital signature, which can only be generated given a secret key. Since storing this key in a single device poses a risk of key leakage, PETs can be employed to distribute the signing power (and thus the power to move the asset) among many entities, in such a way that the system is only compromised if all entities are compromised.

Markets & Settlements (Section 6): Both the traditional and decentralized financial markets use complex trading instruments that may be abused by entities who retain privileged information about trades. PETs provide a robust solution to this issue via distributed "Dark Pools" or privacy preserving DeFi mechanisms (*e.g.* Automated Market Makers) that process trades without revealing any sensitive information to the entities envolved.

Future Applications (Section 7): Besides financial applications that have already been addressed in previous work, we propose that several PETs can be potentially used to address other interesting challenges in finance. In particular, recent advances in PETs enable the execution of advanced machine learning (ML) algorithms on private data, allowing for detecting patterns (*e.g.* for fraud) without revealing neither the ML models nor the data.

1.2 Organization

In Section 2, we start with an introduction to relevant advanced PETs and their security guarantees. Next, in subsequent sections, we survey both traditional and decentralized finance applications, pinpointing the PETs that have been shown to solve important challenges in each application. Applications related to Identity, KYC & AML are discussed in Section 3. Applications related to Legal Procedures are discussed in Section 4. Applications related to Digital Asset Custody are discussed in Section 5. Applications related to Markets & Settlements are discussed in Section 6. Finally, in Section 7, we conclude with remarks on potential future applications of PETs in finance.

2 Available Privacy-enhancing Technologies

Before we describe applications of *Privacy-Enhancing Technologies* (PETs) to finance, we will give a short overview over existing PETs and how mature they are.

Zero-Knowledge proofs. Zero-Knowledge proofs [GMR85] are cryptographic algorithms which allows a prover to convince a distrusting verifier that a certain statement is true. While the statement (usually specified in the form of a program) is known to the verifier, the proof (e.g. a certain input that makes the program output 0) is never leaked to the verifier. There exists a large variety of different ZK proof algorithms, and choosing the optimal proof depends largely on the application. Recently, efforts have been underway to standardize ZK proofs[4] to make them more accessible to practitioners.

Private Set Intersection. Private Set Intersection (PSI) [FNP04] allows two (or more) distrusting parties with respective input sets S_1 and S_2 to securely learn their intersection, i.e. $S_1 \cap S_2$, without revealing the non-intersecting elements to the other party. For example, if party 1 has $S_1 = \{a, b, d\}$ and party 2 has $S_2 = \{b, c, e\}$ then both parties will learn that they have b in common in their sets. At the same time, party 2 will not learn that party 1 also had a, d in its input set and vice-versa for c, e. Highly efficient PSI protocols have been developed recently and some, such as the one developed by Chen *et al.* [CLR17] found applications in industry.

Threshold Secret Sharing. Threshold Secret Sharing (TSS) allows a dealer to distribute [Sha79] a secret x among n different parties, who each receive a *share* of the secret. Given a threshold $t < n$, TSS guarantees that if t or less parties pool their shares together, then they cannot reconstruct any information about x. If instead more than t parties cooperate (*i.e.* pool their shares), then x can be reconstructed. Multiple versions of secret sharing exist, for example with security against share-holders who don't act honestly during the reconstruction of the secret [CGMA85,RB89]. Moreover, secret-sharing can be generalized so that not a threshold decides about the possibility of reconstruction, but instead any pattern can be used by the sender of the shares.

Multiparty Computation. Cryptographic protocols for Multiparty Computation (MPC) [BGW88,CCD88,GMW86] allow 2 or more mutually distrusting parties who each have an input x_i to evaluate an arbitrary function $y = f(x_1, \dots, x_n)$ on their inputs. MPC guarantees that only the function output y and no other information about the inputs is revealed. One can see PSI as a special case of MPC where the computed function is the intersection of input sets. MPC can be made robust against parties who maliciously deviate from the protocol description, and security usually holds if less than a threshold t of the participants in the computation collaborate to undermine the security. Therefore, MPC can be seen as constructing a *distributed trusted entity*. Recent progress in MPC research has made practical use of MPC possible[5].

Fully-Homomorphic Encryption. Fully-Homomorphic Encryption (FHE) is a special type of encryption scheme first proposed in [RAD+78] and later realized in [Gen09]. In FHE, everyone with a so-called public key can encrypt information, while only the holder of the private key can decrypt it later. In addition, given encrypted of data as well as the public key, anyone can *perform computations* on the encrypted data and evaluate algorithms on secret inputs. For example. Given encryptions $[x], [y], [z]$ of the values x, y, z, FHE allows to compute an

[4] See `https://zkproof.org/`.
[5] https://www.mpcalliance.org/

encryption $[x \cdot y + z]$ of $x \cdot y + z$ or *any other efficiently computable algorithm* on these inputs. The clue is that the decryptor who obtains $[x \cdot y + z]$ will only learn $x \cdot y + z$ but not the *inputs to the computation*. Although concrete FHE schemes are relatively new, the technology is already somewhat mature[6] and powerful testing implementations[7] are available.

Differential Privacy. Differential Privacy [DMNS06] (DP) is a technique to compute *add noise to outcomes of algorithms* such that leakage about the inputs of the computation is minimized. The level of the noise is calibrated such that mathematical guarantees about the privacy of the inputs can be given.

A note on Trusted Execution Environments. Trusted Execution Environments (TEE) such as Intel's SGX are special modes of modern processors. A processor in its trusted execution setting guarantees that programs and their data are shielded from every other program running on the computer - even the operating system or any user having full access. A secure TEE allows to build many of the aforementioned PETs such as ZK proofs, PSI, MPC etc. "cheaply" and without additional cryptographic tools. In practice, SGX and similar technologies from other vendors[8] are regularly broken and do not offer the protection that they claim. We therefore do not consider it as a PET in this document.

3 Identity, KYC, AML

A general issue facing the financial world is the validation of customer identities and attributes. Laws and regulations require financial institutions, both classical and decentralized, to employ Know Your Customer (KYC) rules, for example to prevent money laundering and to be able aid in criminal cases - or even to lock accounts in case of sanctions. Being able to correctly determine a legal owner of an account can in itself help in preventing money laundering by precluding the use of fake accounts which could otherwise aid in *smurfing*, see Sec. 3.2. In the EU this is for example in place through the Anti-Money Laundering Directives and in the US through the Money Laundering Control Act of 1986. In the classical banking setting such validations are carried out through customers going to their physical bank and bringing required documents to prove their identity, residence or perhaps even criminal history, of which the bank would keep a copy. However, with the advent of online-only banks such as Lunar, Revolut and N26, along with crypto-currency exchanges like Binance and Coinbase, such validations become tricky, as there are no physical locations to validate identities.

Today KYC is instead carried out online, and in many cases through machine learning algorithms, where customers upload copies of their data which gets validated. When it comes to security this unfortunately as several disadvantages: i) it is easy to create a picture of a document or manufacture it, or even modify

[6] https://fhe.org/
[7] https://www.openfhe.org/
[8] See e.g. the exhaustive list on `https://sgx.fail/`.

some data of a real document [SSLM21]; and ii) the leakage of legitimate documents online allows an adversary to steal identities. Simply considering how often a copy of ones passport is needed (e.g. basically any hotel or accommodation in any country), it is not hard to see that copies of legitimate documents will be easy to find on the dark market. Even though requirements can be made to include selfies or short videos to validate authenticity, this has turned into a race against Photoshop and deep fakes, which have shown tremendous advancement in the recent years [TJW22]. While such attacks are also possible in physical space (i.e. creating fake documents and having them be validated by a human), it is significantly more cumbersome for an attacker due to human involvement and more expensive to mount and therefore does not *scale* like digital-only attacks. Thus it is clear that the digital attack vector on KYC is the weakest link in account validation.

3.1 Identity management

One possible way of combating attacks when validating digital copies of physical identity documents is simply to move the documents into the digital realm. Digital signatures and revocation systems can ensure that digital documents are legitimate. This is done by combining them with an identification scheme where a user needs to prove they know a password/key used in the construction of their digital identity. This can prevent theft through simply copying the digital document, e.g. when validating it. Often a simple password or key is not deemed secure enough in financial applications by law [PU15], and a second factor is required when used for financial transactions of a certain size. Thus the use of authentication apps is common in electronic ID (eID) solutions, like the Danish MitID. Such eID solutions validate user-identities by the a trusted issuer during setup, allowing other applications to piggy-back on existing validation. This naturally comes with a risk of compromise or identity sharing through the eID provider, although it may arguably be harder than with their physical counterparts.

Single Sign-On eIDs are typically validated by a centralized and trusted server that is able to perform relevant logging, and hence poses a risk to user privacy. Furthermore, such identity management is not exclusive to official or government identities, but can involve any kind of self-reported identity, which is the case for example for a Facebook or Google account. These platforms act as *federated identity management* services, allowing the sharing of the user's identity, along appropriate attributes of the user, to third-party websites. Thus facilitating a *single sign-on* (SSO) system. In this setting, the server validating the user's identity is known as the *identity provider* (IdP), which would be Facebook or Google in the above example. The third-party website is known as the *service provider*. This could for example be Netflix or Spotify. The idea of an SSO service goes beyond simply having an IdP facilitating a user authenticating towards a service provider. In fact an IdP may gather certified attributes about a user from multiple trusted issuers, and sign off on the user indeed being validated to have such attributed. This for example happens when Facebook validates that a given user has access to a specific email account, or phone number. While using an SSO makes things much simpler for a user, it also is a big privacy issue as an IdP now hold a large amount of the user's personal information, along with knowledge of whenever the user users this information and towards which service provider. Furthermore, it also means that large amount of trust has to be put in the IdP as they are would be able to impersonate any of their users towards any service provider. While such a thing is also possible for any attribute issuer (to a lesser extent) it becomes more of a problem for an IdP as they must be user-friendly enough that they can be used several times a day and since their only job is authentication. Hence becoming more exposed. Beyond this, simply using an SSO service can also lead to *traceability* and *linkability* of the user across the web. Traceability means that a user can be identified from the data resulting from using their eID. Whereas linkability means that it is possible for different service providers to find out if they have the same users. This can be an issue even if the user is authenticated using a pseudonym, since all it takes is one sharing of personal data, such as credit card information at a service provider, to de-anonymize the user. However, works like PASTA [AMMM18] and PESTO [BFH$^+$20] use threshold cryptographic to enhance the security of IdPs and limit traceability and linkability without reducing the usability. I.e. password based authentication can still be used and they remain compliant with solutions like OAuth and OpenID Connect. While they only focus on password-based authentication, they can be generalized to support multi-factor authentication [Fre21] and thus be used when multi-factor authentication is required for financial compliance, as e.g. in Europe according to PSD2 [PU15].

Decentralized Identifiers With the advent of blockchain technologies a lot of work has sprung up, trying to remove centralization from the management of eIDs. This is generally known as a Decentralized Identifier (DID) [SLS$^+$22]. The overall idea is that any kind of attribute provider issues a pseudonym to a user's blockchain account, reflecting a specific attribute. The user can then later use the pseudonym to prove certain certain attributes, or to simply get a reusable link to their pseudonymous identity at the same identity provider. However, it is clear to see that this basic construction is unfortunately not enough to ensure privacy, as again it possible to link the user across the internet (or blockchain) through their pseudonym. For this reason DID systems are starting to incorporate more advanced cryptographic constructions allowing users to anonymously prove that they hold a certain pseudonym towards a service provider (in order to facilitate authentication). Such a construction is known as a cryptographic *credential*.

Camenisch and Lysyanskaya [CL01] were the first to show a fully self-managed solution allowing users to prove their identity has been certified by a trusted provider, in an anonymous manner. Their credential construction affords validation of issuance from a trusted authority, while allowing the user to anonymously use it and preventing anyone who does not know the user's key[9] to impersonate it. However their construction did not allow the validation of arbitrary attributes. Something which is needed in many financial situations. Consider for example the case for loan or insurance issuance, where the customer's financial situation or health status has to be validated. Classically these must be provided as signed physical documents from the customer's attribute provider (such as credit bureaus), but the line of work on credentials, known as *attribute based credentials* shows how to achieve this in the digital sphere [CL04,San20] with cryptographic security and privacy guarantees. It was later shown how to compute arbitrary predicates on the certificated attributes [CG08]. Further development of such schemes into commercial products have been done by both IBM with their Idemix framework [CV02] and by Microsoft through U-Prove [Paq13]. The underlying primitives have even been taken up by standardization frameworks such as W3C [SLC22]. Still, despite such commercial traction, widespread adoption is still lacking. Moreover, in the context of DeFi systems, decentralized versions of anonymous credential schemes [GGM14,CDD17,ACC+20,DGK+21] have been proposed.

One could imagine that the requirement for self-managed private keys could be the reason that such approaches lack adoption since the regular news bulletins of people having lost their cryptocurrency keys, show that self-administered key management is not for the general public. However, multiple solutions based on threshold cryptography can be used to securely store keys under a client's password [CLLN14,JKKX17]. A more likely explanation might be the need of existing attribute providers to completely change their work-flow and systems, without any direct financial, legal or customer requirements.

Deploying Privacy Preserving Identity Management Fortunately, recent research have shown how PETs can be used to get certified attributed from issuers without modifying existing infrastructure, when such attributes be retrieved from the provider through TLS-secured connections. The Town Crier system [ZCC+16] shows how to construct certified attributes using secure hardware (like Intel SGX and using a TLS connection with a trusted provider). Concretely, they discussed how such certified attributes could be relayed to smart-contracts to allow more advanced decentralized user-attribute validation. Later, DECO [ZMM+20] then showed how to remove the need for secure hardware and replace it with MPC while achieving the same goal. However, they extended their construction to also integrate with zero-knowledge proofs, allowing clients to construct certified proofs of arbitrary *predicates* on attributes from any provider, trusted through a TLS certificate which provides online access to the user's attributes. This could for example include a bank providing online banking access, where a user would then be able to construct a proof that they hold a bank account with e.g. at least \$20.000. If the user's government provides an online residency portal, then

[9] Allowing the user to fully control the use of their credential through a single key can be conceptually advantageous.

it could also be used for the users to prove that they legally reside in a given city in a given country without leaking their exact address.

The CanDID system [MMZ$^+$21] shows how to fully realize a DID system with legacy support though either Town Crier or DECO. This is achieved through the usage of an MPC committee that validates legacy identity data and constructs a zero-knowledge friendly credential. Based on this credential the user can then prove arbitrary predicates on their attributes towards any provider.

Using attribute based credential allows the construction of fully private identity and attribute-based systems. However, in some situations full privacy might be undesirable, we would rather want to privately validate whether transactions are permissible based on attributes or identity, for example by ensuring that the identity of the credential holder is not on a deny-list. Kohlweiss *et al.* [KLN22] showed that such a system can efficiently be constructed on top of credentials. The construction allows an auditor to specify any predicate on the attributes in a credential, where the identity of the credential holder gets leaked if the predicate is fulfilled. Such conditional privacy leakage could prove tremendously helpful in fighting money laundering as we discuss next.

3.2 Anti Money Laundering

Money laundering is the process of concealing the origins of money, such as financial gains from drug trafficking or other serious crimes, by changing its origin to a benevolent source. This is because criminals must acquire many services and goods in the regular economy: put simply, most luxury car dealers don't accept briefcases full of bills. Money laundering is a huge problem in the financial sector: the estimated amount of laundered money is at the level of 2-3% of the national GDP in the US alone, excluding tax evasion [RT04, Chap. 2].

Getting large amount of illegitimate cash into the financial system requires multiple steps and multiple accounts to avoid raising suspicion. Simply getting dirty money into the system is known as *placement*. A concrete and common approach for this is known as *smurfing*, where multiple legal people deposit small amounts of money for a criminal, with the promise of earning a small amount as a kick-back. After a period of time the smurfs move the money out of their accounts (minus their fee), by transferring to other accounts controlled by the criminal. If this process is done with small amounts, and the receiving party's account is not flagged, then smurfing is hard to identify[10].

Once the money is in the legal financial system, it needs to be mixed with legitimate transfers, to counter suspicions caused by the initial transfers. This involves creating reasonable and justifiable transfers among multiple accounts of multiple entities in a process known as *layering*. By setting up a layering scheme through multiple banks, in different legal jurisdictions, using different legal entities, it becomes almost impossible to trace the flow of money. This is because the involved banks are (reasonably!) not allowed to communicate

[10] This step is sometimes also realized through other means, such as deposits from cash-driven businesses such as laundromats or food trucks.

private account and customer information about the sender and recipient of a money transfer. After the layering, the money is finally moved out of financial institutions and into legitimate investments such as real estate or legitimate businesses. This last step is known as *integration.*

What banks do to counter money laundering. As banks cannot share account and customer information with each other it is extremely hard for them to trace dirty money during layering. To address this, banks use multiple approaches usually subsumed as Anti-Money Laundering (AML) techniques. For example, banks internally use a *suspiciousness* score for customers. It is based on a *base* score, which is derived from the meta information about the account and its owner. The score may be derived from e.g. age of the account/holder, amount of money in the account, expected income and nationality of the owner. Through transfers, the base score is then updated, e.g. based on the score of the account a transfer goes to or comes from if both sender and receiver account are held by the same bank. If they instead are held by different banks, then metadata such as the amount of money going in/out and the frequency of the transfers is used in updates.

Finally, banks do have one common tool in measuring the suspiciousness of transfers, and that is a common, yet secret, grey list. This grey list contains accounts that have been deemed significantly suspicious, but for whom no provable money laundering has been identified yet. A transfer to or from a grey-listed account significantly increases the suspiciousness score. At certain time intervals, the suspiciousness score of an account is checked against a certain threshold and if the score is too high, then it gets flagged for manual[11] inspection.

What can banks do? Due to GPDR and other privacy laws, it is not possible for banks to directly share meta-information about accounts or its owner without their consent. Furthermore, if a bank finds a flagged account it believes is engaging in illegal activities then when informing authorities, it must be able to *explain* to said authorities how they came to this conclusion. Hence the bank's judgements must be auditable by a third party. If the conclusion depends on data received from other financial institutions, the bank must be able to point to this data and the third party must trust it as well. While data from banks from within the same legal framework (the EU, USA, etc.) is usually considered as valid, data from international banks, in particular those from countries with a history of corruption, has less trustworthiness.

Implementing sufficient and efficient AML techniques is also difficult due to the quantities of information involved. AML technologies should ideally be scalable to include all transactions and accounts. At the same time, even a limited AML technology which only covers cross-country transfers or an arbitrary subset of accounts, could still make a substantial dent into the suspected large amount of money laundering currently going unnoticed.

[11] In practice it turns out that about 95% of automatically flagged accounts are false-positives.

Cryptography and AML. The conjunction of AML and MPC is new and the main bodies of work on the topic are by Zand *et al.* [ZOP20] and Egmond *et al.* [vERS21]. Zand *et al.* show how computation on secret data can be used to notify an auditor of suspicious behavior. Egmond *et al.* show, in collaboration with multiple banks, how to use additively homomorphic encryption to obliviously update risk scores, and eventually (with consent from collaborating banks) decrypt the risk scores and flag accounts and customers appropriately.

However, related to this is the area of auditability of confidential transactions. As for example discussed by Tomescu *et al.* [TBA+22] where users are given a limited monthly "anonymity" budget. This budget is a certain amount of currency they are able to transfer anonymously per month. However, transfers surpassing this amount, is subject to deanonymization and clearance by a trusted auditor.

Finally, we note that a survey of real world concepts using PETs to combat financial crime has been conducted by the Future of Financial Intelligence Sharing consortium [Max21]. Unfortunately, many of their mentioned solutions require a trusted party to be involved.

As mentioned above, AML in centralized banking is challenging as the transaction graph is hidden due to e.g. privacy regulations. However in the decentralized finance space, such transaction graphs are usually visible. This is why most popular cryptocurrencies, such as Bitcoin, Ethereum or Cardano, are only pseudonymous and not anonymous[12]. Cryptocurrency exchanges such as Coinbase and Binance allow to turn large amount of cryptocurrency into Fiat currencies. These exchanges are required by law to enforce know-your-customer (KYC) rules. Through the help of transaction graph analysis firms such as Chainalysis, it has become hard to launder money using pseudonymous cryptocurrencies.

Researchers have also proposed mechanisms to enforce AML even if transactions are kept private. This includes using an escrow system where anonymity and privacy can be broken in case suspicious activities occur, such as transfers to or from an account known to be used by criminals [ST99,NVV18,BG20,DGK+21]. Such escrow mechanisms does not necessarily imply the usage of a trusted third party, as the data for escrow activities can e.g. be shared using Threshold Secret Sharing. Another approach is to specify a small budget per client which they can use every month for anonymous payments. After the client has made more transactions than covered by this budget, any future transactions can be traced [WKCC19,TBA+22]. Although seemingly a good compromise between privacy and security, this does still pose a risk to smurfing.

[12] We note that there exist privacy-focused blockchains like ZCash, Monero, or Dash that hide the transaction graph. Moreover, one can build private transactions on top of non-privacy focused blockchains using e.g. Zether [BAZB20a] that leverages encryption and ZK proofs. Finally, mixers such as Tornado [PSS19] take transfers from many users and put them into a holding account, from which they can later be transferred to the intended recipient.

4 PETs and the law

Usage of "classical" cryptography such as (public/private key) encryption, MACs, hash functions and digital signatures have been prevalent for decades and have so found their reasonable places within the legal framework of nations. For example legally speaking robust encryption schemes, used with long, high-entropy keys, provides a reasonable measure to secure sensitive data [Eur16, § 83]. However, besides affirming the security provided by cryptography, the law can only be used to deteriorate the security cryptography offer. This can occur through the usage of back-door mandates, forced usage of insecure parameters or the forced assistance in by people and companies to break circumvent encryption as is for example the case in India [MoLA00, Sec. 69]. This is not something new, and goes back to 90's, where export restrictions of highly secure encryption schemes were in place in the US [DL05]. Even more concerning than imposed weaknesses in underlying cryptography, is the legal issue of forced decryption. While this has generally been an issue for individuals [CP18], it has also become an issue for companies. Apple was for example mandated to decrypt a user's iPhone the San Bernardino terror attack of 2015 [Gro16]. Compelling a company to decrypt data should in itself not be regarded as a problem for PETs, as such cases already occur in non-private computation. However, this does become an issue in situations where PETs could be used to carry out transactions, which would otherwise not be possible due to their sensitive nature. In such a case, legal requests could also become preemptive requirements. Thus meaning that even if a company itself might not be trusted to not behave maliciously, it could be compelled to do so by its government. This is concretely a problem when entities in distrusting countries need to collaborate. One entity might trust another one to *not* behave maliciously, perhaps due to the public backlash of getting caught actively cheating[13] However if an entity acts malicious due to a governmental mandate, then such a backlash will be practically non-existing. Thus such legal potential could require the use of the strongest possible models of security.

While law could be used to break the security of PETS in certain weaker models, PETs can also help keeping the law. An example of this can be seen in the need for different governmental institutions or law enforcement agencies to share personal data in order to e.g. thwart terrorist plots. The need can be as simple as checking whether the same individuals are present in two different databases, and if so share relevant data. However, doing so without the aid of PETs would require leaking the individuals present in at least one of these databases. This goes against privacy laws, and in jurisdiction such as the EU, This would require explicit consent by all people in the database. While lawful sharing of personal data within law enforcement agencies might be possible in certain situations, such as sharing the list of publicly convicted criminals, this is not possible when it comes to other potentially relevant databases, such as people

[13] Certain flavours of MPC [AO12] can for example be used to ensure that if someone tries to cheat, the honest parties will get a publicly audible proof of this, which can then be release to the appropriate authorities.

with a record of mental illness or people with gun permits. A study of how to use MPC in such cases has been thoroughly studied in both a cryptographic and legal framework by Treiber *et al.* where different law enforcement agencies aim to find and share data about common entries in their databases, under approval by a judge [TMSgD22]. This framework could also prove relevant within law enforcement and financial institutions to ensure legality, such as law enforcement requests to the financial institution e.g. in relation to anti-money laundering (see Sec. 3.2). Thus PETs can be used to ensure the rights of citizens when the law gets involved. However, the law can also hinder certain computations and model on certain types of data as we discuss in the following.

GDPR and MPC On May 25th, 2018 the General Data Protection Regulation (GDPR) [Eur16] came into effect in the EU. The law dictates how data concerning EU citizens, should be protected and handled, along with legal requirements concerning individuals' rights; such as requiring appropriate consent for data storage and computation, and the possibility of withdrawing such consent.

While the GDPR is an 88 page law document consisting of 11 chapters and a total of 99 articles, its general gist can be described from the a few terms:

Personal Data : Any information that relates to an individual, which can be directly or *indirectly* used to identify the individual. For example name, gender, social security number, religious belief, web cookies, etc. It should be noted that some pieces of data alone uniquely defined an individual, such as social security number, whereas as some pieces of data are just indirect identifiers, such as gender, town, religious belief. A single indirect identifier does not uniquely identify an individual, but combining several of these *may* uniquely identify the individual. The GDPR considers even a single indirect identifier as personal data.

Data processing : Any *process* performed on data. For example computing, storing, transmitting, deleting, etc.

Data subject : The legal person whose data is processed. For example a customer or a web page visitor.

Data controller : The entity who decides what actions will happen to personal data. I.e. the holder of such data. For example the employee at a company responsible for data storage. In practice a data controller will generally be consider a legal entity such as a company, who holds personal data. For example Google, Tesco, Die Bahn, etc.

Data processor : A third party that performs actions data on behalf of a data controller. For example cloud storage providers such as Amazon or Microsoft, or researchers computing statics.

Based on these definitions the GDPR requires that data gathering must be as *minimal* as needed and *measures* must be taken to ensure that personal data kept is up to date. Furthermore, any processing must be as *minimal* as needed, *transparent*, and done in a way that ensures *confidentiality* and *integrity*. Finally the data controller must be able to *demonstrate* compliance with the requirements. The GDPR has further requirements such as disclosure of breaches within 72 hours and in some cases the need for employee training and the appointment of a data protection officer. The requirements of a data controller is unsurprisingly much higher than for a data-processor. For example, a data controller requires consent from the data subjects for storage, computation and other actions on their data. These are not required by the data processor, although the data processor must still ensure that the data is protected and can still risk huge fines for failure to do so.

Relating the GDPR to PETs we see that the main question is whether any legal entity other than the data controller has access to personal data on its data subjects. This becomes a question of what "access" means. The GDPR states that if it is not *reasonably* possible recover the identity of persons based on the data in their possession, then they are not data controllers [SHD17, 3.2.1]. Furthermore, there is precedence suggesting that a server simply storing encrypted data, for which it does not hold the key, is generally out of scope of GDPR requirements [PAR18, Sec. II. D.]. Thus the legal requirements and scope of different entities participating using PETs on personal data comes down to the definition of *reasonably*.

For secure hardware is seems reasonable to assume that outsourcing the computation on personal data would either not be considered reasonable, *or* the provider of the secure hardware would be considered a data controller. E.g. for an execution on personal data on SGX on a server, either 1) the server and the initial data controller together would be considered a joint data controller (if SGX does not reasonably protect the data). Or 2) the initial data controller and Intel would be considered joint data controllers (as Intel is claiming their hardware reasonably secure).

In the case of secure computation, the situation becomes more unclear, firstly since it is undefined if collusion can be considered *reasonable*, and secondly since there is no single legal entity that can be held accountable in case of a breach. Thus there is a chance that the servers executing MPC would all be considered joint controllers on the set of all personal data which is computed on. This is called the *absolute* interpretation of the law. A less conservative view would imply that they become data processors, and thus need to adhere to general data safety and operational requirements, but do *not* need consent from the data subjects whose data they compute on, assuming they gave their data controller consent to carry out such a computation [ATV20, 2.2]. This means that MPC could be used as tool for proving data privacy by design. However, the general consensus is that if data is secret share or encrypted when computed on, in a way where a single malicious legal entity is not able to *reasonably* recover the personal data, then the data is no longer personal and out of the scope of GDPR [SHD17, 3.2.1] *if* the result of the computation does not *reasonably* allow deanonymization of the individual's whose personal data was used [HR22]. Crucially this applies, not just for a single computation but for the conjunction of all computations done on the data subject's personal data. This is known as the *relative* interpretation. Technically, the interpretation comes down to whether personal data is considered *anonymous* if no-one can reconstruct or if no single legal entity can reconstruct the data (up to cryptographic hardness).

More concretely, regardless the legal status of the different servers, when it comes to computing on personal data, the GDPR imposes other requirements in that 1) data subjects must consent to the *specific* computation to be carried out, at the time they give their personal data and 2) and the result of the computation must not be able to lead back to the data subject's personal data or identity. Thus according to the GDPR care must also be taken to only perform computations on personal data, where the result of the computation cannot lead to deidentifying the data subjects

One interesting exception to personal data under the GDPR is *pseudonomyzation*. Pseudonomization involves replacing identifiable parts of personal data with pseudonyms. It should not be confused with *anonymization*, which completely removes the coupling of personal data to a data subject. Pseudonomyzation is considered a reasonable approach to securing raw personal data, but at the same time is still considered personal data! Hence sharing and computation on pseudonomized data still make participating entities data processors at a minimum.

An example application of how to achieve utility from personal data without breaking privacy was demonstrated by Damgård *et al.* [DDN+16]. They constructed a scheme based on MPC where personal data held by a consultancy house, was used to compute credit scores (relative to other applicants in the same category) for load applicants, hence helping banks to give rational interest requirements and loan offers. Their concrete case was focused on Danish farmers, as they are not required to publish financial information about their business, and thus it is a challenge for banks to give reasonable loan requirements as they don't know how the specific applications compare to the general market. While not done explicitly the protocol of Damgård *et al.* could have allowed banks to give farmers loan offers without the farmers needing to disclose their financial situation to the bank beforehand.

Other laws Many countries have privacy laws but few have been studied in the relation to PETs and to keep the scope of this survey simple we have only covered GDPR. Furthermore, the scope of GDPR is in a sense the one of the strictest privacy framework when comparing e.g. with HIPAA and CCPA in the US.

PETs assisting businesses and governments Disregarding personal data, there are often other situations where companies, or even governmental instances might want to validate that the other party holds the information they claim they to do. This could for example be the case of mergers and acquisitions, where one company claims to hold some algorithms or machine learning model capable to perform certain actions, but they would of course not want to disclose this before the acquisition is complete.

A similar problem has also occurred during trials, such as the case of U.S. v. Michaud and U.S. v. Coplon, where the defendants were charged based on evidence gathered through the use of proprietary and secret law enforcement software. However the defendants defense wanted to inspect the software to ensure that it did not have faults and that it did not violate the Fourth Amendment (unreasonable searches and surveillance). In situations like these, zero-knowledge proofs could be constructed and used to convincingly validate the necessary constraints, without leaking proprietary information [BCGW22]. The same is true when the proprietary information is not a program, but classified data instead. If the data has been certified by an authority, then zero-knowledge proofs could be used to validate such data against a public predicate without leaking any content. Such zero-knowledge proofs could furthermore be augmented to allow for *public verifiability*, meaning that *any* external party can validate the proof. By combining this with a blockchain, such proofs could remain publicly accessible for anyone to validate. The possibility for public verifiability is for example highly relevant in the case of U.S. secret laws. These are laws whose content, or even existence, are classified. Such laws for example contain constraints on how law enforcement are allowed to circumvent security measurements to access people's personal data, without warrants. In court cases it is useful for the public and jury to be able to validate that the constraints in the secret laws have been obeyed by law enforcement [GP18].

In the finance sector such approaches could also be useful, for example for banks to prove that their AML software fulfill the minimum legal requirements, without leaking their proprietary algorithms, see Sec. 3.2. Such proofs could also be useful to post publicly for public verifiability, allowing (potential) customers to validate that their bank of choice is following legal requirements. Another example where this might prove useful, is in the situation of acquisition where information about the quantity, or demographic, of customers might be highly relevant to the price purchaser is willing to pay.

5 Digital Asset Custody

As the vast majority of financial transactions are automatically executed over vast inter-bank and payment networks, the signing and encryption of sensitive transaction messages require secret cryptographic key material that must be carefully managed to prevent impersonation, theft and forgery. In traditional finance, this is implemented at the device instance level with hardware security modules (HSM). These generate, store and use sensitive key material locally; any signing or encryption operations are performed strictly on the HSM such that the key material never leaves the device during usage. Thus, physical access to HSMs must carefully guarded and its operation must adhere to rigorous standards, such as those set by the Payment Card Industry (PCI).

HSMs are widely used in payment networks, for example, when the EMV protocol [BST21] for credit card transactions requires the online verification of a customer PIN entered at the point-of-sales (POS). In this case, the PIN must be forwarded in authenticated and encrypted form to the card issuer for verification. However, as the POS supplier does not have a direct relationship with the card issuer, the PIN is forwarded via hops between intermediaries, where only neighboring intermediaries have established shared keys for encrypted communication; Such keys for encryption are managed by HSM's. However, security which HSM provide also means they are costly to acquire (tens of thousands of dollars) and to operate, as even the most basic maintenance items such as firmware upgrades must be conducted on-site and involve multiple, redundant operators with requiring specific access authorizations. Furthermore, strict PCI standards on the design and operation of HSM's make changes to the underlying cryptography very difficult, as custom HSM firmware are generally not permitted.

Threshold signatures computed by MPC committees have emerged as an alternative to HSMs in the context of digital assets, which have deployed non-standardized signature schemes such as ECDSA based on non-standard elliptic curves (e.g. secp256k1 in Bitcoin). Whilst HSMs have traditionally played the role of secure signing, they are difficult to customize and adapt in an industry with rapidly evolving cryptography on ever newer blockchain protocols. Furthermore, since HSMs represent a single point of failure, their installation and physical access control requires expertise, not readily available to fast-moving cryptocurrency startups and institutions. MPC can be offered as Software-as-a-Service or cloud solutions. Suppliers of such digital asset custody solutions include Unbound[14] [Coi21] and Fireblocks [Fir21], which have implemented highly performant threshold signing algorithms [LN18,CGG+20,DJN+22]. In contrast to HSMs, MPC servers can be run on standard virtual cloud instances, offering a high level of elasticity for both (1) performance and (2) key security; (1) Signing of independent transactions to authorize transfer of digital assets is easily parallelizable and (2) the number of MPC servers can be scaled to increase decentralization of the key material. MPC instances can also be easily upgraded to perform new cryptographic tasks as there is a lesser need for standardization and hardening of individual MPC devices given a lack of a single point of failure. Furthermore, using MPC allows for easy and simple portability and back-up of key material. Another benefit of using this technology is employing the concept of *proactive security* for MPC protocols to periodically refresh their internal scret states in order to recover from momentary security compromises. In particular, this has been implemented in the context of MPC-based key management for digital asset custody [CGG+20].

We note several critical differences between traditional finance and decentralized finance, which explain the quicker adoption of MPC by the latter. In traditional finance, transactions can generally be revoked; if an individual HSM is compromised and its key material exposed, the log of the attack can be traced to an individual device. The post-mortem analysis can thus establish a clear breach event and entry point, providing clear evidence that a transaction was authorized with stolen keys for its later revocation. In contrast, transactions in digital ledgers can generally not be reverted; they are final. For this reason, cryptocurrency exchanges and custodial services will operate "cold" and "hot" wallets; the former are generally disconnected from the business logic and re-

[14] Acquired by Coinbase, Inc.

quire human authorization to move funds. The latter hold a fraction of the total assets, but are automatically triggered to produce valid digital signatures when prompted by the customer-facing application.

We consider the adoption of MPC for the application of digital asset custody an illustrative one; the improved updateability and tuneability of both performance and security in MPC (over classical HSM's) is a major selling point for emerging applications, and we anticipate MPC to spread to other key management domains in finance over time.

6 Markets & Transaction Settlement

In this section, we provide an overview of privacy-enhancing technologies in market and settlement applications.

In financial markets, there is a need for auctions and markets with *fairness* guarantees, as rational actors are incentivized to collude and front-run honest parties, if the true valuation or trade-intent of the latter is revealed. Here, we first consider the *traditional finance* setting (§6.1), where the settlement of transactions is handled by traditional, asynchronous settlement processes. In the presence of a *public ledger* (§6.2), settlements occur *synchronously* and *immediately* after a transaction is completed. Such a mechanism also permits the "netting" of inter-bank payments §6.3) to minimize the liquidity requirements on participating banks; this must done with PET approaches, since the public ledger would otherwise expose all individual payment orders, a clear breach of consumer privacy.

Finally, we highlight approaches to achieve bidder privacy in *demand-response* electricity markets (§6.4), which coordinate the remote scheduling of power consuming devices to match forecast production from sustainable production sources; the submission of granular device-level information to an auction in the clear can reveal the activity and presence of customers at home, violating their privacy.

6.1 Markets in Traditional Finance

The first setting reflects an idealized view of *traditional finance*, where accounts and balances are generally maintained by financial institutions and considered private. Here, the settlement of auctions, or exchange transactions, occur asynchronously; whilst the *counterparty risk* from defaulting on obligations implied by pending transactions is real, we consider it an orthogonal challenge addressed in the public ledger setting (§6.2, §6.3). Clearing prices and executed volumes are considered public information as this information is forwarded to institutions executing the settlement. This first setting intends to achieve resilience against dishonest venue operators and participants attempting to obtain a financial gain from unwarranted information flow. Communication between parties generally assumes direct, authenticated channels, implying the knowledge of identities and

Setting	Applications		Privacy	Benchmarks	PET	Works
Markets in Traditional Finance (§6.1)	Distributed sealed-bid auctions	Single-sided	Bid privacy	○	MPC	[FR96], [HTK98], [Cac99], [NPS99]
		Double-sided	Bid privacy	◐	MPC	[BDJ$^+$06], [BCD$^+$09]
		Public verifiability	-	○	HE+ZK	[PRST08]
	Distributed Dark Pools	Continuous matching	Order privacy	◐	MPC	[CSTA19]
		Periodic matching	Order privacy	●	MPC	[CSTA21], [dGCP$^+$22]
			Order privacy	◐	FHE	[BDP20]
		with many assets	Order privacy	●	MPC+HE	[CSTA21]
		with many servers	Order privacy	●	MPC	[dGCP$^+$22]
		Public verifiability	-	○	HE+ZK	[TP07]
Markets on Public Ledgers (§6.2)	Decentralized sealed-bid auctions	Single-sided	Order privacy	○	MPC+ZK	[BHSR19], [DGP22], [GKS22]
	Privacy-preserving Decentralized Exchanges	Futures Periodic matching	Net position privacy	◐	MPC+ZK	[MNN$^+$18]
		Periodic matching	Balance privacy Partial order privacy	◐	MPC+ZK	[GVJR22]
			Order privacy (Balance privacy)	◐	MPC+ZK	[BDF21], ([BCDF22])
		Intent-based order matching	Balance privacy Order privacy	◐	ZK	[BCG$^+$20], [XCZ$^+$22]
			Balance privacy Order privacy	●	WKA	[NMKW21]
Settlement on Public Ledgers (§6.3)	Liquidity preserving inter-bank netting	-	Payment privacy	◐	ZK	[CYDC$^+$20]
			Payment privacy & Robustness	◐	MPC+ZK	[DCMA22]
Demand-Response Markets (§6.4)	Distributed auctions for demand flexibility	Double-sided (Single buyer)	Device power constraint privacy	◐	MPC	[AACM16], [ZGND22] [GZN22]

Fig. 2. Auctions & Markets: no benchmarks (○), preliminary benchmarks (◐), benchmarks achieving real-world performance with traditional market parameters (●).

a pubic key infrastructure.

Distributed Sealed-bid Auctions. One-off, sealed-bid auctions are frequently performed in the sale of frequency-spectrum rights, government contracts, real-estate and other private items, such as art. In open-cry, single-sided auctions, bids are broadcast publicly until no additional bids are made. However, the leakage of bids or orders can be exploited by the adversary for financial gain. In Vickrey auctions, where the winner pays the price submitted by the second-highest bid, the auction operator collecting the submitted bids is incentivized to collude with other bidders to increase the second-highest bid price and maximize auction fees. The auction operator must also be *trusted* to not reveal anything about submitted bids, in order for all bidders to submit their *true valuation*. The advent of the public, commercial internet coincides with the first protocol proposals which permit the execution of one-time, sealed-bid auctions by distributing the role of the auction operator, thereby removing the need for a trusted auction venue.

Franklin et al. [FR96] propose a one-sided auction protocol, where the auction venue is distributed amongst multiple servers; bidders to submit their signed bids as *verifiable secret-shares* (VSS) to participating servers during the bidding phase. Subsequently, bids and signatures are jointly reconstructed by all servers, upon which all bid information becomes public. Verifiable secret-sharing ensures that bidders submit well-formed bids. As long as a single server is honest, the reconstruction of bids cannot occur before the end of the bidding phase. However, it is often important to protect the privacy of bids, even if they are not successful; the valuation may reveal a bidding strategy for another, related auction.

In the work of Harkavy *et al.* [HTK98], MPC is deployed to maintain the privacy of all submitted bids; only the winning bid is made public. Naor *et al.* [NPS99] propose a variant of MPC with garbled circuits, to reduce the rounds of communication, thereby improving performance. Cachin [Cac99] builds a purpose-built, privacy-preserving protocol, which permits the comparison ($>$) of *prices* between two parties with the help of an untrusted third party. An auction determining the highest bidder is then constructed from this primitive.

The first work to demonstrate the feasibility of privacy-preserving (one-time) *double-sided, sealed-bid auctions* was proposed by Bogetoft *et al.* [BDJ+06]. Later secure auctions were used in practice [BCD+09], to facilitate the auctioning of sugar beet delivery contracts in Denmark. Concretely, farmers producing sugar beets hold contracts which represent an obligation and right to deliver beets to the the (single) Danish sugar beet processor Danisco. The trading of such contracts permits the reallocation of contracts to the most efficient producers, but such an exchange run by Danisco would permit it to learn information about the economic circumstances of producers, potentially compromising sugar beet farmers during contract negotiations. The matching and determination of a price computation from 1229 buy and sell orders was achieved in approximately half an hour by an MPC committee of 3 servers; a throughput volume sufficient for a one-time auction, but unacceptable for traditional electronic security exchanges. It should be noted, however, that cryptographic techniques have improved drastically since the work first appeared. Such an auction would run much more efficiently today.

Publicly verifiable auction operators are proposed in the work of Parkes *et al.* [PRST08], a weaker alternative to implementing the auction operator with MPC; instead, the dark pool venue is still operated by single entity, but provides cryptographic proofs that the auction algorithm is performed correctly by the venue operator. Whilst this prevents the auction venue from *manipulating* the correct evaluation of auction bids, it does not prevent the auction operator from leaking bid information to malicious participants.

Distributed Dark Pools. Dark pools have gained adoption in traditional finance as venues where submitted orders are not publicly accessible, thus minimizing the potential price impact caused by signalling *trade intent* to front-running market participants. As is the case with auction venues, the dark pool operator must be trusted to not *share* the order flow information with malicious participants, a trust assumption that is frequently violated in practice (Table 1 in [CSTA21]), motivating the need for distributing the role of the venue operator.

Whilst the matching of orders in a *secret* order book is a natural domain of MPC, we observe that current proposals illustrate specific configurations and architectures for distributed dark pools that may or may not match throughput observed in traditional, centralized dark pool markets. In particular, the choice of auction clearing algorithm remains a deciding feasibility factor. Whilst secret-sharing based MPC schemes permit secret computation with n participating servers, the runtime bottleneck generally lies in the amount of *communication* that is required between servers. This is because MPC has a specific model of defining computations, and certain auction clearing algorithms can be realized with less communication overhead in MPC than others.

In the case of auction clearing algorithms, which must compute a *clearing price* from current bids and sell orders, the *sorting* thereof by *price limit* induces many *comparisons* $(>, <, =)$ between secret-shared values, which in turn imply sub-protocols generating the majority of communication cost. Alternatively, order matching based on volume only (where prices are determined by third party price feeds) can greatly accelerate throughput, as the expensive clearing price evaluation is not required. Furthermore, whether orders are processed *continuously* or *periodically* also greatly affects the real-world applicability of the following distributed dark pool protocols.

Continuous Double Auctions with MPC: A recent line of work by Smart *et al.* [CSTA19,CSTA21,dGCP+22,dGCSA22] implements and examines real-world, double-side auction algorithms. In the initial work [CSTA19], continuous double auctions (CDA) are implemented in a distributed fashion across servers running an MPC. Continuous double auctions maintain a limit order book (LOB) where buy and sell orders are ordered by ascending and descending price respectively; *each incoming order* is matched against one or more LOB orders if it crosses the "spread" between best buy (or sell) prices; its remaining volume is then inserted into the LOB. It is also the most *expensive* exchange algorithm since (1) each single order must be matched against m other fulfilled orders and (2) its remaining trade volume must be inserted into a (potentially large) order book of N size. Benchmarking such an algorithm requires specifying the *expected* state of the order book, given the sensitivity of CDA run-time on (1) the average number of matched orders m; and (2) the expected order book length N. In the dark CDA implementation of Cartlidge *et al.* [CSTA19], run with 3 servers and Shamir-sharing based MPC, a worst-case throughput of $34 - 43$ orders per second for LOB parameters $m = 3$ and $N \approx 30$ is achieved. This work demonstrates that distributed CDA with MPC cannot yet match the throughput volumes of traditional CDA venues[15]. In contrast, *periodic* order matching greatly improves the performance of distributed dark pools.

Periodic Double Auctions with MPC: Periodic auctions in the dark pool setting implemented with MPC promise throughput that match those of traditional dark pool markets, as shown in these works by Cartlidge *et al.* [CSTA19,CSTA21]. In periodic auctions, limit orders are submitted during an open auction period

[15] In their work, the runtime of MPC pre-processing is neglected, which represents a non-trivial "hidden" computational cost that can be performed during "offline" hours or outsourced to dedicated pre-processing workers; pre-processing generally does not limit the maximum, sustainable throughput of MPC.

after which a clearing price is computed during the clearing phase, which maximizes the volume of matched orders (unmatched orders are carried over to the next round). In contrast to CDA algorithms, where orders are processed individually against a potentially large order book, periodic auctions only need to compute a single clearing price for the entire batch in a given period. In fact, real-world order execution throughput has been achieved with a realistic number of asset pairs.

In [CSTA21], the London Stock Exchange Group's *Turquoise Plato Uncross*, a widely-used traditional dark pool supporting thousands of assets, is implemented with promising results. Based on volume observed on the real-world Turquoise Plato Uncross venue, it is assumed that order book clearing occurs at most every 5 seconds, where at most 2000 newly input orders must be processed across an asset universe of 4000 financial instruments. This throughput was successfully handled by smaller MPC committee sizes of 2 (dishonest majority) and 3 (honest-majority), but required multiple MPC instances, each handling orders trading a small subset of all assets (Figure 3). For example, $\sim$ 280 MPC committee instances are each randomly assigned 16 assets in each round by a *gateway* engine. This gateway periodically reassigns asset subsets to new MPC instances in order to break potential linkages between orders across time periods. We note that auction algorithms are not entirely *oblivious*. Indeed, the direction of orders are leaked in [CSTA19,CSTA21], whilst volumes and order limits remain private.

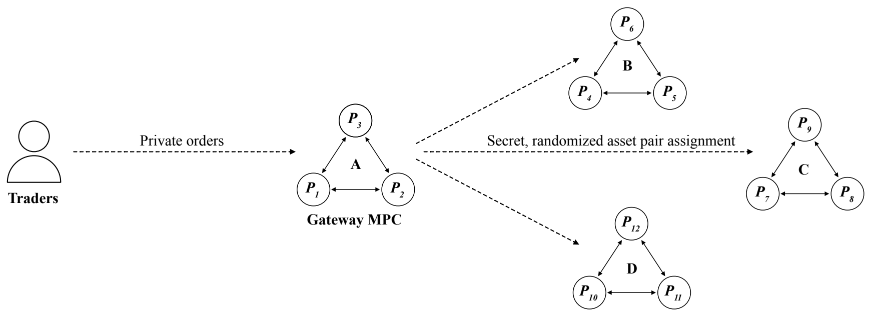


Fig. 3. In [CSTA21], a gateway MPC (A) distributes inbound received orders across multiple MPC committees (B,C,D) to improve order clearing throughput. MPC servers never learn the asset pairs its committee is assigned in each round.

Complementary follow-up work by Da Gama *et al.* [dGCP+22] and [dGCSA22] both focus on (single-asset) privacy-preserving *volume matching*, which enables further performance gains since the clearing prices are determined by an external reference price; [dGCP+22] introduces an improved MPC volume matching algorithm which permits dummy orders and hides the trade direction. [dGCSA22] scales volume matching up to MPC instances consisting of $\sim$ 100 servers and shows the economic costs associated with operating such a single server in a MPC instance of up to 100 servers to be below ~ 0.10 USD and ~ 0.025 USD for computation and network communication respectively in each auction round; the negligible cost demonstrates the feasibility of *market participants* contributing to the distributed operation of dark pool venues.

Periodic Double Auctions with FHE: JP Morgan has demonstrated initial results in work by Balch *et al.* [BDP20] to realize dark pool venues where the venue operator is not distributed, but computes the periodic volume matching over data encrypted under a jointly controlled public key; here, the secret encryption key material used in the *threshold fully homomorphic encryption* is held in secret-shared form by all participants (Figure 4). While the computation can be done by one party on the ciphertexts (not knowing their plain values), the participants then later take part in a so-called distributed decryption protocol which reconstructs the outcomes to the venue operator. Whilst [BDP20] benchmark periodic volume matching implemented with threshold FHE, the omission of the *partial decryption sub-protocol* complicates the evaluation of its performance. Still, FHE offers an alternative approach to secret sharing-based MPC which promises competitive performance; fewer communication rounds are required, since computation is performed locally by the dedicated venue operator over encrypted data, although local computation (over encrypted data) is more costly.

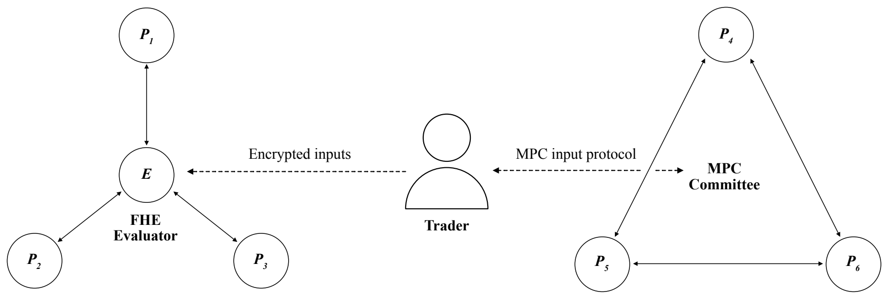


Fig. 4. In [BDP20], market auctions are implemented with fully homomorphic encryption (FHE); here, the encryption key is jointly generated by key servers (P1-P3), but the FHE evaluation is solely performed by the evaluator, who never learns the plaintext of the inputs or intermediary results. Decryption of the FHE output requires interaction with all key servers. In contrast, private computation with MPC in [CSTA19] requires interaction amongst MPC servers (P4-P6) for each (multiplicative) operation.

Publicly, verifiable dark pool operator: Similar to verifiable one-sided auctions [PRST08], a weaker notion of order privacy for dark pools is proposed in the following work by Thorpe *et al.* [TP07], where the venue operator only reveals a homomorphically encrypted order book to traders; the operator itself, however, maintains the encryption key and can thus compute over the order book plaintext. Each update to the public, encrypted order book triggered by a submitted order is accompanied with a zero-knowledge range proof generated by the operator; any public party can locally re-compute the claimed update over encrypted values and verify zero-knowledge range proofs that guarantee that the plaintext values lies within certain ranges, thereby enabling verification of comparison statements between encrypted values. This system ensures the correctness of each order book update without revealing the order details themselves. [TP07] implements the CDA algorithm in a publicly verifiable manner; as in [PRST08], this approach does not prevent any misuse of the order information held by the operator.

6.2 Markets on Public Ledgers

The advent of public ledger protocols [GKL15,KRDO17,GHM+17] resulting from permissionless participation of servers across the public internet promises a truly "server-less" system of transaction settlements, no longer dependent on any single trusted intermediary. The state of the ledger is public and its integrity is publicly verifiable (by any online party) by local verification of all previously finalized transactions sequenced in form of a append-only list or *blockchain*. The realization of a global transaction history also implies a Turing-complete *state machine*; smart contracts represent user-deployed programs run on blockchain protocols that, in addition to custom ledgers [ERC20, ERC721], can realize decentralized auctions or decentralized exchanges (DEX), which forgo the need for trusted venue operators. In contrast to traditional finance, market applications in the public ledger setting offer instant settlement; any market application implemented with smart contracts instances permits the simultaneous evaluation and settlement between participants. Despite scalability challenges arising from the vast number of participants running the blockchain backbone protocol, the promise of instant settlement would allow the mitigation of counter-party risk, a real cost to transactions conducted in traditional finance today.

However, the public verifiability of a public ledger also introduces novel challenges for financial applications; account balances are public by default and leak information about submitted bids, trades or margin positions; the latter must be backed by valid balances. In decentralized finance (DeFi) [WPG+21], front-running is indeed rampant in decentralized exchanges (DEX) [TCS21], since pending transactions leak trade intent to the adversary which can precisely order and inject transactions to execute optimal front-running strategies. Thus, proposals have been made to implement private balances on public ledgers with publicly verifiable, non-interactive zero-knowledge [SCG+14,BAZB20b]. However, a privacy-preserving ledger (even with standard smart contract support) is generally *not sufficient* for privacy-preserving financial applications such as exchanges [ByCD+21].

Privacy-preserving ledgers generally complicate the realization of *smart contracts*, since these must verify and update account balances known only to its *owners* according to an agreed-upon transition logic. For decentralized exchanges implemented in the privacy-preserving ledger setting, this requires the presence of a secure multiparty computation instance, to which users can privately input their trade orders and private balances; the MPC then computes an updated DEX state and private balances, which are then updated on the ledger (Figure 5). Enforcing consistency between the *secret*, internal MPC state and *private* account balances on the ledger requires protocol design advances illustrated in the subsequent paragraphs. We emphasize that counter to popular belief, zero-knowledge is not sufficient to realize universally expressive, privacy-preserving smart contracts, as the witness (or secret state) for decentralized privacy-preserving applications are partially held by separate, distrusting parties; instead, function evaluation over private inputs from separate parties and secret-shared data is the natural domain of secure multiparty computation.

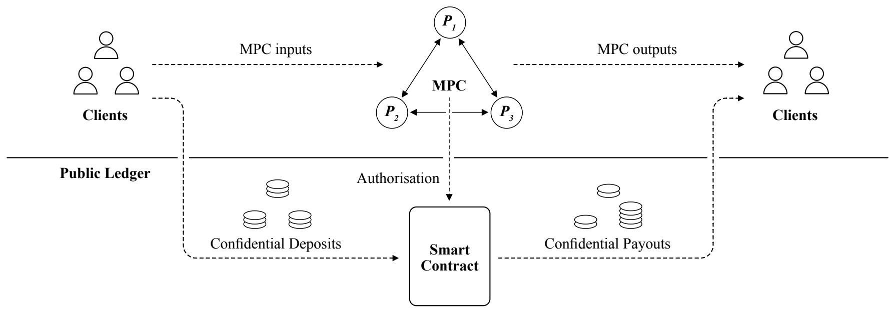


Fig. 5. We sketch the architecture of privacy-preserving smart contract applications in MPC with instant settlement on a (confidential) ledger; clients provide input parameters to the MPC instance, and forward financial deposits to a smart contract in a confidential manner. The MPC privately returns computation output to clients, but also authorizes a new financial distribution which is paid out to the clients by the smart contract functionality.

We note there are privacy-preserving smart contract proposals which shield *private data* [SBG+19,SBBV22] held by individual users or *private contract logic* [BCG+20], but such techniques are generally limited in their expressiveness. The work of Bowe *et al.*[BCG+20] only supports two communicating parties, and is not widely used to realize privacy-preserving financial applications.

Sealed-bid Auctions (with Instant Settlement). The first work by Bag *et al.* [BHSR19] to realize sealed-bid auctions specifically in the setting of public ledgers focuses on using the blockchain as a *communication medium* instead of a settlement layer; as a permissionless protocol, any party can anonymously post an arbitrary message to the bulletin board, visible to all other parties. For protocols with low communication rounds, this is a practical solution; in particular, the simplicity of evaluating single-sided sealed-bid auctions permits task-specific secure multiparty protocols which only require public message broadcasts. The SEAL [BHSR19] protocol proposes the use of a *anonymous veto protocol* [HZ09] requiring only two communication two rounds, that is then repeated once by auction bidders for *each bit* of their bid price, thereby removing the necessity an auctioneer role entirely. In its particular, the veto protocol of [HZ09] receives the private input $b_i \in \{0, 1\}$ for party $i \in [n]$, for each of the m bits representing the permissible price range; the parties learn the highest bid, bit by bit. Each execution of the veto protocol will thus publicly output 1 if one of the users submits a veto; thus, by repeating the veto protocol for each bit position, all participants receive the highest bid without revealing the prices of failed bids. [BHSR19] assumes participants to behave according to the protocol (semi-honesty).

This mechanism was later adopted and hardened by FAST [DGP22] to be secure against malicious participants not adhering to the protocol. Furthermore, [DGP22] introduces *guaranteed settlement* on the public ledger featuring privacy-preserving deposits. Participants are thus committed to execute the payment for their bids if these are successful during the auctions. Cheating participants are penalized by having their deposits slashed and reimbursed to other parties; non-interactive zero-knowledge proofs from all parties ensure that parties only submit a veto if it is consistent with their initial bid (in commitment form on the ledger); still, despite the privacy-preserving aspects of the anonymous veto protocol, privacy leakage occurs when the highest bidder learns when he overtakes the second highest bidder. The work of Ganesh *et al.* [GKS22] adopts a similar construction which is proven to be game-theoretically secure; it is rational to pursue the honest protocol despite any strategy chosen by other players. The work of Chin *et al.* [CEOS22] does not employ zero-knowledge proofs to shield commmitted funds for a single-sided, sealed-bid auction; deposits are sent to committed, yet undeployed contracts and are thus indistinguishable from normal Ethereum transactions, a similar technique used in Breidenbach *et al.* [BDTJ18] to commit inputs to smart contracts without revealing them to the front-running adversary. This approach only guarantees k-anonymity and relies on the presence of other, unrelated transactions.

Privacy-preserving Decentralized Exchanges. We note a number of recent proposals for privacy-preserving DEX applications in recent years; intent-based privacy-preserving DEX applications mirror the functionality of over-the-counter (OTC) venues (in traditional finance) and only require a public ledger, but do not scale well and are not widely deployed. Privacy-preserving and front-running secure DEX protocols generally involve private ledger deposits and perform the order matching in an MPC instance, as is the case in distributed Dark Pool proposals previously described in Section 6.1, but offer instant settlement following each DEX round (Figure 5).

Intent-based, privacy-preserving DEX: In the works of [BCG$^+$20] and [NMKW21], a simpler model of a decentralized exchange is implemented; a bulletin board functionality provided by a public ledger permits a "maker" to broadcast their trade intent. An interested counter-party or "taker" then directly opens an authenticated communication channel with the maker to jointly perform a privacy-preserving atomic swap on the public ledger [BCG+20]. Intent-based DEX protocols resemble over-the-counter models in traditional finance. [NMKW21] introduces a "witness key agreement" (WKA) construction which preserves the privacy of the maker's offer; the WKA allows a taker to establish a shared secret key with a maker which has posted its order in commitment form to the ledger. The key agreement protocol succeeds if the committed, private order fulfills a relation determined by the taker. This key permits subsequent anonymous communication with the maker to finalize the transaction.

Privacy-preserving Futures DEX: An interesting example of decentralized exchanges is illustrated by Massacci *et al.* [MNN+18], which realizes a futures exchanges modelled closely after the Chicago Mercantile Exchange; here, the *future obligation* (or contract) to buy or sell a commodity is traded. The net position of a market participant is the sum of both current liquidity balances and future obligations; importantly, a party holding a future to "sell" a given commodity, must always hold sufficient liquidity to acquire the respective commodity, as it otherwise would default on its contractual obligation. Thus, a net position that falls below zero must be liquidated to protect the counter-party of any futures contract held by the liquidated party. Achieving this in a privacy-preserving manner without *revealing the net position* of a party is the goal of the work of [MNN+18].

If the net position of a participant is revealed, price manipulations could be conducted with the explicit intention of forcing the liquidation of otherwise valid positions. Thus, [MNN+18] proposes a similar scheme to [SCG+14], where the net position of each account is committed in a cryptographic accumulator. The validity of each update to the account is proven in zero-knowledge, whilst the trading venue is executed in a MPC instance, similarly to the Dark Pool proposals in Section 6.1. [MNN+18] requires parties participate in the protocol for each account update, even if this means the liquidation of their own account. We note that the subsequent privacy-preserving smart contract framework instantiated with MPC and a confidential ledger [BCDF22] achieves the privacy guarantees of [MNN+18] without permitting users to block application liveness.

Front-running Secure DEX: A general motivation for privacy in decentralized exchanges is the front-running of DEX applications in Decentralized Finance due to public transactions and accounts in the default ledger setting; Despite offering instant settlement of trades and transactions, pending user input authorizations generally broadcast a users trade intent before their finalization. To this end, P2DEX [BDF21] proposes the first privacy-preserving decentralized exchange, which can operate and settle transactions across multiple ledger instances; clients submit orders to an MPC committee which computes the order matching and subsequently settles these on the respective public ledgers; since the trade inputs are private, front-running is mitigated. Follow-up work [BCDF22] generalizes this model to a setting with confidential accounts; here, all zero-knowledge proofs are moved *outside* the MPC computation, as computing such proofs *inside* the MPC remains generally unfeasible for real-world application. The work of Govindarajan *et al.* [GVJR22] realizes a privacy-preserving DEX in a similar manner; here, however, the actual order matching is computed in the clear of a smart contract over anonymized trade lists to accelerate the determination of a clearing price.

6.3 Inter-bank Netting on Public Ledgers

Inter-bank payment requests are currently submitted to the real-time gross settlement (RTGS) system managed by the central bank to update the accounts of sending and receiving financial institutions. In times of low liquidity, a bank may fail to honor *individual payment instructions*, as the liquidity requirement may exceed its balance and credit line granted by the central bank; a *gridlock* occurs, when a failed payment settlement prevents further payment instructions from being processed. Given the large payment volumes processed each day, liquidity saving mechanisms are implemented which settle payment instructions on a *netting basis* (Figure 6).

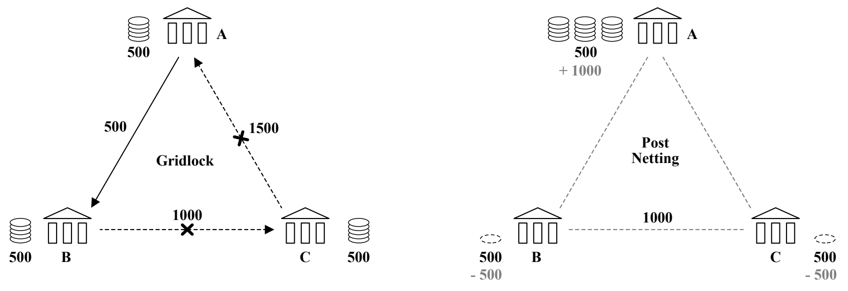


Fig. 6. We adopt a netting example from [WXF+18]; processing of individual payment orders may fail due to a lack of liquidity (left), as balances must remain positive following execution of each individual payment. Netting relaxes this constraint; balances need only to be positive following execution of all payments orders (right).

Recent work has proposed *distributing* the role of the RTGS operator with a public ledger protocol, whilst implementing efficient netting protocols with smart contracts [WXF+18,NYS+18], therby increasing system resiliency as the operational liability burden on the central bank operator today is very high. Whilst the aforementioned works implement inter-bank netting of queued payments, the nature of public ledgers means that payment instructions are revealed to parties participating in the underlying blockchain backbone protocol. Instead, [CYDC+20] proposes payment instructions to be posted to the ledger in *commitment form* accompanied with non-interactive zero-knowledge proofs attesting their well-formedness. Here, local netting solutions are computed *by each* *participating bank* and verified by a coordinating smart contract, which verifies correctness of all submitted, local netting solutions without revealing amounts and the identity of institutions. Since parties must compute partial netting solutions, the protocol of [CYDC+20] is not *robust* against cheating participants, who can stall or abort the netting process by posting invalid partial netting proposals. In contrast, [DCMA22] computes the netting solution inside an MPC instance, thereby achieving *fault tolerance* against dishonest participants. Despite initial implementation benchmarks provided by works above, it remains an open question in what configuration such systems can scale to real-world payment settlement volume and what netting frequency is required in practice.

6.4 Privacy-Preserving, Demand Response Markets

Privacy in markets is applicable to other domains, such as demand-response auctions; the growth of sustainable energy generation has introduced the necessity of increased coordination between the *production* and *consumption* of electricity. In contrast to traditional power generation sources, such as gas turbines, which can be throttled to match current power demand on the grid, sustainable power sources such as wind or solar cannot. Today, grid operators purchase future *flexible demand* from large, industrial scale consumers of energy or even power generators with flexibility to scale production in either direction to balance the grid.

There are efforts to aggregate retail consumers of electricity to form sellers of demand-response capacity; for example, home HVAC system of buildings can easily shift their power consumption forwards or backwards in time whilst maintaining set temperature. Residential electric vehicle charging stations can easily defer charging until opportune time periods to stabilize the grid, without compromising the convenience of the owner. However, serious privacy concerns arise when submitting device-level power consumption constraints to a demand-response auction run by the utility, as such information easily reveals the type of activity occurring in private homes.

Thus, in the work of Zobiri *et al.* [ZGND22], future demand capacity at the retail device-level can be sold to buyers; in this work, each auction round for a time frame within the *day-ahead* will accept submitted bids that include the maximum power consumption or power draw (KW), price limit ($/KWh) as well as characterization of each the demand-flexibility of each individually controllable device. For example, a washing machine can only provide demand flexibility for its starting time; once activated, a washing cycle cannot be interrupted. In contrast, an electric vehicle charger can charge intermittently at different power levels, but must completely charge the vehicle by a certain time. Thus, the auction does not simply involve buy and sell bids by ”volume” and ”price”; rather, it must take into consideration the forecast, future consumption-demand of the buyer, and device-level constraints submitted by sellers. When implemented in MPC, this arguably leads to a more complicated auction algorithm than in traditional markets; the work [ZGND22] permits private bids to be submitted to capture the consumption flexibility as *constraints* for each device (temporal constraints, power consumption cycle constraints), in addition to a electricity price *limit*; power consumption constraints are aggregated over private bids inside the MPC, such that device-level information is protected from the buyer of demand-response capacity. The auction clearing mechanism must then match device-level consumption constraints against predicted power generation schedule published by the buyer. Thus, demand-response markets imply direct device scheduling in residential homes by the buyer (or utility operator); privacy of end-users must therefore be protected by such solutions where any scheduling information is evaluated inside an MPC instance.

Da Gama *et al.* [GZN22] propose a peer-2-peer electricity market run a similar MPC setting, where local producers and consumers of electricity can trade energy *intra-day*; here, the auction design resembles that of the author's prior work in dark-pools [dGCP$^+$22], and exhibits sufficient transaction throughput for intra-day auction applications. [GZN22] builds on prior work [AACM16], but also considers geographical proximity of buyers and sellers, thereby stabilizing grid operations as power generation and consumption can be optimized to occur locally.

7 Future applications

We will now outline which other PET use cases could be of interest in the financial sector in the foreseeable future. While many of the use cases previously described in this work may also not yet be production-ready, we want to highlight areas in this section which we think deserve more attention by researchers and practitioners. This necessarily is of speculative nature, so the reader may see this as food for thought.

Voting. Voting is a standard mechanism in deciding on future policies. While in many cases it is sufficient to make the whole voting process public, this is not always possible. For example, a voter may fear repercussions or embarrassment if his or her vote becomes public. Hence, to ensure *honest digital voting*, cryptographic voting algorithms have to be used. These ensure that election outcomes can be computed while individual votes cannot be attributed to participants. While such cryptographic voting can be realized using MPC or FHE, a dedicated line of work started by Chaum [Cha81] presents highly efficient dedicated voting protocols. Cryptographic voting mechanisms find interesting applications in the DeFi space, e.g. for privacy-preserving Decentralized Anonymous Organizations (DAOs). In particular, a treasury system for DAOs based on electronic voting has been proposed in [ZOB19] and a board room voting scheme based on smart contracts (and this amenable to the DAO scenario) has been proposed in [MSH17]. We believe that these techniques may also be useful for coordination among classical banking institutions and other financial operations (*e.g.* shareholder meetings).

Fraud detection. Both insurance and gambling are known as industries where companies in the sector exchange information on their customers in order to detect fraud or exploitative customers[16]. This information sharing may be problematic for privacy reasons, and it also leaks information about suspected but ultimately honest customers if done in plain. PETs such as PSI might be an interesting tool to construct a trusted intermediary. This intermediary can obtain information from participating companies and alerts them if e.g. more than 3 of them share the same customer. Here, PSI can ensure that only those customers are revealed that appear often enough.

Better and fair recognition of patterns. In section 3 we have outlined how AML does benefit from recognition of suspicious patterns. Such patterns, if one wants to keep up in the digital age, must be learned from a large dataset and must be updated frequently. Moreover, many companies in an industry have an interest in pooling their data with other institutions for the purpose of learning these patterns. At the same time, they may not want to share raw customer or transaction data. Another, related area is assessing the credit risk of potential customers. Here, the risk becomes more accurate the more participants can contribute information or models. At the same time, input providers have an interest that their inputs to the scoring mechanism remain private (for data protection or to protect intellectual property).

Both applications fall into the area of privacy-preserving Machine Learning [LP00,DA$^+$01,MZ17] or confidential benchmarking [DDN$^+$16] which are subfields of MPC. While these areas have received much attention recently[17], optimized applications to finance seem to be lacking.

A further important aspect is that (automatically generated) models should not be biased against certain groups. While fair machine learning itself is a rapidly developing field, its application to finance [dCCP20] may deserve more attention.

Privacy preserving mitigation of systemic risk. Audits of financial institutions guarantee that their balances plus credit cover outstanding obligations. This reduces counter-party risk and means that the overall system can rely less on biasable methods such as ratings and reputation. At the same time, an audited company may not want to open its books fully to the public, or it might not be guaranteed that these books are correct. [MNN$^+$18] have shown how audits can be realized using ZK proofs, although limited to the futures market. We believe that this concept may be generalized to the wider financial system to permit privacy-preserving audits.

[16] For example the infamous "Griffin Book".

[17] Privacy-preserving Machine Learning opens up interesting use cases, but it does not come without its own problems. See [DC21] for a good overview.

References

AACM16. Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A Mustafa. An mpc-based privacy-preserving protocol for a local electricity trading market. In *Cryptology and Network Security: 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings 15*, pages 615–625. Springer, 2016.

ACC$^+$20. Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. Privacy-preserving auditable token payments in a permissioned blockchain system. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 255–267. ACM, 2020.

AMMM18. Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. PASTA: PASsword-based threshold authentication. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 2042–2059. ACM Press, October 2018.

AO12. Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, Heidelberg, December 2012.

AS22. Ghada Almashaqbeh and Ravital Solomon. Sok: Privacy-preserving computing in the blockchain era. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 124–139, 2022.

ATV20. Ignacio Alamillo, Cristina Timon, and Julian Valero. Oblivious identity management for private user-friendly services: D3.2 security and privacy-aware olympus framework impact assessment, 2020.

AZ19. Nasser Alsalami and Bingsheng Zhang. Sok: A systematic study of anonymity in cryptocurrencies. In *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–9, 2019.

BAZB20a. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 423–443. Springer, Heidelberg, February 2020.

BAZB20b. Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 2020.

BCD$^+$09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, February 2009.

BCDF22. Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient Privacy Preserving Smart Contracts. *Cryptology ePrint Archive (To appear in Financial Cryptography and Data Security 2023)*, 2022. https://eprint.iacr.org/2022/1435.

BCG$^+$20. Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020.

BCGW22. Dor Bitan, Ran Canetti, Shafi Goldwasser, and Rebecca Wexler. Using zero-knowledge to reconcile law enforcement secrecy and fair trial rights in criminal cases. In Daniel J. Weitzner, Joan Feigenbaum, and Christopher S. Yoo, editors, *Proceedings of the 2022 Symposium on Computer Science and Law, CSLAW 2022, Washington DC, USA, November 1-2, 2022*, pages 9–22. ACM, 2022.

BDF21. Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In *International Conference on Applied Cryptography and Network Security*, pages 163–194. Springer, 2021.

BDJ$^+$06. Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 142–147. Springer, Heidelberg, February / March 2006.

BDP20. Tucker Balch, Benjamin E Diamond, and Antigoni Polychroniadou. SecretMatch: inventory matching from fully homomorphic encryption. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–7, 2020.

BDTJ18. Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and {Exploit-Resistant} smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, 2018.

BFH$^+$20. Carsten Baum, Tore Kasper Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. PESTO: proactively secure distributed single sign-on, or how to trust a hacked server. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 587–606. IEEE, 2020.

BG20. Amira Barki and Aline Gouget. Achieving privacy and accountability in traceable digital currency. Cryptology ePrint Archive, Report 2020/1565, 2020. https://eprint.iacr.org/2020/1565.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

BHSR19. Samiran Bag, Feng Hao, Siamak F Shahandashti, and Indranil Ghosh Ray. SEAL: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 15:2042–2052, 2019.

BKB20. Joseph Burleson, Michele Korver, and Dan Boneh. Privacy-protecting regulatory solutions using zero-knowledge proofs: Full paper, 2020.

BST21. David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV standard: Break, fix, verify. In *2021 IEEE Symposium on Security and Privacy*, pages 1766–1781. IEEE Computer Society Press, May 2021.

ByCD$^+$21. Carsten Baum, James Hsin yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. Sok: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Paper 2021/1628, 2021. To appear on the Proceedings of the The 2nd Workshop on Decentralized Finance (DeFi) in Association with Financial Cryptography 2022.

Cac99. Christian Cachin. Efficient private bidding and auctions with an oblivious third party. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 120–127, 1999.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

CDD17. Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 683–699. ACM Press, October / November 2017.

CEOS22. Kota Chin, Keita Emura, Kazumasa Omote, and Shingo Sato. A Sealed-bid Auction with Fund Binding: Preventing Maximum Bidding Price Leakage. In *2022 IEEE International Conference on Blockchain (Blockchain)*, pages 398–405. IEEE, 2022.

CG08. Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 345–356. ACM Press, October 2008.

CGG$^+$20. Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1769–1787, 2020.

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th FOCS*, pages 383–395. IEEE Computer Society Press, October 1985.

Cha81. David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

CLLN14. Jan Camenisch, Anja Lehmann, Anna Lysyanskaya, and Gregory Neven. Memento: How to reconstruct your secrets from a single password in a hostile environment. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 256–275. Springer, Heidelberg, August 2014.

CLR17. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

Coi21. Coinbase. Coinbase to acquire leading cryptographic security company, Unbound Security, Nov 2021. `https://www.coinbase.com/blog/coinbase-to-acquire-leading-cryptographic-security-company-unbound-security`.

CP18. Aloni Cohen and Sunoo Park. Compelled decryption and the fifth amendment: Exploring the technical boundaries. *Harvard Journal of Law and Technology*, 32(1):169–233, 2018.

CSTA19. John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. MPC joins the dark side. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 148–159, 2019.

CSTA21. John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Multi-party computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross. *Intelligent Systems in Accounting, Finance and Management*, 28(4):239–267, 2021.

CV02. Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002.

CYDC$^+$20. Shengjiao Cao, Yuan Yuan, Angelo De Caro, Karthik Nandakumar, Kaoutar Elkhiyaoui, and Yanyan Hu. Decentralized privacy-preserving netting protocol on blockchain for payment systems. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 137–155, Cham, 2020. Springer International Publishing.

DA$^+$01. Wenliang Du, Mikhail J Atallah, et al. Privacy-preserving cooperative scientific computations. In *csfw*, volume 1, page 273, 2001.

DC21. Emiliano De Cristofaro. A critical overview of privacy in machine learning. *IEEE Security & Privacy*, 19(4):19–27, 2021.

dCCP20. Leo de Castro, Jiahao Chen, and Antigoni Polychroniadou. Cryptocredit: securely training fair models. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

DCMA22. Angelo De Caro, Andrew Miller, and Amit Agarwal. Privacy-Preserving Decentralized Multi-Party Netting, September 29 2022. US Patent App. 17/216,644, https://patents.google.com/patent/US20220309492A1/en.

DDN$^+$16. Ivan Damgård, Kasper Damgård, Kurt Nielsen, Peter Sebastian Nordholt, and Tomas Toft. Confidential benchmarking based on multiparty computation. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 169–187. Springer, Heidelberg, February 2016.

dGCP$^+$22. Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P Smart, and Younes Talibi Alaoui. Kicking-the-bucket: Fast privacy-preserving trading using buckets. In *International Conference on Financial Cryptography and Data Security*, pages 20–37. Springer, 2022.

dGCSA22. Mariana Botelho da Gama, John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/923.

DGK$^+$21. Ivan Damgård, Chaya Ganesh, Hamidreza Khoshakhlagh, Claudio Orlandi, and Luisa Siniscalchi. Balancing privacy and accountability in blockchain identity management. In Kenneth G. Paterson, editor, *CT-RSA 2021*, volume 12704 of *LNCS*, pages 552–576. Springer, Heidelberg, May 2021.

DGP22. Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. FAST: fair auctions via secret transactions. In *International Conference on Applied Cryptography and Network Security*, pages 727–747. Springer, 2022.

DJN$^+$22. Ivan Damgård, Thomas P Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ecdsa with honest majority. *Journal of Computer Security*, 30(1):167–196, 2022.

DL05. Whitfield Diffie and Susan Landau. The export of cryptography in the 20th century and the 21st, 2005.

DMNS06. Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal

Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006.

Eur16. European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), 2016.

Fir21. Fireblocks. Fireblocks. Institutional Digital Asset Custody, Settlement & Issuance, Nov 2021. https://www.fireblocks.com/.

FNP04. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

FR96. Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1996.

Fre21. Tore Kasper Frederiksen. A holistic approach to enhanced security and privacy in digital health passports. In Delphine Reinhardt and Tilo Müller, editors, *ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021*, pages 133:1–133:10. ACM, 2021.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

GGM14. Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *NDSS 2014*. The Internet Society, February 2014.

GHM$^+$17. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

GKL15. Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

GKS22. Chaya Ganesh, Bhavana Kanukurthi, and Girisha Shankar. Secure Auctions in the Presence of Rational Adversaries. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1173–1186, 2022.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

GMW86. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

GP18. Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist? Cryptology ePrint Archive, Report 2018/664, 2018. https://eprint.iacr.org/2018/664.

Gro16. Lev Grossman. Inside apple CEO tim cook's fight with the FBI. *Time*, 2016. Accessed on 31/01/2022.

GVJR22. Kavya Govindarajan, Dhinakaran Vinayagamurthy, Praveen Jayachandran, and Chester Rebeiro. Privacy-preserving decentralized exchange marketplaces. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2022.

GZN22. Mariana Gama, Fairouz Zobiri, and Svetla Nikova. Multi-party computation auction mechanisms for a p2p electricity market with geographical prioritization, 2022. https://www.esat.kuleuven.be/cosic/publications/article-3526.pdf.

HR22. Lukas Helminger and Christian Rechberger. Multi-party computation in the GDPR. *IACR Cryptol. ePrint Arch.*, page 491, 2022.

HTK98. Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *USENIX Workshop on Electronic Commerce*, 1998.

HZ09. Feng Hao and Piotr Zieliński. A 2-round anonymous veto protocol. In *International Workshop on Security Protocols*, pages 202–211. Springer, 2009.

JKKX17. Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. TOPPSS: Cost-minimal password-protected secret sharing based on threshold OPRF. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 39–58. Springer, Heidelberg, July 2017.

KLN22. Markulf Kohlweiss, Anna Lysyanskaya, and An Nguyen. Privacy-preserving blueprints. *IACR Cryptol. ePrint Arch.*, page 1536, 2022.

KRDO17. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

LN18. Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854, 2018.

LP00. Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 36–54. Springer, Heidelberg, August 2000.

Max21. Nick Maxwell. Case studies of the use of privacy preserving analysis to tackle financial crime, Jan 2021. https://www.future-fis.com/uploads/3/7/9/4/3794525/ffis_innovation_and_discussion_paper_-_case_studies_of_the_use_of_privacy_preserving_analysis_-_v.1.3.pdf.

MMZ$^+$21. Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy*, pages 1348–1366. IEEE Computer Society Press, May 2021.

MNN$^+$18. Fabio Massacci, Chan Nam Ngo, Jing Nie, Daniele Venturi, and Julian Williams. FuturesMEX: secure, distributed futures market exchange. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 335–353. IEEE, 2018.

MoLA00. Justice Ministry of Law and Company Affairs. The information technology act, 2000, 2000. https://bit.ly/3JuiUwy.

MSH17. Patrick McCorry, Siamak F. Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 357–375. Springer, Heidelberg, April 2017.

MZ17. Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society Press, May 2017.

NMKW21. Chan Nam Ngo, Fabio Massacci, Florian Kerschbaum, and Julian Williams. Practical witness-key-agreement for blockchain-based dark pools financial trading. In *International Conference on Financial Cryptography and Data Security*, pages 579–598. Springer, 2021.

NPS99. Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

NVV18. Neha Narula, Willy Vasquez, and Madars Virza. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In Sujata Banerjee and Srinivasan Seshan, editors, *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, pages 65–80. USENIX Association, 2018.

NYS$^+$18. Ken Naganuma, Masayuki Yoshino, Hisayoshi Sato, Nishio Yamada, Takayuki Suzuki, and Noboru Kunihiro. Decentralized netting protocol over consortium blockchain. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 174–177. IEEE, 2018.

Paq13. Christian Paquin. U-Prove Technology Overview V1.1. Tech report, Microsoft Corporation, April 2013.

PAR18. ARTICLE 29 DATA PROTECTION WORKING PARTY. Wp250: Guidelines on personal data breach notification under regulation 2016/679, 2018.

PNP20. Juha Partala, Tri Hong Nguyen, and Susanna Pirttikangas. Non-interactive zero-knowledge for blockchain: A survey. *IEEE Access*, 8:227945–227961, 2020.

PRST08. David C Parkes, Michael O Rabin, Stuart M Shieber, and Christopher Thorpe. Practical secrecy-preserving, verifiably correct and trustworthy auctions. *Electronic Commerce Research and Applications*, 7(3):294–312, 2008.

PSS19. Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado Cash Privacy Solution, version 1.4, Dec. 2019. https://web.archive.org/web/20211026053443/https://tornado.cash/audits/TornadoCash_whitepaper_v1.4.pdf.

PU15. THE EUROPEAN PARLIAMENT and THE COUNCIL OF THE EUROPEAN UNION. Directive (EU) 2015/2366 of the european parliament and of the council, Nov 2015. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L2366&from=EN.

RAD$^+$78. Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

RB89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

RT04. Peter Reuter and Edwin M. Truman. *Chasing Dirty Money: The Fight Against Money Laundering*. Peterson Institute for International Economics, 2004.

San20. Olivier Sanders. Efficient redactable signature and application to anonymous credentials. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 628–656. Springer, Heidelberg, May 2020.

SBBV22. Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. ZeeStar: Private Smart Contracts by Homomorphic Encryption and Zero-knowledge Proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1543–1543. IEEE Computer Society, 2022.

SBG$^+$19. Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1759–1776, 2019.

SCG$^+$14. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014.

Sha79. Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

SHD17. Gerald Spindler, Anna Zsoofia Horvaath, and Lukas Dalby. Scalable oblivious data analytics: D.3.1 general legal aspects, 2017.

SLC22. Manu Sporny, Dave Longley, and David Chadwick. Verifiable credentials data mode, 2022.

SLS$^+$22. Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, and Christopher Allen. Decentralized identifiers (DIDs), 2022.

SSLM21. Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1955–1972. USENIX Association, August 2021.

ST99. Tomas Sander and Amnon Ta-Shma. Flow control: A new approach for anonymity control in electronic cash systems. In Matthew Franklin, editor, *FC'99*, volume 1648 of *LNCS*, pages 46–61. Springer, Heidelberg, February 1999.

TBA$^+$22. Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. UTT: Decentralized ecash with accountable privacy. Cryptology ePrint Archive, Report 2022/452, 2022. https://eprint.iacr.org/2022/452.

TCS21. Christof Ferreira Torres, Ramiro Camino, and Radu State. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1343–1359. USENIX Association, 2021.

TJW22. Shahroz Tariq, Sowon Jeon, and Simon S. Woo. Am I a real or fake celebrity? evaluating face recognition and verification apis under deepfake impersonation attack. In Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini, editors, *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, pages 512–523. ACM, 2022.

TMSgD22. Amos Treiber, Dirk Müllmann, Thomas Schneider, and Indra Spiecker genannt Döhmann. Data protection law and multi-party computation:

Applications to information exchange between law enforcement agencies. Cryptology ePrint Archive, Report 2022/1242, 2022. https://eprint.iacr.org/2022/1242.

TP07. Christopher Thorpe and David C Parkes. Cryptographic securities exchanges. In *International Conference on Financial Cryptography and Data Security*, pages 163–178. Springer, 2007.

vERS21. Marie Beth van Egmond, Thomas Rooijakkers, and Alex Sangers. Privacy-Preserving Collaborative Money Laundering Detection. *ERCIM News*, 2021(126), 2021.

WKCC19. Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. PRCash: Fast, private and regulated transactions for digital currencies. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 158–178. Springer, Heidelberg, February 2019.

WPG$^+$21. Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. Sok: Decentralized finance (defi). *arXiv preprint arXiv:2101.08778*, 2021. https://arxiv.org/abs/2101.08778.

WXF$^+$18. Xin Wang, Xiaomin Xu, Lance Feagan, Sheng Huang, Limei Jiao, and Wei Zhao. Inter-bank payment system on enterprise blockchain platform. In *2018 IEEE 11th international conference on cloud computing (CLOUD)*, pages 614–621. IEEE, 2018.

XCZ$^+$22. Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. Veri-zexe: Decentralized private computation with universal setup. *Cryptology ePrint Archive*, 2022.

ZCC$^+$16. Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 270–282. ACM Press, October 2016.

ZGND22. Fairouz Zobiri, Mariana Gama, Svetla Nikova, and Geert Deconinck. A Privacy-Preserving Three-Step Demand Response Market Using Multi-Party Computation. In *13th Int. Conf. Innov. Smart Grid Technol.(ISGT North Am. 2022), Washingt. DC (to Appear)*, 2022. https://www.esat.kuleuven.be/cosic/publications/article-3451.pdf.

ZMM$^+$20. Fan Zhang, Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. DECO: Liberating web data using decentralized oracles for TLS. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1919–1938. ACM Press, November 2020.

ZOB19. Bingsheng Zhang, Roman Oliynykov, and Hamed Balogun. A treasury system for cryptocurrencies: Enabling better collaborative intelligence. In *NDSS 2019*. The Internet Society, February 2019.

ZOP20. Arman Zand, James Orwell, and Eckhard Pfluegel. A Secure Framework for Anti-Money-Laundering using Machine Learning and Secret Sharing. In *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–7, 2020.

Part IV

Conclusion

6 Future work

In this thesis, we introduce new approaches and techniques towards advancing the state-of-the-art in security and privacy of cryptoeconomic systems (§2). We highlight three directions of future work which address pressing, open problems in the field of cryptoeconomics; future progress in these research directions would extend the results contributed by this thesis.

6.1 Incentive composition in cryptoeconomics

As we highlight in §4.2.3, the incentive in automatic market makers (§4.2.2) to align the marginal price with the external, fair market price does not securely compose with the incentive to liquidate undercollateralized loans in lending protocols (§4.2.3); when considered in isolation by rational parties, the automatic market maker will be aligned with the fair market price, and lending protocol liquidations only occur when loans become undercollateralized. In composition, where the lending protocol obtains pricing data from automatic market makers, the rational actor may be incentivized to manipulate prices on automatic market makers in order to trigger liquidations pre-maturely.

Another example is front-running by the block leader (§4.3.1), where a party ordering inputs at cryptoeconomic layer 1 is incentivized to do so in an unfair manner by a cryptoeconomic layer 2 application (P4). Another cross-layer example of unintended incentive interference is the competition for staked tokens between proof-of-stake and lending protocols [Chi21]. Although not addressed in this thesis, we note a rapidly advancing line of research on cross-chain bridges, which permit *synchronization* of state between different instances of permissionless consensus [MBYS21], further complicating the incentive analysis of composed systems.

Traditional protocol design and analysis assumes *honest* behaviour as specified by the protocol designer. In the permissionless setting (§3.3), this is perhaps too strong of an assumption to place on anonymous parties on the public internet. Instead, honest behaviour should ideally also represent the winning strategy of a rational party, also referred to as incentive-compatibility. Progress towards a theory on incentive-compatibility under composition across crytoeconomic layers 1 and 2 (Def. 1.1) would undoubtably represent an important contribution to cryptoeconomics and computer science. We note the long-running line of work of *rational cryptography* [HT04, KN08, FKN10, AAH11, ACH11, GK12, HP15] which may provide inspiration towards this goal.

6.2 Permissionless, privacy-preserving smart contracts

Permissionless consensus realizes a notion of serverless computing, where participating users remain anonymous and can come and go; the ephemeral nature of protocol roles represents a challenge for achieving privacy in smart contracts.

Although we have we have shown an expressive, privacy-preserving contract manager model (§5.2) and contributed the first practical realization thereof in P7, this comes at the cost of introducing a secure multi-party server committee that assumes authenticated, private communication links, and is therefore no longer permissionless (§3.3.1). More restricted notions of privacy-preserving smart contracts have been proposed (§5.1) which adhere to the permissionless setting, but are not sufficiently expressive to realize most applications in decentralized finance, the most widely used cryptoeconomic application domain today.

We highlight a promising line of MPC research [GHK+21, BGG+20, CDGK23] inspired by anonymous role election mechanisms in proof-of-stake (§3.3.5), which promises secure multi-party computation in the permissionless setting; this setting has been formalized as You-Only-Speak-Once or YOSO MPC in [GHK+21]. Its practical realization would imply universally expressive privacy-preserving smart contracts in the contract manager model (§5.2), where the contract manager is distributed by a permissionless YOSO-MPC protocol instance.

A key challenge of YOSO-MPC is the forwarding of secret-shared material distributed across MPC servers; in the permissionless setting, each player must be replaceable anytime. This is achieved in permissionless consensus (§3.3.5) by electing a fresh party for the computation and broadcast of each protocol message; since secret-shared state must be carried forward to committee members *elected in the future*, this requires a notion of *encrypting to a future committee*, currently not trivially achievable in a practical manner.

6.3 Universal, differentially private mechanisms for MPC

In our investigation to achieve pre- and post-trade privacy (§4.3.2) in privacy-preserving markets in decentralized finance, we introduce an extended notion of differential privacy applicable to the trusted curator setting (§5.3.1), where users not only submit **private inputs** but also receive individual (potentially correlated) **private outputs** in each evaluation round; in contrast to standard differential privacy over a private database queried by an analyst, the entire (input & output) transcript of the honest client must be protected from the adversary corrupting clients in this setting. We achieve this with an extended definitional framework named round-differential privacy (Def. 5.3 and P8), which also applies when the role of the trusted curator is distributed by a MPC committee. We then propose round-differentially private market mechanisms for MPC and Eagle (P7). In the process, we observe that there exist many applications in this MPC setting which require round-differential privacy to prevent function leakage from the full private (input & output) client transcript. Whilst standard, differentially private mechanisms have been implemented in MPC, these works do not consider privacy over the full, individual transcript in the trusted curator model (§5.3.1); here, the output is generally a query result returned to the adversary, as in the case of privacy-preserving machine learning with MPC and differential privacy [PRR10, ACA+17, WHMM21, PRM+22].

Thus, we highlight the need for a more general, differential privacy framework for the MPC setting; whilst we have devised application-specific mechanisms which achieve round-differential privacy in P8 , we believe general noise mechanisms may exist, which would greatly facilitate the design of round-differentially private applications in MPC and the trusted curator setting. In the classic setting of differential privacy, the Laplace mechanism [DR+14], provides a straightforward and effective tool to transform many statistical database queries into differentially private ones. We think the investigation of **universal noise mechanisms** for the MPC setting with private inputs and private outputs to individual clients would provide a powerful, yet understudied privacy framework.

Bibliography

[AAE+22] Guillermo Angeris, Akshay Agrawal, Alex Evans, Tarun Chitra, and Stephen Boyd. Constant function market makers: Multi-asset trades via convex optimization. In *Handbook on Blockchain*, pages 415–444. Springer, 2022. https://doi.org/10.1007/978-3-031-07535-3_13.

[AAH11] Ittai Abraham, Lorenzo Alvisi, and Joseph Y Halpern. Distributed computing meets game theory: combining insights from two fields. *Acm Sigact News*, 42(2):69–76, 2011. https://doi.org/10.1145/1998037.1998055.

[ACA+17] Abbas Acar, Z Berkay Celik, Hidayet Aksu, A Selcuk Uluagac, and Patrick McDaniel. Achieving secure and differentially private computations in multiparty settings. In *2017 IEEE Symposium on Privacy-Aware Computing (PAC)*, pages 49–59. IEEE, 2017. https://doi.org/10.1109/PAC.2017.12.

[ACG+18] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. A fair consensus protocol for transaction ordering. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 55–65. IEEE, 2018. https://doi.org/10.1109/ICNP.2018.00016.

[ACH11] Gilad Asharov, Ran Canetti, and Carmit Hazay. Towards a game theoretic view of secure computation. In *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*, pages 426–445. Springer, 2011. https://doi.org/10.1007/978-3-642-20465-4_24.

[AECB22] Guillermo Angeris, Alex Evans, Tarun Chitra, and Stephen Boyd. Optimal routing for constant function market makers. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 115–128, 2022. https://doi.org/10.1145/3490486.3538336.

[AZS+21] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. *Tech. rep., Uniswap, Tech. Rep.*, 2021. https://berkeley-defi.github.io/assets/material/Uniswap%20v3%20Core.pdf.

[BA21] Jaya Klara Brekke and Wassim Zuhair Alsindi. Cryptoeconomics. *Internet Policy Review*, 10(2), 2021. https://doi.org/10.14763/2021.2.1553.

[BCDF22] Carsten Baum, James Hsin-yu Chiang, Bernardo David, and Tore Kasper Frederiksen. Eagle: Efficient Privacy Preserving Smart Contracts. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/1435.

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964. IEEE, 2020. https://doi.org/10.1109/SP40000.2020.00050.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *Advances in Cryptology–CRYPTO 2005: 25th*

Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25, pages 361–377. Springer, 2005. https://doi.org/10.1007/11535218_22.

[BCT21] Aritra Banerjee, Michael Clear, and Hitesh Tewari. zkhawk: Practical private smart contracts from mpc-based hawk. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 245–248. IEEE, 2021. https://doi.org/10.1109/BRAINS52497.2021.9569822.

[BDD$^+$21] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: a foundation of time-lock puzzles in uc. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III*, pages 429–459. Springer, 2021. https://doi.org/10.1007/978-3-030-77883-5_15.

[BDKJ21] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. *arXiv preprint arXiv:2109.04347*, 2021. https://arxiv.org/abs/2109.04347.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, pages 213–229. Springer, 2001. https://doi.org/10.1007/3-540-44647-8_13.

[BF21] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–326. Springer, 2021. https://doi.org/10.1007/978-3-030-77870-5_11.

[BGG$^+$20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 260–290. Springer, 2020. https://doi.org/10.1007/978-3-030-64375-1_10.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018. https://arxiv.org/abs/1807.04938.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings*, pages 236–254. Springer, 2000. https://doi.org/10.1007/3-540-44598-6_15.

[BO83] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983. https://doi.org/10.1145/800221.806707.

[Bra84] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162, 1984. https://doi.org/10.1145/800222.806743.

[But13] Vitalik Buterin. Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2013.

[BZ23] Massimo Bartoletti and Roberto Zunino. A theoretical basis for blockchain extractable value. *arXiv preprint arXiv:2302.02154*, 2023. https://arxiv.org/abs/2302.02154.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001. https://doi.org/10.1109/SFCS.2001.959888.

[CDGK23] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. Yolo yoso: Fast and simple encryption and secret sharing in the yoso model. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part I*, pages 651–680. Springer, 2023. https://doi.org/10.1007/978-3-030-84245-1_3.

[Chi21] Tarun Chitra. Competitive Equilibria Between Staking and On-chain Lending. *Cryptoeconomic Systems*, 0(1), apr 5 2021. https://cryptoeconomicsystems.pubpub.org/pub/chitra-staking-lending-equilibria.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999. https://www.usenix.org/legacy/publications/library/proceedings/osdi99/full_papers/castro/castro.ps.

[CMSZ22] Christian Cachin, Jovana Mićić, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers*, pages 316–333. Springer, 2022. https://doi.org/10.1007/978-3-031-18283-9_15.

[CSTA19] John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. Mpc joins the dark side. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 148–159, 2019. https://doi.org/10.1145/3321705.3329809.

[DFMPS19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25*, pages 248–277. Springer, 2019. https://doi.org/10.1007/978-3-030-34578-5_10.

[dGCP+22] Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P Smart, and Younes Talibi Alaoui. Kicking-the-bucket: Fast privacy-preserving trading using buckets. In *International Conference on Financial Cryptography and Data Security*, pages 20–37. Springer, 2022. https://doi.org/10.1007/978-3-031-18283-9_2.

[dGCSA22] Mariana Botelho da Gama, John Cartlidge, Nigel P Smart, and Younes Talibi Alaoui. All for one and one for all: Fully decentralised privacy-preserving dark pool trading using multi-party computation. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/923.

[DHMW22] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/433.

[DKT+20] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020. https://doi.org/10.1145/3372297.3417290.

[DKT21] Soubhik Deb, Sreeram Kannan, and David Tse. Posat: proof-of-work availability and unpredictability, without the work. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised*

Selected Papers, Part II 25, pages 104–128. Springer, 2021. https://doi.org/10.1007/978-3-662-64331-0_6.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988. https://doi.org/10.1145/42282.42283.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014. http://dx.doi.org/10.1561/0400000042.

[DRCA23] Theo Diamandis, Max Resnick, Tarun Chitra, and Guillermo Angeris. An efficient algorithm for optimal routing through constant function market makers. *arXiv preprint arXiv:2302.04938*, 2023. https://arxiv.org/abs/2302.04938.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983. https://doi.org/10.1137/0212045.

[ES18] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018. https://doi.org/10.1145/3212998.

[FKN10] Georg Fuchsbauer, Jonathan Katz, and David Naccache. Efficient rational secret sharing in standard communication networks. In *Theory of Cryptography: 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings 7*, pages 419–436. Springer, 2010. https://doi.org/10.1007/978-3-642-11799-2_25.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985. https://doi.org/10.1145/3149.214121.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: You only speak once: Secure mpc with stateless ephemeral roles. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*, pages 64–93. Springer, 2021. https://doi.org/10.1007/978-3-030-84245-1_3.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017. https://doi.org/10.1145/3132747.3132757.

[GK12] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 81–98. Springer, 2012. https://doi.org/10.1007/978-3-642-29011-4_7.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 281–310. Springer, 2015. https://doi.org/10.1007/978-3-662-46803-6_10.

[GWPK20a] Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In *Proceedings of* *the 2nd ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020. https://doi.org/10.1145/3419614.3423254.

[GWPK20b] Lewis Gudgeon, Sam Werner, Daniel Perez, and William J Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency. In *ACM Conference on Advances in Financial Technologies*, pages 92–112, 2020.

[HP15] Joseph Y Halpern and Rafael Pass. Algorithmic rationality: Game theory with costly computation. *Journal of Economic Theory*, 156:246–268, 2015. https://doi.org/10.1016/j.jet.2014.04.007.

[HSW22] Lioba Heimbach, Eric Schertenleib, and Roger Wattenhofer. Risks and returns of uniswap v3 liquidity providers. *arXiv preprint arXiv:2205.08904*, 2022. https://arxiv.org/abs/2205.08904.

[HT04] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 623–632, 2004. https://doi.org/10.1145/1007352.1007447.

[KDK22] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, pages 3–14, 2022. https://doi.org/10.1145/3494105.3526239.

[KKK21] Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Kachina–foundations of private smart contracts. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, pages 1–16. IEEE, 2021. https://doi.org/10.1109/CSF51468.2021.00002.

[KLX20] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part III 18*, pages 390–413. Springer, 2020. https://doi.org/10.1007/978-3-030-64381-2_14.

[KMS$^+$16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016. https://doi.org/10.1109/SP.2016.55.

[KN08] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 320–339. Springer, 2008. https://doi.org/10.1007/978-3-540-78524-8_18.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017. https://doi.org/10.1007/978-3-319-63688-7_12.

[KRS18] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 acm sigsac conference on computer and communications security*, pages 729–744, 2018. https://doi.org/10.1145/3243734.3243814.

[Kur20] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 25–36, 2020. https://doi.org/10.1145/3419614.3423263.

[KZGJ20] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual In-*

ternational Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40, pages 451–480. Springer, 2020. https://doi.org/10.1007/978-3-030-56877-1_16.

[MBYS21] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. Sok: Validating bridges as a scaling solution for blockchains. *Cryptology ePrint Archive*, 2021. https://eprint.iacr.org/2021/1589.pdf.

[MGZ23] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *Security and Privacy in Communication Networks: 18th EAI International Conference, SecureComm 2022, Virtual Event, October 2022, Proceedings*, pages 250–271. Springer, 2023.

[MMR23] Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. Complexity-approximation trade-offs in exchange mechanisms: Amms vs. lobs. *arXiv preprint arXiv:2302.11652*, 2023. https://fc23.ifca.ai/preproceedings/65.pdf.

[MXC$^+$16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016. https://doi.org/10.1145/2976749.2978399.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[Pen23] Penumbra. ZSwap documentation. https://protocol.penumbra.zone/main/zswap.html, 2023.

[PRM$^+$22] Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson Nascimento, and Martine De Cock. Training differentially private models with secure multiparty computation. *arXiv preprint arXiv:2202.02625*, 2022. https://arxiv.org/abs/2202.02625.

[PRR10] Manas Pathak, Shantanu Rane, and Bhiksha Raj. Multiparty differential privacy via aggregation of locally trained classifiers. *Advances in neural information processing systems*, 23, 2010. https://proceedings.neurips.cc/paper_files/paper/2010/file/0d0fd7c6e093f7b804fa0150b875b868-Paper.pdf.

[PS17a] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM symposium on principles of distributed computing*, pages 315–324, 2017. https://doi.org/10.1145/3087801.3087809.

[PS17b] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II 23*, pages 380–409. Springer, 2017. https://doi.org/10.1007/978-3-319-70697-9_14.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part II*, pages 643–673. Springer, 2017. https://doi.org/10.1007/978-3-319-56614-6_22.

[PWXL21] Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: Defi on a knife-edge. In *Financial Cryptography*, 2021. (to appear) https://arxiv.org/abs/2009.13235.

[Ren19] Ling Ren. Analysis of nakamoto consensus. *Cryptology ePrint Archive*, 2019. https://eprint.iacr.org/2019/943.pdf.

[RSW96] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-locked Puzzles and Time-release Crypto. https://people.csail.mit.edu/rivest/pubs/RSW96.pdf, 1996.

[SA21] Ravital Solomon and Ghada Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. *Cryptology ePrint Archive*, 2021. https://eprint.iacr.org/2021/133.

[SBBV22] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 179–197. IEEE, 2022. https://doi.org/10.1109/SP46214.2022.9833732.

[SBG$^+$19] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and Martin Vechev. zkay: Specifying and enforcing data privacy in smart contracts. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 1759–1776, 2019. https://doi.org/10.1145/3319535.3363222.

[SCG$^+$14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*, pages 459–474. IEEE, 2014. https://doi.org/10.1109/SP.2014.36.

[Shi20] Elaine Shi. Foundations of distributed consensus and blockchains. *Book manuscript*, 2020. http://elaineshi.com/docs/blockchain-book.pdf.

[Vos22] Eric Voskuil. Cryptoeconomics, 2022. https://voskuil.org/cryptoeconomics/.

[Wat17] Roger Wattenhofer. *Distributed Ledger Technology: The Science of the Blockchain*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2nd edition, 2017.

[WHMM21] Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021. https://doi.org/10.1145/3418290.

[WRA21] Dave White, Dan Robinson, and Hayden Adams. Time-weighted Average Market Maker (TWAMM). 2021. https://www.paradigm.xyz/2021/07/twamm/.

[XCZ$^+$22] Alex Luoyuan Xiong, Binyi Chen, Zhenfei Zhang, Benedikt Bünz, Ben Fisch, Fernando Krell, and Philippe Camacho. Veri-zexe: Decentralized private computation with universal setup. *Cryptology ePrint Archive*, 2022. https://eprint.iacr.org/2022/802.

[YMR$^+$19] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019. https://doi.org/10.1145/3293611.3331591.