LightPay: A Lightweight and Secure Off-Chain Multi-Path Payment Scheme Based on Adapter Signatures

Liu, Yaqin; Liang, Wei; Xie, Kun; Xie, Songyou; Li, Kuanching; Meng, Weizhi

Published in:
IEEE Transactions on Services Computing

Link to article, DOI:
10.1109/TSC.2023.3333806

Publication date:
2024

Document Version
Peer reviewed version

LightPay: A Lightweight and Secure Off-chain Multi-path Payment Scheme based on Adapter Signatures

Yaqin Liu, Wei Liang, Kun Xie, Songyou Xie, Kuanching Li, and Weizhi Meng

***Abstract*—The payment channel network aims to solve the problems of long payment confirmation time and limited throughput in cryptocurrencies through off-chain payments. Hash Time-Lock Contract (HTLC) is an off-chain payment protocol that Lightning Network (LN) adopted. Unfortunately, when performing high-valued payments off-chain, due to the impact of payment channel capacity, it is often necessary to split a single payment, which increases the transaction fees and time. Therefore, we propose LightPay, an atomic off-chain multi-path payment protocol based on adapter signature and discrete logarithm problem. Among different conditions encoded in the multi-path contract, the multi-path transmission of a single high-valued payment can be realized under the premise of the unlinkability of partial payments. We construct an ideal functionality in the Universal Composability framework and demonstrate that LightPay UC-realizes it, thereby providing proof of its security and privacy. Experimental results indicate that the payment success rate of LightPay can be increased by 11.08% in 0.0025 BTC payments compared with the single-path payment protocol Multihop HTLC in LN. Additionally, compared with the multi-path payment protocol CryptoMaze, the communication overhead required by LightPay is reduced to about 55.6% on average in the simulated network. Overall, LightPay has advantages regarding payment success rate and overhead.**

***Index Terms*—Blockchain, Off-chain Payment, Layer-2 Scaling, Payment Channel Network, Privacy Protection.**

I. Introduction

BLOCKCHAIN is a decentralized distributed ledger technology, and its core ideas originated from the Bitcoin white paper published in 2008 by Satoshi Nakamoto [1]. Blockchain has the characteristics of anonymity, tamper proof, decentralization, and traceability, among others. These features have swiftly captured the attention of academia and industry, leading to the widespread utilization of blockchain in various research fields, including data privacy protection and identity verification. Compared with centralized applications, blockchain nodes can establish a trusted channel without the help of a trusted third party and can be widely used in many decentralized scenarios, such as insurance claims, supply chain traceability [2], mobile crowdsourcing, and Internet of Things.

Cryptocurrency is an important application of blockchain, and there is much related research [3]. The anonymity, traceability, immutability, and decentralization of the blockchain guarantee the security and privacy of cryptocurrency transactions. However, due to the publicly available transaction records on the blockchain, where the transaction addresses and amounts are stored on-chain, which can easily lead to the leakage of user privacy [4]. For instance, Ron *et al.* [5] downloaded the complete historical records of Bitcoin and generated transaction correlation graphs by studying user behavior, hence breaking anonymity. In Ethereum, ledger information can be analyzed through address clustering technology by obtaining the identity relationship between accounts and real users [6]. The remote side-channel attack [7] can detect private information such as transaction amount and user IP address of encrypted currency systems in Zcash. Consequently, it is crucial to adopt appropriate methods to protect blockchain privacy when making payments [8], [9].

A Payment Channel (PC) is a temporary channel created on-chain that allows off-chain payments between two users without recording every transaction on the blockchain [10]. The PC can realize fast, low-cost payments, an effective method to protect the privacy of transactions. The PC realizes multiple off-chain payments by only requiring on-chain transactions during the opening or closing of the channel. Most of the time, the transaction records in the PC are only stored locally and are not uploaded on-chain, so their efficiency and privacy will be guaranteed. As Figure 1 shows, Bitcoin opens a payment channel by transferring coins from both nodes to a multi-signature address. After establishing the payment channel, unlimited-time payments can be made off-chain without interacting with the blockchain. When the channel needs to be closed, the coins of multi-signature address are allocated to each node on-chain according to the latest balance state of the corresponding PC. To distinguish between on-chain and off-chain transactions, we denote off-chain transactions as off-chain payments.

Yaqin Liu, Kun Xie, and Songyou Xie are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410082, China. Email: {liuyaqin963, cskxie, syxie}@hnu.edu.cn

Wei Liang is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan 411201, China, the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410082, China, and also with the Hunan Key Laboratory for Service Computing and Novel Software Technology, Xiangtan, Hunan 411201, China. Email: wliang@hnust.edu.cn

Kuanching Li is with the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, Hunan 411201, China, and also with the Hunan Key Laboratory for Service Computing and Novel Software Technology, Xiangtan, Hunan 411201, China. Email: aliric@hnust.edu.cn

Weizhi Meng is with the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark. Email: weme@dtu.dk

(Corresponding author: Wei Liang.)

Manuscript received April 19, 2021; revised August 16, 2021.

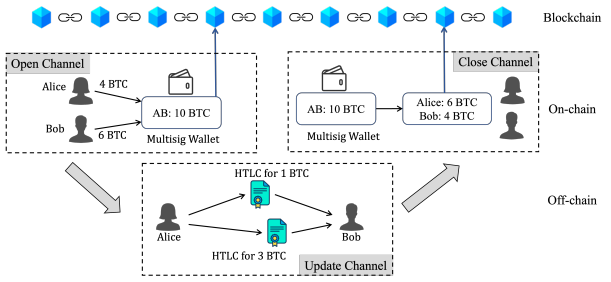


Fig. 1. Bitcoin payment channel life cycle.

The Payment Channel Network (PCN) is composed of multiple payment channels. Two users who have not established a payment channel can form a payment path through multiple intermediaries in PCN to complete the payment [11]. The sender of the payment can select the appropriate payment path according to PCN routing protocols [12]–[14]. The Hash Time-Lock Contract (HTLC) protocol implements atomic payment in the Lightning Network (LN). In particular, HTLC's nodes need to agree on a hash value $R = Hash(r)$ and then require each node to submit the same hash pre-image r within a certain period of time to complete the payment. If it expires, all payments promised by the nodes are invalidated. Nevertheless, HTLC is vulnerable to wormhole attacks, where malicious users can skip honest intermediaries to complete the payment, thereby stealing fees. In addition, using the same hash value in the construction of the off-chain payments may reveal the privacy of the payment [15].

When making a high-valued payment within a PCN, it is common to split the payment into multiple smaller single-path payments due to the limited channel capacity. With the transaction fees $0.1c$ and processing time t for each payment channel, let's consider a scenario where S intends to pay $6c$ to R. We also assume that the high-valued payment can be divided into three smaller single-path payments: $path_1 = S \rightarrow A \rightarrow B \rightarrow E \rightarrow R$ for $2c$, $path_2 = S \rightarrow A \rightarrow C \rightarrow E \rightarrow R$ for $1c$ and $path_3 = S \rightarrow A \rightarrow D \rightarrow R$ for $3c$. The total fees and processing time are $0.8c$ and $11t$, respectively, where S pays A three times, and node E pays R twice. However, in the proposed multi-path payment scheme, the public part of the three paths ($S \rightarrow A$, $E \rightarrow R$) does not need to be paid repeatedly, and the other parts can be executed concurrently. The total fees and processing time required by the multi-path payment are $0.6c$ and $4t$. Therefore, multi-path payment can reduce transaction fees and processing time. In addition, in a real payment environment, since the three single-path payments are not executed synchronously, other payments will likely be inserted in these single-path payments, so these payments cannot be completed atomically.

A. Related Work

1) Single-path payment protocol: Off-chain payments can address limitations such as blockchain scalability and transaction speed. Green *et al.* [16] proposed an anonymous payment channel Bolt based on Zerocash. Bolt combines blind signature and zero-knowledge proof to realize anonymous payment, tailor-made for Zcash. Malavolta *et al.* [17] proposed a hash time-lock contract suitable for multi-hop payments in Bitcoin. However, this scheme uses zero-knowledge proof to protect user privacy, and the calculation and storage overhead is considerable. At the same time, this scheme does not support multi-path payment. In 2019, Malavolta G*et al.* [18] then provided a single-path off-chain payment scheme called Anonymous Multi-Hop Locks (AMHL), which is built with one-way homomorphic functions for those that support scripting languages and built with ECDSA signatures for those only support simple scripting languages. However, AMHL is challenging to scale to multi-path payment. In 2021, Lukas Aumayr *et al.* [15] implemented a payment protocol – Blitz, based on a round of multi-hop payment forwarding, which only needs digital signatures and time lock functions to construct off-chain payment. At the same time, Blitz is also resistant to wormhole attacks.

Thyagarajan *et al.* [19] proposed an off-chain payment protocol compatible with any signature scheme, using scriptless scripts to construct BLS signatures. These signatures are short, unique, and aggregatable and have a more negligible overhead, which can be used for cross-chain atomic payments. In 2022, the same authors proposed an off-chain atomic swap protocol [20] based on adapter signatures and timelock, which can be applied to multiple asset transfers in different blockchains. In addition, considering that the long payment path of a single-path payment leads to large time consumption and increased overhead, some solutions are based on the payment channel hub [21].

2) Multi-path payment protocol: In the single-path payment protocol, only one payment path is allowed, while bifurcation is not. As the capacity of the payment channel is fixed, it cannot handle payments larger than the existing channel balance. Thus, it is difficult for payments to find a suitable path from the off-chain network for larger amounts. If payment can be divided into multiple sub-paths, this will allow more routing options for payment forwarding.

In 2020, Eckey *et al.* [22] proposed an off-chain payment scheme named SplitPay, which dynamically divides paths by intermediaries and uses homomorphic encryption to process payment information during the forwarding phase. However, since all partial routing nodes must know the receiver's public key when dividing the path, malicious nodes can link payments through the same public key of the receiver, so the receiver's privacy cannot be guaranteed. In 2022, Subhra Mazumdar *et al.* [23] proposed a privacy-preserving atomic multi-path payment protocol-CryptoMaze, extending the xLumi [24] protocol and uses discrete logarithm operations to set the conditional value of the off-chain contract, which can resist the wormhole attack of the multi-path off-chain payment protocol.

B. Our Contributions

In Table I, we summarize the main characteristics of existing schemes and the proposed scheme, table shows that LightPay can provide atomicity, unlinkability, wormhole attack resistance, and path privacy. The contributions of this work are described as follows:

TABLE I
COMPARATIVE ANALYSIS OF DIFFERENT METHODS

	A	U	W	M	P
AMHL [18]	✔	✔	✔	✖	✔
Blitz [15]	✔	✖	✔	✖	✔
Multihop HTLC [17]	✔	✔	✔	✖	✔
SplitPay [22]	✔	✔	✖	✔	✖
CryptoMaze [23]	✔	✔	✔	✔	✔
Ours(LightPay)	✔	✔	✔	✔	✔

A, U, W, M, P represent Atomicity, Unlinkability, Wormhole attack resistance, Multi-path, and Path privacy respectively.

- LightPay uses a simple adapter signature and encryption process to implement payments without needing custom complex scripts. The scheme can also provide good compatibility with existing blockchains.
- LightPay uses discrete logarithms to construct the consumption conditions of each payment channel and sets different secret values according to the number of right neighbors of the node to realize the multi-path payment of high-valued payments.
- To achieve atomicity and improve the confirmation speed, LightPay completes the process of claiming coins from receiver to sender in the opposite direction of the payment path when only one output node responds during the contract release phase.
- Experimental analyses are conducted and show that LightPay can achieve security goals such as atomicity and unlinkability. Compared with schemes such as CryptoMaze, LightPay requires approximately 55.6% less communication overhead on average.

C. Organization

The remainder of this work is organized as follows. We present the relevant background in Section II, and introduce the formal definition of the LightPay protocol along with the attack model and security goals in Section III. Section IV provides a security analysis of the protocol, the implementation and the evaluation of the proposed protocol are presented in Section V, and finally, the concluding remarks and future directions of this work are depicted in Section VI.

II. BACKGROUND

In this section, we introduce the background knowledge about the payment network under the blockchain. The symbols used in this work and their meanings are explained in Table II.

A payment channel network can be represented by a graph $\mathcal{G} := (V, E)$, where the nodes V and the edges E in the graph represent the off-chain node P, and the payment channel β in the payment channel network, and two vertices P_i and P_j are connected by an edge. The payment channel β can be defined as a tuple $(\beta.\mathrm{id}, \beta.\mathrm{users}, \beta.\mathrm{coins}, \beta.\mathrm{state})$, where $\beta.\mathrm{id} \in \{0,1\}^*$ is the unique identifier of the channel, $\beta.\mathrm{users} := (P_i, P_j)$ is the users connected on the channel, $\beta.\mathrm{coins} := \beta.\mathrm{users} \rightarrow R^{\geq 0}$ is the distribution of coins of each user on the channel, $\beta.\mathrm{state} := (\theta_1, \cdots, \theta_n)$ represents the state of the current channel, which is composed of a list of coins distribution updates θ_i and will be updated after each off-chain payment is completed.

TABLE II
NOTATION DESCRIPTION

Notations	Description
$\mathcal{G} := (V, E)$	Off-chain payment channel network graph
V	The sets of nodes in the graph $\mathcal{G}$
E	The sets of payment channels in the graph $\mathcal{G}$
$\mathbb{E}$	The set of payment channels in one payment
$\mathcal{L}$	Blockchain
ID_{ij}	The unique identifier of the payment channel $(U_i, U_j) \in E$
S_{ij}	Legal signature of payment channel ID_{ij}
G	A cyclic multiplicative group with large prime order q
g	The generator of G
p	A large prime number, where $p = 2q + 1$
Δ	The maximum transaction confirmation time in blockchain
U_s	Sender of an off-chain payment
U_r	Receiver of an off-chain payment
T_i	The timestamp of the first contract request received
τ	The waiting time after receiving the contract request

A. Payment Channel

There are three operations when using a Payment Channel (PC): open channel, update channel, and close channel. Among them, both open and closed channels need to interact with the blockchain. The update channel only involves both parties to the payment, and uploading the payment state to the blockchain is unnecessary. A brief introduction to these three operations follows next.

Open channel. To open a PC, two users must post a transaction on-chain to lock the collateral in a shared address. For example, Alice and Bob must provide coins α_i, α_j respectively to build a PC β with the following parameters: capacity $\alpha_i + \alpha_j$, $\beta.\mathrm{id}$, $\beta.\mathrm{users} := (\mathrm{Alice}, \mathrm{Bob})$, $\beta.\mathrm{coins}\,(\mathrm{Alice}) = \alpha_i, \beta.\mathrm{coins}\,(\mathrm{Bob}) = \alpha_j$, $\beta.\mathrm{state} = \varnothing$. When the channel is opened, β is added to G.

Update channel. Two users need to update the current channel state after paying off-chain. Specifically, when Alice needs to pay $\alpha_a (\alpha_a \leq \alpha_i)$ to Bob, a new state $\theta_i = (\beta.\mathrm{coins}\,(\mathrm{Alice}) - = \alpha_a, \beta.\mathrm{coins}\,(\mathrm{Bob}) + = \alpha_a)$. After θ_i is confirmed by Alice and Bob, it will be appended to $\beta.\mathrm{state}$, thus completing the update of the current channel. This process does not require the payment to be posted on-chain.

Close channel. Two users claim their coins by posting the latest state to the chain, and the PC will be closed. At the same time, the channel β needs to be removed from G. In particular, when Alice does not receive Bob's confirmation message for a long time after sending Bob a request to update the state, Alice can close the channel and submit the latest state to the chain. Within Δ time, if Bob cannot provide a newer version of the PC state, the state submitted by Alice will be accepted on-chain.

Nodes need to lock certain collateral on-chain to build a PC, since each node cannot establish PCs with all nodes, as it is very resource-consuming. The Payment Channel Network (PCN) is composed of several PCs, so two nodes that are not directly connected can complete an off-chain payment through the participation of other intermediaries. For example, suppose A and E do not build a PC on-chain; if A intends to pay

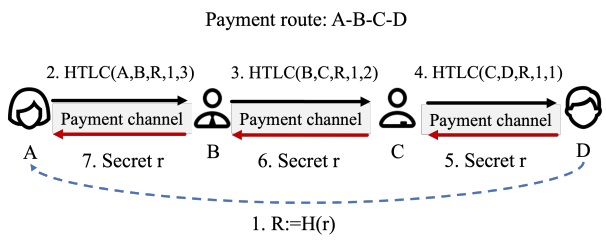


Fig. 2. Build off-chain payment through HTLC.

coins to E, the payment can be made through a PCN, i.e., $A \rightarrow B \rightarrow C \rightarrow E \rightarrow F$, called multi-hop payments.

B. Hash Time-Lock Contract

For the convenience of understanding, we omitted the setting of the transaction fees in the construction of HTLC. Figure 2 shows the specific process of using HTLC for off-chain payment. First, the receiver will generate a conditional value $R = Hash(r)$ and send it to the sender. The sender will generate an HTLC with the address of both nodes, condition value, payment amount, and expiration time and send it to the neighbor node. The neighbor node then constructs HTLC with a similar structure to the next neighbor node until the receiver. Then, the receiver will reveal the hash pre-image r from the condition value and pass it to each node to update the balance of the channel. It is worth noting that hash time-locks include hashlocks and timelocks. Hashlocks mean that the corresponding coins can only be unlocked by revealing the pre-image value of the hash, while timelocks denote that the coins can only be spent until a specific time in the future.

In other words, each node must reveal the hash pre-image within the expiration time set in HTLC to complete payment. However, since the condition value R of each payment is the same in each channel, any two nodes can quickly determine whether they belong to the same transaction and thus can further infer the sender and receiver of the transaction, which leaks path privacy for off-chain payments.

C. Wormhole Attack

HTLC sets the same pre-image for each PC in the payment path to complete the off-chain payments. During the payment phase, the payment receiver passes the pre-image forward to complete the payment for each channel. We assume there is a payment path of $A \rightarrow B \rightarrow C \rightarrow E \rightarrow F$, and A needs to pay α_f fees to B, C, and E, respectively. The receiver F will create a pre-image r, calculate the hash value $R = Hash(r)$, and then send R to the sender A. Then A constructs the payment paid to B using R and delivers R according to the payment path until it is passed to F for confirmation. After F checks the correctness of R, the pre-image r is sent to the previous node until it is delivered to the sender, a complete HTLC protocol process.

However, since each node uses the same pre-image, it is vulnerable. For example, in the payment phase, F sends r to E to complete the payment of $E \rightarrow F$. Next, there may be an attack situation: if E and B collude, E can send r to B directly, and then B sends r to A, to claim coins. Thus B and E conspire to steal C's fees, but A cannot detect the abnormality, which is a wormhole attack. The wormhole attack causes honest nodes to fail to get the transaction fees for forwarding transactions. When designing the off-chain payment scheme, we need to avoid the occurrence of a wormhole attack.

III. The Proposed Protocol

A. Security Goals

Currently, most off-chain payment schemes are only for single-path payments and do not allow splitting payments. We define the following security goals:

- **Atomicity**. Each payment must be atomic; that is, all payments involved in the payment process either succeed or fail. Otherwise, some honest nodes will lose coins.
- **Unlinkability**. It ensures that nodes participating in partial payments cannot know the complete payment information through collusion. It can prevent malicious nodes participating in partial payments from deliberately refusing to participate in certain forwarding payments.
- **Wormhole attack resistance**. When constructing the off-chain contract of each PC in the payment path, it is necessary to properly set the secret value to ensure that malicious nodes cannot skip honest nodes to complete payment. The protocol must resist wormhole attacks and prevent honest nodes from losing their fees.
- **Balance security**. Honest nodes will never lose their coins when participating in off-chain payments.
- **Path privacy**. Any intermediate node can only interact with its direct neighbors and cannot know the information of other nodes in the entire payment path.
- **Endpoint privacy**. Any intermediate node cannot know whether its left (right) node is the sender (receiver) of the payment or an ordinary intermediate node.

B. Cryptographic Preliminaries

We assume that G is a cyclic multiplicative group with large prime order q. g is the generator of G, p is a prime number, where $p = 2q + 1$. $a,\ b \in Z_q$ are secret random numbers. We consider all elements in G to be in the group Z_p, i.e., the style of the $g^a \ mod\ p$ element. For readability purposes, we omit $mod\ p$ in the remainder of this work. The proposed protocol is mainly based on the following two computationally hard problems. (i) Discrete Logarithm Problem (DLP): Even if the attacker $\mathcal{A}$ knows $g^a \in G$, it is computationally hard to compute a in polynomial time.

Standard hash function. The cryptographic hash function is a one-way function: $H : \{0,1\}^* \rightarrow \{0,1\}^\lambda$, where λ is a security parameter used in the model. Given an input of any length, H produces a unique fixed-length output.

Adapter signature. An adapter signature scheme [25] contains four algorithms $\Sigma = (PreSig, PreVf, Adapt, Ext)$. $\hat{\sigma} \leftarrow PreSig(sk, mes, Y)$ is a pre-signature algorithm, where sk represents the secret key, mes represents the message that needs to be pre-signed, Y is a statement, and the output

is the pre-signature $\hat{\sigma}$. $\sigma := Adapt(\hat{\sigma}, y)$ is an adaptation algorithm that inputs the pre-signature $\hat{\sigma}$ and the witness y, and outputs the legal signature σ. $PreVf(pk, \hat{\sigma})$ is a pre-signature verification algorithm that verifies whether the pre-signature is correct. $y := Ext(\hat{\sigma}, \sigma, Y)$ can obtain the witness y by inputting the pre-signature $\hat{\sigma}$ and signature σ and statement Y.

C. Formal Definition

U_s wishes to effectively transfer val to U_r via $\mathcal{G} := (V, E)$, where U_s, $U_r \in V$. The LightPay protocol includes a pre-processing phase, a contract forwarding phase, and a release phase. U_s constructs an off-chain multi-path payment through a routing protocol [26], generates a payment channel set $\mathbb{E}$, and uses it as the input to the pre-processing phase. It is worth noting that HushRelay is a privacy-preserving distributed routing protocol. Compared with the SpeedyMurmurs [27], HushRelay can protect the privacy of transaction sender s and the receiver r by constructing a dummy source node s' and a target node r'. Specifically, by adding directed virtual edges of $s' \rightarrow s$ and $r \rightarrow r'$ to generate the path, we prevent attackers from guessing the identity of the transaction recipient and sender, thereby ensuring the privacy of the routing protocol. At the same time, since the proposed routing protocol is modular, it can be used in conjunction with any other privacy-preserving payment protocols. Next, each phase of the payment protocol is introduced in detail.

1) Pre-processing Phase: This phase must set the conditional value for each channel in the payment path. Its input is the channel set $\mathbb{E}$, and the output is the encrypted information sent to the next neighboring node. We divide this phase into the following phases, as depicted in Algorithm 1.

a) U_s calculates payment secret value: U_s generates a random number r as the payment secret value and starts to build a signed hidden value for each off-chain contract. The process is the opposite of the payment process; it builds from U_r. U_s first checks the number of input channels of U_r, the number of all direct neighbor nodes of U_r. If there are z input channels $ID_{ir} \in \mathbb{E}$, then U_s selects z random numbers $n_i \in Z_q$ for U_r, and calculates $x_{ir} := n_i * ID_{ir} - \sum_{t=1}^{z} n_t * r$. Among them, x_{ir} is the key information of the contract signature S_{ir} calculated by U_r in ID_{ir}, and $ir = n_i * ID_{ir}$ is the hidden value of S_{kr_i}. In the contract release phase, U_s will provide r to U_r, and U_r can calculate the signature S_{ir} through r to release the contract on ID_{ir} and ask U_i for the fee.

b) Signature of off-chain contracts: According to the different topological structures of different PCNs, LightPay constructs the signatures of off-chain contracts for the following four situations in Figure 3: (1) Scenario 1: The intermediate node U_j has an input node and an output node. (2) Scenario 2: U_j has multiple input nodes and one output node. (3) Scenario 3: U_j has one input node and multiple output nodes. (4) Scenario 4: U_j has multiple input nodes and multiple output nodes. Depending on the different scenarios, the calculation process of the contract signature is also different. The details are shown as follows:

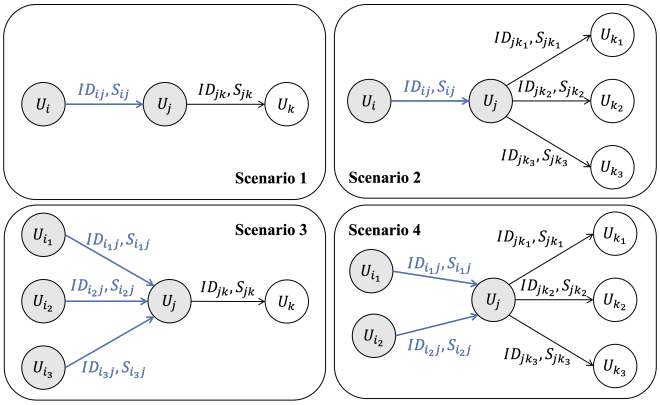


Fig. 3. Node distribution of different scenarios.

Scenario 1: If any intermediate node U_j forwards payment to a single node U_k.

S_{ij} and S_{jk} denote the legal signatures of channel ID_{ij} and ID_{jk} respectively. If $U_j \neq U_r$ only forwards payments to one neighbor U_k, for a pair of channels ID_{ij} and ID_{jk}, with signatures S_{ij} and S_{jk}. U_s calculates x_{ij} for U_j according to Equation 1:

$$x_{ij} = n * ID_{ij} - jk, \tag{1}$$

$$S_{ij} = x_{ij} + jk + H\left(m \parallel IJ \parallel P_i\right) * p_i, \tag{2}$$

where $n * ID_{ij} = ij$ is the hidden value in the signature S_{ij}, and n is a random number. In the contract forwarding phase, U_j judges whether $JK * g^{x_{ij}} = IJ$ is equal to determine the consistency of the input and output contracts $(IJ = g^{ij})$. In the contract release phase, U_j calculates the signature S_{ij} of ID_{ij} through Equation 2.

Scenario 2: The number of input nodes of U_j exceeds 1.

To achieve atomicity in the contract forwarding phase, it must be guaranteed that only when all the input contracts are received, U_j can send the contract to the output node. We assume that the number of input channels of U_j is z=3, so U_s selects z random numbers n_i, and calculates:

$$x_{i_1 j} = n_1 ID_{i_1 j} - \sum_{t=1}^{z} n_t * jk, \tag{3}$$

$$x_{i_2 j} = n_2 ID_{i_2 j} - \sum_{t=1}^{z} n_t * jk, \tag{4}$$

$$x_{i_3 j} = n_3 ID_{i_3 j} - \sum_{t=1}^{z} n_t * jk, \tag{5}$$

where $n_i * ID_{i_t j} = i_t j$ is the hidden value in $S_{i_t j}$. Each ij_t needs to know $\sum_{t=1}^{z} n_t$ first, and only after the intermediate node U_j receives all the input contracts, U_j sends the contract to the output node U_k. In the contract forwarding phase, U_j judges $g^{x_{i_m j}} * (JK)^{\sum_{t=1}^{z} n_t} = I_m J$ is equal to determine the consistency of the input and output contracts.

Scenario 3: U_j has multiple output nodes.

If U_j forwards the payment to multiple neighbor nodes, it must be ensured that during the release phase, when one of the neighbors U_{k_t} fails to provide the signature of the ID_{jk_t}, U_j will not lose its coins. To solve this problem, LightPay employs a 1-out-of z strategy, when at least one output node U_{k_t} returns a signature S_{jk_t}, U_j can also claim coins from U_i.

In Figure 3 Scenario 3, U_j splits the payment to nodes U_{k_1}, U_{k_2} and U_{k_3}. Suppose U_j uses the same hidden value for nodes U_{k_1}, U_{k_2} and U_{k_3}, then $jk_1 = jk_2 = jk_3$. If U_{k_1}, U_{k_2} and U_{k_3} collude, they can link their payments. To avoid this problem, U_s assigns different jk_1, jk_2 and jk_3 to channels ID_{jk_1}, ID_{jk_2} and ID_{jk_3}. Therefore, U_s calculates:

$$S = \sum_{t=1}^{z} jk_t, \tag{6}$$

$$x_{ij_t} = nS * ID_{ij} - jk_t, \tag{7}$$

where $nS * ID_{ij} = ij$. In the contract forwarding phase, the intermediate node U_j can aggregate and verify the consistency of the input and output contracts by judging whether $g^{x_{ij_t}} * JK_t = IJ$ is equal. In the contract release phase, U_j only needs to know any jk_t to calculate the hidden value ij of S_{ij} through Equation 7.

Because jk_1, jk_2, and jk_3 are not equal to each other, even if the off-chain contract is settled on-chain, miners still cannot link these three transactions.

Scenario 4: U_j has multiple output and input nodes.

We combine Scenario 2 and Scenario 3 to construct the hidden value calculation formula of Scenario 4. Assuming that U_j has y input nodes and z output nodes, we need to ensure that only after U_j receives the output contracts from U_{i_1} and U_{i_2}, the corresponding input contract can be sent to U_{k_1}, U_{k_2} and U_{k_3}. We achieve the atomicity of the input contract by selecting m different random numbers. At the same time, we construct S to implement a 1-out-of z strategy. The calculation formulas of S and x_{ij} are as follows:

$$S = \sum_{t=1}^{z} jk_t, \tag{8}$$

$$x_{i_1 j_t} = n_1 S * ID_{i_1 j} - \sum_{m=1}^{y} n_m * jk_t, \tag{9}$$

where $n_m S * ID_{i_m j} = i_m j$. Similarly, during the contract forwarding phase, the intermediate node U_j can determine whether $g^{x_{i_m j_t}} * (JK_t)^{\sum_{m=1}^{y} n_m} = I_m J$ is equal to verify the consistency of the contract.

c) Set timeout: The minimum timeout allocated to all upcoming contracts of U_i is denoted as t_{end}. From this point on, all previous contract timeouts are determined.

2) Contract Forwarding Phase: Each node U_i maintains variables in_i, out_i, $success_i$ and T_i, where in_i and out_i represent the total amount of the input and output contracts, respectively, both initialized to 0. $success_i$ is initialized to $false$, indicating that all input contracts have not been received yet. When all input contracts are received, $success_i$ is set to $true$. T_i is set to the current time when U_i receives its first input contract request. U_i waits for time $T_i + \tau$ to receive all input contract requests, where $\tau > 0$ is the waiting delay. If the elapsed time is greater than $T_i + \tau$, but $success_i$ is still $false$, then U_i will send a quit message to the previous contract to cancel the payment.

Algorithm 1: Pre-processing phase for node U_s

Input: Payment channel set $\mathbb{E} = ID_{s1}, ID_{12}, \ldots, ID_{nr}$
Output: The encrypted information E_{S1} to U_1

```
Generate a random number r as a secret value
Compute R = g^r mod p
foreach incoming node j in U_r do
    compute n_j ∈ Z_q
    Sum up all n_j to N
end
x_jr = n_j * ID_jr − N * r
jr = n_j * ID_jr
JR = g^jr mod p
Encrypt (x_jr, val, R, t_end) to E_jr with pk_r
foreach edge ID_ij in Edges do
    x_ij = n * ID_ij − jr
    Encrypt (x_ij, val_jr, JR, t_jr, E_jr) to E_ij with pk_j
end
return E_S1
```

For the convenience of understanding, we explain the situation in Scenario 1 here, and the algorithm is shown in Algorithm 2. Starting from node U_s, after receiving the input contract (that is, $success_i$=$true$) and checking the correctness of related messages, U_i first constructs an adapter signature $S'_{ij} = H(m \parallel IJ \parallel P_i) * p_i$ for its neighbor U_j, where m is the payment information of U_i paying U_j. Then sends a request to U_j to build a contract $Re_{ij} = (IJ,\ val_{ij},\ t_{ij}, S'_{ij})$. U_j will verify the adapter signature, and after the verification is successful, it will send $accept$ information to U_i. After U_i receives the $accept$ message, it sends the encrypted message E_{ij} to U_j. After decrypted, U_j can know $D_j = \{(ID_{jk}, x_{ij}, val_{jk}, JK,\ t_{jk},\ E_{jk}) : \forall U_k \in V,\ ID_{jk} \in \mathbb{E}\}$, where E_{jk} is the encrypted message to be forwarded to node U_k. U_j judges the consistency of the input and output contracts by judging whether $JK * g^{x_{ij}} = IJ$ is true. If the return fails, U_j sends a quit message to U_i to cancel all off-chain contracts established with the previous nodes. At the same time, U_j also needs to ensure $in_i = out_i + fee$. When all verification is successful, U_j starts to forward the payment contract to its output nodes.

This phase continues until all payments reach U_r. To ensure the atomicity of the payment, only when all the input contracts of U_r arrive, U_r calculate the hidden value kr of S_{kr} and can claim the coins from U_k. Once U_r has received all the payment contracts within a limited time, it will send the payment confirmation information to U_s. After U_s receives the payment confirmation information, it will send the value r to U_r through a secure channel, thus triggering the contract release phase.

3) Release Phase: For the convenience of understanding, we explain the situation in Scenario 2. U_r has z input nodes

Algorithm 2: Contract forwarding for $U_j \in V$

Input: $Re_{ij} = (IJ, val_{ij}, t_{ij}, S'_{ij})$ from U_i
Output: E_{jk}
if $PreVf\left(pk_i, S'_{ij}\right) = 0$ **then**
  send $quit$ to U_i
end
Initialize in_j=0, out_j=0, T_j=$time_{cur}$, $success_i$=$false$
$in_j = in_j + val_{ij}$
Accept forward request, then receive E_{ij} from U_i
Decrypt E_{ij} to get $D_j = \{(ID_{jk}, x_{ij}, val_{jk}, JK, t_{jk}, E_{jk}) : \forall U_k \in V,\ ID_{jk} \in \mathbb{E}\}$
foreach U_k in D_j **do**
  if $JK.g^{x_{i,j}} \neq IJ$ **or** $t_{ij} < t_{jk} + \Delta$ **then**
    send $quit$ to U_i
  end
  $out_j = out_j + val_{jk}$
end
if $out_j + fee > in_j$ **then**
  wait for $T_j + \tau$
  if $time\ reached\ the\ T_j + \tau$ AND $success_i = false$ **then**
    Send $quit$ to U_i
  end
else
  $success_i = true$
  $S'_{jk} \leftarrow PreSig\left(sk_j, m, JK\right)$
  send Re_{jk} to U_k
  if receive $accept$ from U_k **then**
    Send E_{jk} to U_k
  end
end

U_j, and the relevant algorithm is shown in Algorithm 3. U_r can obtain $(JR,\ val_{jr},\ t_{jr}, S'_{jr})$ from Re_{jr}, and can decrypt E_{jr} to obtain $(x_{jr}, n_j, val, R_0, t_r)$. Then U_r sends payment confirmation information to U_s. After receiving the r value from U_s, U_r first checks the correctness of r. That is, verifying whether $R_0 = g^r$ is true. Next, U_r calculates $x_{jr} = n_j * ID_{jr} - \sum_{j=1}^{z} n_j * r$ to obtain $jr = n_j * ID_{jr}$, which can be used to obtain the signature $S_{jr} = jr + H\left(m \parallel P_j \parallel IR\right) * p_j$. If the signature is correct, U_r sends the success message to its left neighbor with the signature S_{jr}, and claims coins from its left neighbor; otherwise, it sends a quit message to the left neighbor, and the payment fails. After the U_j receives S_{jr}, it verifies the correctness of the signature and calculates the signature S_{ij} of the payment for ID_{ij} (the process is similar to the above process) to claim coins from the left neighbor.

Any intermediate node participating in the forwarding adapter signature payment, as long as at least one output node returns the correct signature, the current node can ask the input node for coins. Since the pre-signature will be included in the request payment information Re in the contract forwarding phase, the current node only needs to be able to calculate the hidden value of the adapter signature of the payment. The calculation of the hidden value depends on the pre-signature, legal signature, and conditional value provided by the output node through $jr := Ext(S'_{jr}, S_{jr}, JR)$. In other words, the signature provided by the output node will help the current node calculate the signature of the payment paid by the input node to the current node. This release process will continue until the sender U_s, so the payment is successful.

Algorithm 3: Release Phase for Node U_j

Input: $r, \left(JR,\ val_{jr},\ t_{jr}, S'_{jr}\right), (x_{jr}, n_j, val, R_0,\ t_r)$
Output: S_{ij}
if $U_j = U_r$ **then**
  if $R_0 \neq g^r$ **then**
    send $quit$ to previous node
  end
  Initialize $R = 0$
  foreach n_j from D_r **do**
    $R = R + n_j$
  end
  $R = R * r$
  $jr = x_{jr} - R$
  $S_{jr} := Adapt(S'_{jr}, jr)$
  Send S_{jr} to U_j
else
  Initialize $ij = 0$
  Receive S_{jr} from U_r
  $jr := Ext(S'_{jr}, S_{jr}, JR)$
  foreach n_i from D_j **do**
    $ij = ij + n_i$
  end
  $ij = x_{ij} - ij * jr$
  $S_{ij} := Adapt(S'_{ij}, ij)$
  Send S_{ij} to previous node
end

IV. Security Analysis

We formalize LightPay's security and privacy using a Universal Composability (UC) framework [28] and rely on a synchronized version with the global UC (GUC) framework [29]. Compared with the UC framework, GUC allows global settings to model the blockchain as a global ledger $\mathcal{F}_{\mathcal{L}}$. The security of the GUC framework can allow the concurrent combination of protocols, which means that a protocol can remain secure when executed simultaneously with any other protocol.

A. Adversary model and communication model

Adversary model. We assume that the adversary of this protocol is $\mathcal{A}$ using a Probabilistic Polynomial Time (PPT) algorithm. We adopt a static corruption model, where the adversary $\mathcal{A}$ must specify the node to attack before the protocol starts [18], [23]. $\mathcal{A}$ can control and obtain the internal status and message transmission methods of some or even all intermediaries and send and receive arbitrary messages on behalf of these nodes.

Communication model. We consider a synchronous communication network where parties communicate in rounds with an ideal functionality $\mathcal{F}_{LP}$ [30]. All messages a user sends in one round can reach the recipient in the next round. At the same time, we assume that the parties communicate through the authenticated channel $\mathcal{F}_{smt}$. The adversary can change the order of sending messages in the same round but cannot modify or delete the messages. Furthermore, we assume that the integrity of the communication process and identity between honest nodes is not compromised by $\mathcal{A}$.

B. Universal Composability (UC) Security

A protocol Π UC-emulates the ideal functionality $\mathcal{F}_{LP}$ if environment Z is indistinguishable for any PPT between the output of the actual run of the protocol and the simulation of the ideal functionality $\mathcal{F}_{LP}$. Π is defined as a hybrid protocol that has access to an ideal functionality $\mathcal{F}_{\mathcal{C}}$ consisting of a global ledger $\mathcal{F}_{\mathcal{L}}$ and an authenticated channel $\mathcal{F}_{smt}$. Environment Z can receive output messages from real-world and ideal-world parties and provide input to them. The ideal functionality $\mathcal{F}_{LP}$ is attacked by the ideal-world simulator Sim using the PPT algorithm. We use $\text{EXEC}^{\mathcal{F}_{\mathcal{C}}}_{\Pi,\mathcal{A},Z}$ to represent the output set of environment Z when adversary $\mathcal{A}$ interacts with the user running protocol Π, and $\text{EXEC}^{\mathcal{F}_{\mathcal{C}}}_{\mathcal{F}_{LP},Sim,Z}$ to represent the output set of environment Z when the functionality $\mathcal{F}_{LP}$ interacts with the simulator Sim. The formal definition of security is as follows.

Definition 1: If there is a simulator Sim for attacking $\mathcal{F}_{LP}$ for any PPT adversary $\mathcal{A}$ that attacks the real protocol Π, given λ is a security parameter such that no matter which protocol the PPT environment Z uses as a test, we have:

$$\text{EXEC}^{\mathcal{F}_{\mathcal{C}}}_{\Pi,\mathcal{A},Z} \approx \text{EXEC}^{\mathcal{F}_{\mathcal{C}}}_{\mathcal{F}_{LP},Sim,Z}, \tag{10}$$

where $\approx$ represents the computational indistinguishability. In this case, the protocol Π GUC-realizes the ideal function $\mathcal{F}_{LP}$.

C. Ideal Functionality of LightPay

Notation. We define the ideal functionality $\mathcal{F}_{LP}$ for LightPay, where honest nodes in the network are simulated as an interactive Turing-machine. These nodes are called virtual nodes P, and they communicate with each other through the ideal function $\mathcal{F}_{LP}$. In addition, U_s and U_r represent the payment's sender and receiver, respectively. Ideal functionality $\mathcal{F}_{LP}$ internal maintenance list $\mathcal{P}$ stores the closed channel list. The list $\mathcal{P}$ has the format $(ID_{ij}, v^{all}_{ij}, t^{ex}_{ij}, fee_{ij})$, where ID_{ij} represents the identifier of the payment channel between virtual parties U_i and U_j, v^{all}_{ij} represents the total capacity of the channel, t^{ex}_{ij} represents the expiration time of the channel, and fee_{ij} represents the cost of the channel ID_{ij} transaction. In addition, $\mathcal{F}_{LP}$ uses list $\mathcal{W}$ to store the off-chain payment list, represented by $(ID_{ij}, v^{re}_{ij}, t_{ij}, Tx_{ij}, S'_{ij})$, where v^{re}_{ij} is the remaining capacity of the channel, t_{ij} is the expiration time of the payment, Tx_{ij} is the payment event identifier, and S'_{ij} represents the adapter signature. In the pre-processing phase of LightPay, U_s generates a session sid and builds a payment path in $\mathcal{L}$ from U_s to U_r that can pay the amount val and supports the expiration time t_{end}. The set of all payment channels in the payment path is $\mathbb{E}$, and U_s sets the payment amount val_{ij} and payment expiration time t_{ij} of each channel according to the fee_{ij} of each channel in $\mathbb{E}$.

We define the EXECUTION operation in the ideal-world, which is divided into two phases: (1) Contract forwarding phase and (2) Release phase.

Contract forwarding phase. As depicted in Figure 4, after U_s sends $(sid, Payment, U_r, val, t_{end}, \{(ID_ij, val_{ij}, t_{ij}, S'_{ij}) : ID_{ij} \in \mathbb{E}\}, \mathbb{E})$ to $\mathcal{F}_{LP}$, $\mathcal{F}_{LP}$ starts this phase and add U_s into the queue S_p. First, $\mathcal{F}_{LP}$ will check whether there is an open channel ID_{ij} in $\mathcal{L}$ and whether there is sufficient channel balance. It also checks the correctness of timeout, transaction amount, adapter signature, and the consistency of input and output contracts. If these conditions fail, $\mathcal{F}_{LP}$ will delete all off-chain payment entries in $\mathcal{W}$ and terminate the protocol. If all conditions are true, $\mathcal{F}_{LP}$ forwards part of the payment to the next neighboring node U_j. If all input contracts of U_j hold, it is inserted into S_p. If U_j send $abort$, all entries in $\mathcal{W}$ are deleted and $\mathcal{F}_{LP}$ $abort$.

Release phase. When the partial payment is forwarded to U_r, U_r sends a response message (sid, mes) to $\mathcal{F}_{LP}$ to trigger this phase. $\mathcal{F}_{LP}$ initializes queue $\mathcal{R}$ and adds U_r to the queue. If $mes = abort$ sent by U_r, indicating that the payment process failed, $\mathcal{F}_{LP}$ delivers $abort$ to all input nodes of U_r and deletes all entries related to the payment from $\mathcal{W}$. If U_r sends a success message, $\mathcal{F}_{LP}$ will deliver the message to U_r's input nodes, update the entry in $\mathcal{W}$, and send the success message to its input node. We also use vis to denote the access status of nodes. If another input node of U_r sends $abort$, the node is marked as vis and added to S_f.

D. Discussions

In this subsection, we discuss how $\mathcal{F}_{LP}$ captures the security and privacy properties defined by LightPay in Section III.

Atomicity. The challenge with atomicity is how we design conditional payments. Therefore, we use adapter signatures to generate pre-signatures for payments between nodes. The correct signature can be generated when the next neighboring node obtains the hidden value, and once the current node knows this correct signature, it can also obtain the hidden value by calculation.

As shown in Figure 5 (assuming the payment path is $S \rightarrow A \rightarrow B \rightarrow R$), LightPay ensures the atomicity of all sub-paths in the payment path. In the pre-processing phase, S generates the corresponding conditional value SA and pre-signature S'_{SA} for ID_{SA}. Since the pre-signature is not a real signature, it cannot be used as a payment certificate for off-chain state updates. In the contract forwarding phase, S forwards the puzzle, and the pre-signature and the intermediate node sequentially pass the corresponding information to the next neighboring node. Since the intermediate node received the pre-signature, the coins cannot be released temporarily. The release phase is triggered when the last node R receives the information. Since R is the receiver, it can be obtained by solving the puzzle, and r can be obtained to adapt the

Contact forwarding phase
On input $(sid, Payment, U_r, val, t_{end}, \{(ID_{ij}, val_{ij}, t_{ij}, S'_{ij}) : ID_{ij} \in \mathbb{E}\}, \mathbb{E})$ from U_s, $\mathcal{F}_{LP}$ proceeds as follows:
1. For each $ID_{ij} \in \mathbb{E}$, set $ct(sid, ID_{ij}) = 0$
2 Form a set $V_{\mathbb{E}} = U_i \cup U_j$ such that $U_i, U_j \in V_{\mathbb{E}}$ and $ID_{ij} \in \mathbb{E}$.
3. Initialize empty queue S_p. Add U_s into S_p.
4. While S_p is not empty:
i. Add U_i into S_p.
ii. For each $U_j \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}$:
- If U_j send $(sid, abort)$ to $\mathcal{F}_{LP}$ then $abort$.
- If $PreVf(pk_i, S'_{ij}) = 0$ then $abort$.
- If ID_{ij} not in $\mathcal{L}$ or in $\mathcal{P}$ then $abort$.
- Form $z_{ij} = \{(ID_{jk}, val_{jk}, t_{jk}, S'_{jk}) : \forall U_j \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}\}$, if $U_j \neq U_r$. Else $z_{ir} = \{val, t_{end}\}$.
- If $t_{ij} < \Delta + max_{U_k \in V_{\mathbb{E}}: ID_{jk} \in \mathbb{E}} t_{jk}$ and $\sum_{U_i \in V_{\mathbb{E}}: ID_{ij} \in \mathbb{E}} val_{ij} \neq \sum_{U_k \in V_{\mathbb{E}}: ID_{jk} \in \mathbb{E}} val_{jk} + fee(U_j)$ then $abort$.
- If $v^{re}_{ij} < val_{ij}$ then $abort$. Else add $d_{ij} = (ID_{ij}, v^{re}_{ij} - val_{ij}, t_{ij}, abort)$ into $\mathcal{W}$.
- Set $ct(sid, ID_{ij}) = 1$. Sample an identifier Tx_{ij}.
- Send $(sid, forward, U_i, ID_{ij}, val_{ij}, t_{ij}, Tx_{ij}, z_{ij})$ to U_j.
- For all $U_i \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}$. If $ct(sid, ID_{jk}) = 0$ then abort. Else add U_j into S_p.
Release phase
On input (sid, mes) from U_r, $\mathcal{F}_{LP}$ proceeds as follows:
1. For each $U_j \in V_{\mathbb{E}}$, Set $vis(U_j) = 0$.
2. Initialize empty queues $\mathcal{R}$ and S_f.
3. Add U_r to $\mathcal{R}$.
4. While $\mathcal{R}$ is not empty:
i. Pop node U_j from $\mathcal{R}$.
ii. For each $U_i \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}$ and $ct(sid, ID_ij) = 1$:
- Set $ct(sid, ID_ij) = 0$.
- If $mes = abort$ then:
Remove d_ij from $\mathcal{W}$.
Send $(sid, abort, Tx_ij)$ to U_i and U_j..
- Else if $mes = success$.
Update $d_{ij} \in \mathcal{W}$ to $(-, -, -, Tx_{ij})$, send $(sid, success, Tx_{ij})$to U_i and U_j.
If U_i sends $(sid, abort)$ then $vis(U_i) = 1$, add U_i into S_f.
Else if $vis(U_j) = 0$ and $U_i \neq U_s$, set $vis(U_i) = 1$ and add U_i into $\mathcal{R}$.
5. While S_f is not empty:
i. Pop node U_j from S_f.
ii. For each $U_i \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}$ and $ct(sid, ID_{ij}) = 1$:
- Set $ct(sid, ID_{ij}) = 0$. Remove d_{ij} from $\mathcal{W}$.
- If $vis(U_i) = 0$, set $vis(U_i) = 1$, add U_i into S_f.

Fig. 4. The ideal functionality $\mathcal{F}_{LP}$.

pre-signature and generate the correct signature S_{BR}. Then R calculates S_{BR} through r to user B, and B can obtain r_{BR} through S_{BR} to calculate r_{AB}, so the payment is completed when S is introduced to S_{SB} and verified to be correct. If one of the nodes fails to verify the signature, it sends $abort$, indicating that the payment failed. Such a scheme can ensure that when the current node updates the channel state according to the signature, the previous node can obtain the hidden value of the current node's payment according to the signature to calculate the hidden value of the previous node's payment, which ensures the atomicity of the whole process.

Unlinkability. We assume that U_j splits the payment to nodes U_{k_1}, U_{k_2} and U_{k_3}. To mitigate transaction correlation attacks caused by the use of identical hidden value, in the proposed protocol, U_s assigns different jk_1, jk_2 and jk_3 to channels ID_{jk_1}, ID_{jk_2} and ID_{jk_3}, even if the off-chain contract

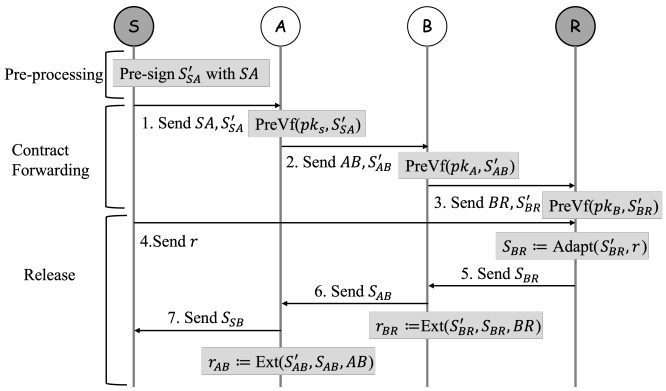


Fig. 5. Atomicity guarantee of LightPay.

is settled on-chain, miners still cannot establish connectivity between these values. At the same time, $\mathcal{F}_{LP}$ will randomly generate an independent identifier id_{ij} for each payment of U_i, and neighbors cannot find any correlation between payment identifiers. The probability that a node participating in a partial payment or miner will link S_{jk_1}, S_{jk_2}, S_{jk_3} without knowing the random number n is negligible.

Wormhole attack resistance. This protocol designs the calculation process of the secret value based on a discrete logarithm problem, so that the secret value of each node U_i is related to the secret values of all its output nodes U_j. In the release phase, When node U_j resolves its off-chain contract, it will send the signature S_{ij} to the previous node, and U_i will be added to the set $\mathcal{R}$. U_i needs signature S_{ij} to calculate the secret value of the input contract. If U_j cannot provide the correct signature, all U_i sends abort. Therefore, no node on the path can be skipped.

Balance security. In the proposed protocol, honest nodes cannot lose their coins. The valid signature of any intermediate node U_j paying to the next neighboring node can restore the valid signature of the previous node U_i paying to the current intermediate node. Even if a malicious node sends a wrong message, an honest node can judge the correctness of the message and send $abort$ to terminate the payment forwarding. If U_i receives $abort$ from all successor nodes, it will also abort, and the balance of all nodes remains unchanged. And when the payment is successful, honest nodes can get their fees.

Path privacy. This protocol uses onion routing to encapsulate forwarded messages to ensure that any intermediate node forwarding a payment can only communicate with its direct neighbor nodes. The information received by each node will be encrypted layer by layer to ensure that only after the previous node decrypts the information, the latter node can decrypt its information. intermediaries cannot decrypt their messages in advance, nor can they decrypt messages belonging to other nodes, thus ensuring path privacy.

Endpoint privacy. For any intermediate node, the message it receives is the information of the input contract $Re_{ij} = (IJ, val_{ij}, t_{ij}, S'_{ij})$ and the information of the output contract $D_j = (x_{ij}, val_{jk}, JK, t_{jk}, E_{jk})$. The message format received by any intermediate node is consistent and does not contain information about the payment sender and receiver, so it is

impossible to judge whether U_i (U_j) is the sender (receiver).

E. Security Analysis

In this section, we utilize the set of real-world and ideal-world outputs of the protocol at different phases to analyze the indistinguishability of environment Z to ideal-world and real-world interactions. We describe the simulator Sim used to handle the situation of nodes corrupted by $\mathcal{A}$ and simulate the execution protocol of the hybrid world while interacting with the ideal function $\mathcal{F}_{LP}$. Sim will maintain the key set of the honest nodes being simulated and, therefore, can decrypt and encrypt information on their behalf. If $\mathcal{A}$ sends any invalid information, then Sim sends the abort instruction to $\mathcal{F}_{LP}$. According to the execution process of the protocol, this work defines three situations of node corruption and corresponding malicious behaviors:

Sender U_s is corrupted, and $\mathcal{A}$ attempts to set a secret value that cannot release the contract. In the contract forwarding phase, $\mathcal{A}$ generates data packets $(IJ, val_{ij}, t_{ij}, S'_{ij}, E_{ij})$ for each $ID_{ij} \in \mathbb{E}, U_i \neq U_r$. Then decrypts E_{ij} to obtain $D_j = \{(ID_{jk}, x_{ij}, val_{ij}, JK, t_{jk}, E_{jk}) : \forall U_k \in V_{\mathbb{E}} : ID_{jk} \in \mathbb{E}\}$, when $U_j \neq U_r$ or $D_j = (x_k r, n_k, R_0)$, if $U_j = U_r$. $\mathcal{A}$ forwards the data packet D_j to Sim. For each node $U_j \in V_{\mathbb{E}} : ID_{ij} \in \mathbb{E}, U_j \neq U_s, U_r$, Sim executes the judgment $PreVf(pk_i, S'_{ij}) = 0$. If failure is returned, then $abort$. Next, Sim constructs the set $M_j = \{(ID_{jk}, x_{ij}, JK, t_{jk}) : \forall U_k \in V_{\mathbb{E}} : ID_{jk} \in \mathbb{E}\}$, and determines $JK * g^{x_{ij}} = IJ$ and $t_{ij} \geq t_{jk} + \Delta$. If success is returned, execution continues. Finally, Sim determines $\sum_{U_i \in V_{\mathbb{E}}: ID_{ij} \in \mathbb{E}} val_{ij} \neq \sum_{U_k \in V_{\mathbb{E}}: ID_{jk} \in \mathbb{E}} val_{jk} + fee(U_j)$; if the check fails, then abort. If the program is not abort, Sim will send $(sid, pay, U_r, val, t_e nd, (ID_{ij}, val_{ij}, t_{ij}, S'_{ij}) : ID_{ij} \in \mathbb{E}, \mathbb{E})$ to $\mathcal{F}_{LP}$. Before forwarding the payment, Sim checks the consistency of the input and output contracts by whether $JK * g^{x_{ij}} = IJ$ is equal. In the release phase, U_r will release the secret value jr, generate the signature S_{jr} of ID_{jr}, and then send it to Sim. Sim will check that jr is generated by $jr := Ext(S_j r', S_{jr}, JR)$, and then check the correctness of the discrete logarithmic value jr by judging whether $ij = jr + x_{ij}$ is equal. When the result returns failure, Sim terminates. Therefore, a contract built by $\mathcal{A}$ with a wrong secret value will be discovered by the Sim and terminated.

Intermediary U_j is corrupted and $\mathcal{A}$ attempts to release U_j's input contract before the secret value is revealed. In the contract forwarding phase, assume that all input and output nodes of U_j are U_i and U_k, respectively. When Sim obtains$(sid, forward, U_i, val_{ij}, t_{ij}, ID_{ij}, id_{ij}, z_{ij})$ from $\mathcal{F}_{LP}$ on behalf of U_i, it will sample jk for each $U_k \in z_{ij}$ and calculate $S = \sum_{U_k \in V_{\mathbb{E}}: ID_{jk} \in \mathbb{E}} jk$. Sim sends $(forward, IJ, val_{ij}, t_{ij}), E_{ij}$ to $\mathcal{A}$ on behalf of U_i. Then $\mathcal{A}$ sends (JK^*, val_{jk}, t_{jk}) for all Sims representing U_k. Sim first checks the correctness of the timeout, transaction amount, and adapter signature, then checks the consistency of the input and output contracts, whether $IJ = g^{x_{jk}} * JK$ is equal. If any check fails, an abort is sent to $\mathcal{F}_{LP}$. In the release phase, U_r generates signatures S_{mr} of all its input contracts and sends them to Sim. Sim will check the signature and extract the secret value of the previous contract. If the check fails, the Sim aborts. Consequently, according to the definition of the protocol, the probability of $\mathcal{A}$ guessing IJ and releasing the input contract without knowing ij is $1/q$.

Receiver U_r is corrupted and $\mathcal{A}$ attempts to release the contract independently. In the contract forwarding phase, U_j is regarded as all input nodes of U_r, and $\mathcal{F}_{LP}$ sends $(sid, forward, U_j, val_j r, t_j r, ID_j r, id_j r, z_j r)$ to Sim representing U_j. Sim takes R_0 from $\mathcal{A}$ and samples n_j, creating $JR = g^n * R_0$, where $n = \sum_{U_j \in V_{\mathbb{E}}: ID_{jk} \in \mathbb{E}} n_j$. It represents all U_j sending $(forward, JR, val_j r, t_j r), E_j r$ to $\mathcal{A}$. In the release phase, Sim representing U_s sends r information to $\mathcal{A}$. $\mathcal{A}$ calculates the secret value of ID_{jr} through r, and then further calculates the signature S_{jr} and sends it to Sim representing U_j. Sim will check the signature S_{jr} and, if successful, continue executing the protocol. If $\mathcal{A}$ generates r and n_j without asking Sim event identifier id_{jr} such that $JR = g^n * R_0$, then send an abort to $\mathcal{F}_{LP}$. Because $\mathcal{A}$ does not know the values of r and n_j, the probability of $\mathcal{A}$ guessing $dlog(JR)$ is $1/q$, that is, the probability of $\mathcal{A}$ independently releasing the contract is $1/q$.

It can be seen that when the sender U_s, the intermediate node U_i, or the receiver U_r is corrupted, the values randomly generated by Sim and the values executed follow the same distribution and are indistinguishable. There is no difference between the operating results of Z in the ideal-world and the hybrid-world. Therefore, we prove that LightPay GUC-realizes the desired functionality $\mathcal{F}_{LP}$ and has the same security as $\mathcal{F}_{LP}$ as defined in Section III.

V. Performance Analysis

A. Implementation Details

The experimental platform comprises one Intel(R) Xeon(R) 2-core Gold 6133 2.50GHz CPU and 4GB memory, running Ubuntu 20.04 LTS operating system. Programming language C and OpenSSL 1.1.1 [1] library are used to implement LightPay, and the generator g is set to 2). In the experimental process, we compare the performance of LightPay with CryptoMaze [23], Multihop HTLC [17], and SplitPay [22], and the experimental configurations are as follows: CryptoMaze uses elliptic curve secp224r1 for encryption. Multihop HTLC uses a zero-knowledge proof library based on C implementation ZKBoo[2] and libgcrypt 1.8.4[3]. The number of ZKBoo rounds uses the default value of 136, and the witness length is set to 32 bytes. SplitPay uses elliptic curve secp224r1 for encryption and realizes homomorphic encryption based on libhcs library[4]. The following items are the metrics used in the experimentation:

- **Success rate.** Randomly selecting the sender and receiver from the network to construct the successful ratio of the payment path. Building a payment path requires ensuring

[1]https://www.openssl.org/
[2]https://github.com/Sobuno/ZKBoo/
[3]https://gnupg.org/software/libgcrypt/index.html
[4]https://github.com/tiehuis/libhcs

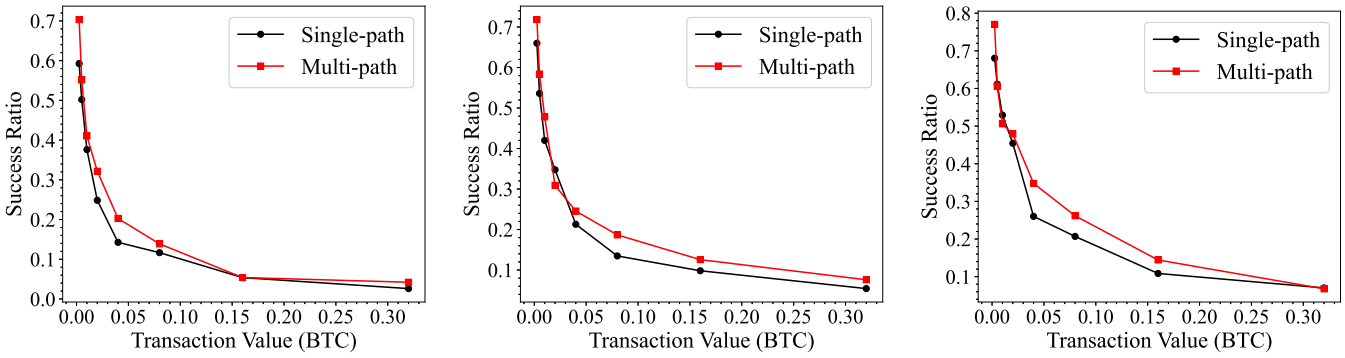


Fig. 6. Comparison of off-chain payment success rates on Lightning Network in March 2021, December 2021, and May 2022.

that each channel in the path has sufficient balance to support payment forwarding.

- **Number of off-chain contracts constructed.** The number of off-chain contracts that need to be constructed in the payment path. Every channel in every path needs to build an off-chain contract.
- **TTP (Time Taken for Payment).** The time required to find a suitable path from the payment network and build an off-chain contract, including releasing the contract. The small unit is s.
- **Communication overhead.** When conducting an off-chain payment, nodes need to forward the size of the message in KB.

B. Performance Evaluation in Lightning Network

We selected the snapshot data of the Lightning Network in March 2021[5] (11072 nodes), December 2021[6] (13023 nodes), and in May 2022[7] (17813 nodes) for the experiment, using the distributed routing protocol HushRelay [26] for off-chain payment path routing. In these networks, we conducted experiments on 5,000 payments with transaction amounts in the range of 0.0025-0.32BTC (the path length between the sender and the receiver is 4) and took the average value. We tested the payment success rate of different protocols and the number of off-chain contracts that need to be built. It is worth noting that in the success rate, we compared single-path and multi-path. Because LightPay, CryptoMaze, and SplitPay all support multi-path, their success rates are consistent. The representative of single-path is Multi-hop HTLC. Similarly, regarding the number of off-chain contracts under multi-path, we compare the split path (i.e., SplitPay) and the shared path (i.e., CryptoMaze and LightPay).

Success rate: As shown in Figure 6, the success rate of the multi-path payment protocol in the lightning network snapshot is generally higher than that of the single-path payment protocol. In Lightning Network[5], when trading 0.0025 BTC, the success rate can be increased by 11.08%. In Lightning Network[6], the success rate can be increased by 5.84%. Similarly, in Lightning Network[7], the success rate can be increased by 9%. This is because when only a single-path is used for payment, the amount that can be forwarded in a path depends on the minimum balance of each channel. After the introduction of multi-path, the selection range of the channel balance can be increased, thereby improving the payment success rate. Meanwhile, we can find that as the payment amount increases, the success rate becomes smaller because tiny payments dominate the lightning network, and most of the channel capacity is concentrated between 0.0025-0.04BTC. This experiment demonstrates that the multi-path scheme can improve the success rate of off-chain payment.

Number of construction contracts: As shown in Figure 7, the number of contracts in the shared path scheme is generally less than that of the split path scheme. In Lightning Network[5], 12.39% of contracts can be saved when transacting 0.32 BTC. In Lightning Network[6], the number of contracts built off-chain can be reduced by 7.36%. Similarly, in Lightning Network[7], 6.61% contract construction can be saved. Because in the shared path, there is no need to build multiple contracts for repeated sub-paths, and only one contract is needed to complete payment forwarding. The fewer contracts to build, the less the transaction fees and processing time is required to forward the payment. Only one fee for the same channel with different paths must be paid. At the same time, as the transaction amount increases, the difference in the number of off-chain contract constructions between the two protocols becomes wider. This is because the larger the transaction amount, the more paths may be split, and the shared path scheme has fewer contracts. This experiment proves that the shared path scheme (i.e., CryptoMaze and LightPay) can reduce the number of contracts that need to be constructed.

C. Performance Evaluation in Simulated Networks

In this section, we generate simulated networks with a network size of 300-38400 based on the Barábasi-Albert model [31], implemented based on the igraph library. We tested the protocol's TTP and communication overhead in a simulated network.

[5]https://www.dropbox.com/s/fkq7kh5xyu3l33t/LN_25_05_2021.json?dl=0

[6]https://git.tu-berlin.de/rohrer/discharged-pc-data/-/blob/master/snapshots/lngraph_2021_12_12__00_00.json.zst

[7]https://drive.google.com/file/d/1jPZHvm2OCovhKUHso6TkPvmJM89vQS5F/view

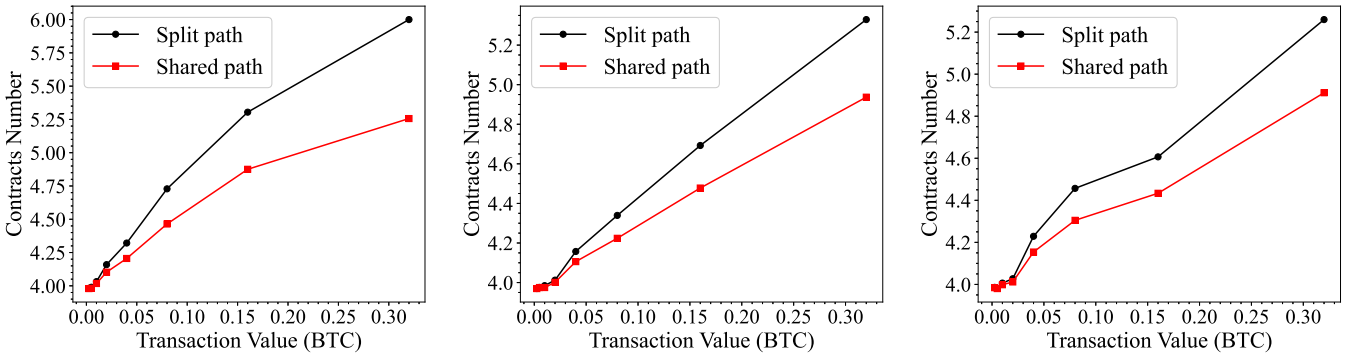


Fig. 7. Comparison of the number of construction contracts for off-chain payment on Lightning Network in March 2021, December 2021, and May 2022.

TTP: In Figure 8, LightPay's processing time did not exceed 0.8s in a simulated network with a size below 4800. When the simulated network size reaches 19200, the processing time does not exceed 5.7s. CryptoMaze's time overhead is close to LightPay. SplitPay is 1.6 times that of LightPay because it uses homomorphic encryption to increase the time overhead; yet, since Multihop HTLC uses zero-knowledge proofs, its time overhead is 42.4 times that of LightPay.

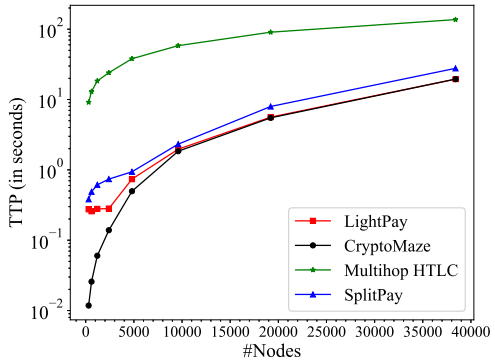


Fig. 8. Comparison of TTP in different protocols on the simulated network.

Communication overhead: As shown in Figure 9, the communication overhead of LightPay would not exceed 900 KB when the network size reaches 38400. The CryptoMaze is about 1.8 times that of the LightPay. This is because, in a path containing n channels, the message that LightPay needs to transmit in the pre-processing phase is 0 bytes, since the payment hidden value can be generated locally in the transaction sender U_s and does not need to be communicated with other nodes. In the contract forwarding phase, each payment channel needs to exchange 5 messages: IJ, val_{ij}, t_{ij}, S'_{ij}, and E_{ij}, and the total communication overhead is $120 \cdot n$ bytes. In the release phase, each payment channel must only exchange one signature information S_{ij}, occupying $64 \cdot n$ bytes. Finally, in the phase of updating the off-chain channel, each node no longer needs to exchange signatures, so the number of required transmission messages is 0. In these four phases, LightPay requires a total of $184 \cdot n$ bytes of messages to be transmitted. Similarly, we analyzed that the number of messages required to be transmitted by CryptoMaze in the pre-processing, contract forwarding, release, and update channel phases are 64, $136 \cdot n$, $64 \cdot n$, and $128 \cdot n$ bytes respectively, for a total of $64 + 328 \cdot n$ bytes. Consequently, when there are 10 channels in the path, LightPay's communication overhead is approximately reduced to 55% compared to CryptoMaze. Multihop HTLC is about 330 times that of LightPay, because it needs to process the zero-knowledge proof message of each channel, so the communication overhead is relatively large. Although LightPay is about 1.8 times faster than SplitPay, SplitPay's sub-path is vulnerable to wormhole attacks.

TABLE III
COMPARISON OF THE NUMBER OF MESSAGES THAT NEED TO BE DELIVERED IN DIFFERENT PHASES OF EACH PROTOCOL.

	MultiHop HTLC	SplitPay	CryptoMaze	LightPay
Pre-processing	1	1	1	0
Contract forwarding	$6 \cdot n$	$5 \cdot n$	$4 \cdot n$	$5 \cdot n$
Release	n	n	n	$n+1$
Receiver address	1	n	1	1
Signature	$2 \cdot n$	$2 \cdot n$	$2 \cdot n$	0

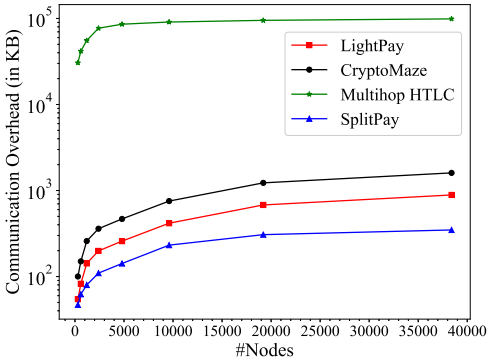


Fig. 9. Comparison of communication overhead in different protocols on the simulated network.

D. Efficiency Comparison

From theoretical analysis, the number of messages that need to be delivered in LightPay is shown in Table III.

In the pre-processing phase, Multihop HTLC, SplitPay, and CryptoMaze all need one message delivery; that is, the receiver calculates the hash value $R = \mathcal{H}(r)$ of the pre-image r, and the R is sent to the sender. The sender generates the r of LightPay, and there is no need to transfer the message in the pre-processing phase, despite the encrypted R being sent to the receiver in the forwarding contract phase. In the forwarding contract phase, Multihop HTLC needs to forward the node information of both sides of the channel, the hidden value of the payment, the payment amount, the expiration time, and a non-interactive zero-knowledge proof, so the protocol requires $6 \cdot n$ message delivery numbers.

For SplitPay, it is necessary to forward the sender information, receiver information, payment amount, hash value, and homomorphically encrypted ciphertext information, a total of $5 \cdot n$ messages. In CryptoMaze, $4 \cdot n$ pieces of information must be forwarded in the forwarding contract phase, including payment condition value, payment amount, timeout time, and encrypted information M_j. Compared with CryptoMaze, LightPay must deliver an additional adapter signature information, $5 \cdot n$ messages. Finally, in the release phase, all protocols require the receiver to transfer the secret value, so n messages are needed. What is noteworthy is that before this process in LightPay, the receiver needs to request the original secret value from the sender, so there is one more message. At the same time, LightPay passes the legal signature instead of the hidden value in the release phase, which can simplify the steps of exchanging signatures after confirming the payment.

It is worth noting that only SplitPay transmits the receiver's address information for every node in the payment path. At the same time, other methods only send the receiver's address in the last forwarding contract, since it needs intermediaries to divide the appropriate path according to the receiver's address, which may leak the receiver's privacy. In addition, LightPay does not need to deliver signatures, while the other protocols must provide $2 \cdot n$ signatures. This is because our protocol will send the adapter signature to the previous node during the release phase, and a legal signature can be generated after the last node completes the adaptation. Therefore, in the subsequent stage, only another node needs to complete one round of signatures.

In conclusion, experimental analysis shows that the proposed LightPay can improve the success rate of off-chain payment, reduce transaction fees, time overhead, and communication overhead, and provide better all-around performance than the existing protocols.

VI. Conclusions and Future Work

All transactions on the blockchain require the consensus of the entire network nodes, so the transaction throughput is minimal. In the off-chain payment channel, only the confirmation of both parties is necessary to complete the transaction consensus. To solve the problem of limited payment throughput in the blockchain, we propose LightPay - an atomic, private, and efficient off-chain multi-path payment protocol. LightPay realizes multi-path payment based on the adapter signature to improve the success rate of the payment and reduce the fee required for the payment process. To ensure unlinkability and privacy, LightPay constructs the conditional value of the off-chain contract according to different scenarios. Through experiments in Lightning Network and simulated networks, LightPay has presented better time overhead and communication overhead than the state-of-the-art schemes.

Due to the frequent one-way coin transfer, it is easy to cause the problem of channel exhaustion, affecting the success rate of transactions. In future directions, we will focus on the dynamic allocation of balances in the payment channel to further improve the success rate of off-chain payments. Additionally, we aim to investigate and develop efficient off-chain payment routing protocols.

Acknowledgment

This work was partially supported by the National Key Research and Development Program of China under Grant 2021YFA1000600, the National Natural Science Foundation of China under Grant 62072170, the Science and Technology Project of the Department of Communications of Hunan Provincial under Grant 202101, the Key Research and Development Program of Hunan Province under Grant 2022GK2015, the Hunan Provincial Natural Science Foundation of China under Grant 2021JJ30141, and the Open Research Fund of Hunan Provincial Key Laboratory of Network Investigational Technology under grant 2020WLZC001.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system."
[2] J. Fu, B. Cao, X. Wang, P. Zeng, W. Liang, and Y. Liu, "Bfs: A blockchain-based financing scheme for logistics company in supply chain finance," Connection Science, vol. 34, no. 1, pp. 1929–1955, 2022.
[3] D. Li, D. Han, Z. Zheng, T.-H. Weng, K.-C. Li, M. Li, and S. Cai, "Does short-and-distort scheme really exist? a bitcoin futures audit scheme through birch & bpnn approach," Computational Economics, pp. 1–23, 2023.
[4] W. Liang, Y. Yang, C. Yang, Y. Hu, S. Xie, K.-C. Li, and J. Cao, "Pdpchain: A consortium blockchain-based privacy protection scheme for personal data," IEEE Transactions on Reliability, pp. 1–13, 2022.
[5] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in Financial Cryptography and Data Security, A.-R. Sadeghi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 6–24.
[6] W. Chen, T. Zhang, Z. Chen, Z. Zheng, and Y. Lu, "Traveling the token world: A graph analysis of ethereum erc20 token ecosystem," in Proceedings of The Web Conference 2020, ser. WWW '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1411–1421.
[7] F. Tramèr, D. Boneh, and K. G. Paterson, "Remote side-channel attacks on anonymous transactions," in Proceedings of the 29th USENIX Conference on Security Symposium, ser. SEC'20. USA: USENIX Association, 2020, p. 2739–2756.
[8] W. Liang, Y. Fan, K.-C. Li, D. Zhang, and J.-L. Gaudiot, "Secure data storage and recovery in industrial blockchain network environments," IEEE Transactions on Industrial Informatics, vol. 16, no. 10, pp. 6543–6552, 2020.
[9] D. Han, Y. Zhu, D. Li, W. Liang, A. Souri, and K.-C. Li, "A blockchain-based auditable access control system for private data in service-centric iot environments," IEEE Transactions on Industrial Informatics, vol. 18, no. 5, pp. 3530–3540, 2022.
[10] A. Gangwal, H. R. Gangavalli, and A. Thirupathi, "A survey of layer-two blockchain protocols," Journal of Network and Computer Applications, vol. 209, p. 103539, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522001801
[11] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 801–815.

[12] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). Santa Clara, CA: USENIX Association, Feb. 2020, pp. 777–796.

[13] Y. Zhang and D. Yang, "Robustpay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks," IEEE/ACM Transactions on Networking, vol. 29, no. 4, pp. 1676–1686, 2021.

[14] Z. Li, W. Su, M. Xu, R. Yu, D. Niyato, and S. Xie, "Compact learning model for dynamic off-chain routing in blockchain-based iot," IEEE Journal on Selected Areas in Communications, vol. 40, no. 12, pp. 3615–3630, 2022.

[15] L. Aumayr, P. Moreno-Sanchez, A. Kate, and M. Maffei, "Blitz: Secure Multi-Hop payments without Two-Phase commits," in 30th USENIX Security Symposium (USENIX Security 21). USENIX Association, Aug. 2021, pp. 4043–4060.

[16] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 473–489.

[17] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 455–471.

[18] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in Proceedings 2019 Network and Distributed System Security Symposium. Internet Society, 2019.

[19] S. A. Krishnan Thyagarajan and G. Malavolta, "Lockable signatures for blockchains: Scriptless scripts for all signatures," in 2021 IEEE Symposium on Security and Privacy (SP), 2021, pp. 937–954.

[20] N. Glaeser, M. Maffei, G. Malavolta, P. Moreno-Sanchez, E. Tairi, and S. A. K. Thyagarajan, "Foundations of coin mixing services," ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1259–1273.

[21] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 106–123.

[22] L. Eckey, S. Faust, K. Hostáková, and S. Roos, "Splitting payments locally while routing interdimensionally," Cryptology ePrint Archive, Paper 2020/555, 2020, https://eprint.iacr.org/2020/555.

[23] S. Mazumdar and S. Ruj, "Cryptomaze: Privacy-preserving splitting of off-chain payments," IEEE Transactions on Dependable and Secure Computing, vol. 20, no. 2, pp. 1060–1073, 2023.

[24] N. Ying and T. W. Wu, "xlumi: Payment channel protocol and off-chain payment in blockchain contract systems," no. arXiv:2101.10621, Jan 2021, arXiv:2101.10621 [cs].

[25] E. Tairi, P. Moreno-Sanchez, and M. Maffei, "A2l: Anonymous atomic locks for scalability in payment channel hubs," in 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021, pp. 1834–1851.

[26] S. Mazumdar, S. Ruj, R. G. Singh, and A. Pal, "Hushrelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments," in 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2020, pp. 1–5.

[27] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_09-3_Roos_paper.pdf

[28] R. Canetti, "Universally composable security: a new paradigm for cryptographic protocols," in Proceedings 42nd IEEE Symposium on Foundations of Computer Science, 2001, pp. 136–145.

[29] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in Theory of Cryptography, S. P. Vadhan, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 61–85.

[30] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in Theory of Cryptography, A. Sahai, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 477–498.

[31] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," Reviews of modern physics, vol. 74, no. 1, p. 47, 2002.

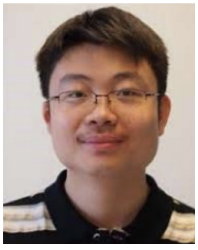
Yaqin Liu received the B.S. degree in computer science and technology from Hunan University of Science and Technology in 2021. She is pursuing a M.S. degree in the Colleges of Computer Science and Electronic Engineering at Hunan University. Her primary researches are focused on the security and privacy in Blockchain.

Wei Liang received a Ph.D. degree in computer science and technology from Hunan University, Changsha, China, in 2013. He was a Postdoctoral Scholar at Lehigh University during 2014–2016. He is currently a Professor and the Dean of the School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan, China. His research interests include blockchain security technology, network security protection, embedded systems, and hardware IP protection.

Kun Xie received a Ph.D. in computer application from Hunan University, China, in 2007. She worked as a postdoctoral fellow in the Department of Computing at Hong Kong Polytechnic University from 2007 to 2010. She was a visiting researcher in the electrical and computer engineering department at the State University of New York Stony Brook from 2012 to 2013. She is currently a professor at Hunan University, China. Her research interests include network monitoring, AI, and big data.

Songyou Xie is currently pursuing a PhD degree at Hunan University. He received a M.S. degree in computer science and technology from Hunan University of Science and Technology in 2019. His research interests include blockchain, smart contracts, and lightweight security authentication. protocols.

Kuanching Li is with Hunan University of Science and Technology. He has published papers in high-ranked journals and conferences and served as associate and guest editor for several scientific journals. His research interests include cloud and edge computing, big data, and blockchain.



Weizhi Meng is currently an Associate Professor in the Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Kongens Lyngby, Denmark. He obtained his PhD degree in Computer Science from the City University of Hong Kong (CityU), Hong Kong. His primary research interests are cyber security and intelligent technology in security, including intrusion detection, smartphone security, biometric authentication, HCI security, malware detection, blockchain in security, cyber-physical security, and IoT security.