Downloaded from orbit.dtu.dk on: May 07, 2024

DTU Library

Vertical Composition of Distributed Systems

Gondron, Sébastien Pierre Christophe

Publication date:
2021

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

Link back to DTU Orbit

Citation (APA):
Gondron, S. P. C. (2021). *Vertical Composition of Distributed Systems*. Technical University of Denmark.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Vertical Composition of Distributed Systems

Sébastien Pierre Christophe Gondron

Kongens Lyngby 2021

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

Ensuring security is a fundamental problem in modern distributed systems. For that purpose, system designers use security protocols, e.g., to transfer confidential data or to authenticate a user. It has become the norm to use simultaneously several of these security protocols. For example, to look up the balance on her bank account, a user Alice has to open a web browser session, and in order to access the application of her bank, she must first login through a webpage. All these components make use of security protocols to ensure that the data are exchanged out of reach from potential eavesdroppers and that the bank is authenticated to the client.

These security protocols are widely formally verified in the literature, but are only considered in isolation for the most part, and interactions between protocols that run simultaneously are ignored. This is because so called composed protocols can be rather difficult to verify since they are often too big for automated methods. To address these problems, we are interested in compositionality results of this form: given a number of protocols that are secure in isolation and that satisfy a number of simple syntactic conditions, then their composition is also secure.

Existing compositionality results focus on parallel composition, where protocols run independently on the same network, only sharing an infrastructure on fixed long-term keys, and on sequential composition where, for instance, one protocol can establish keys that a subsequent protocol then uses. However, vertical composition where protocols that keep a global state interact with each other, such as when a protocol uses another one to transmit messages in a confidential and authenticated manner, have been only scarcely studied.

The main objective of this thesis is to formalize a paradigm for vertical composition of stateful protocols. We can notably model trace-based properties, such as confidentiality or authentication; we are however limited when it comes to equivalence-based properties, such as privacy-type properties, which are of uttermost importance on the Internet. This is the reason why we extended in this thesis (α, β)-privacy. This approach allows to express privacy goals as reachability properties and opens the way for vertical compositionality results that also encompass privacy-type properties.

Our main contributions include:

- the formalization of vertical compositionality results for stateful protocols, based on previous results for parallel compositionality and on a sound abstraction for payloads,
- formalization of voting privacy properties, including receipt-freeness, in (α, β)-privacy,
- formalization of privacy as reachability properties, and
- a conservative extension of (α, β)-privacy with probabilities.

Summary (Danish)

Titel: Vertikal Sammensætning af Distribuerede Systemer

At sikre sikkerhed er et grundlæggende problem i moderne distribuerede systemer. Med henblik herpå bruger systemarkitekter sikkerhedsprotokoller, e.g., for at sende konfidentielle data eller for at autentificere brugere. Det er blevet almindeligt, at mange sikkerhedsprotokoller kører på samme tidspunkt. Når Alice vil tjekke hendes konto, åbner hun en webbrowser session og derefter logger hun på en webside for at få adgang til bankapplikationen. Alle komponenter bruger sikkerhedsprotokoller for at sikre, at dataene, der bliver udvekslet er beskyttet mod aflytning, og som autentificerer banken til klienten.

Sikkerhedsprotokoller er i stor udstrækning formelt verificerede i litteraturen. Generelt påtænkes de i isolation, og interaktioner mellem protokoller, der kører på samme tidspunkt, bliver set bort fra. Det skyldes, at såkaldte sammensatte protokoller kan være ret svære at verificere, fordi de ofte er for store til automatiserede metoder. For at løse disse problemer bør vi koncentrere os om sammensætningsresultater, som har følgende form: hvis et givet antal protokoller hver især er sikker i isolation, og som opfylder et antal enkle syntaktiske forudsætninger, så er deres sammensætning også sikker.

Eksisterende sammensætningsresultater fokuserer på sideløbende sammensætning, hvor protokoller selvstændigt kører på samme netværk og deler langsigtede nøgler, eller på sekventiel sammensætning, hvor en protokol for eksempel kan generere nøgler, som en efterfølgende protokol kan bruge. Imidlertid blev vertikale sammensætninger, hvor tilstandsfulde protokoller påvirker hinanden, som for eksempel, når en protokol bruger en anden for at sende beskeder på en konfidentiel og autentisk måde, knap udforsket.

Hovedformålet med denne afhandling er at formalisere vertikale sammensætningsresultater til tilstandsfulde protokoller. Vi kan især formalisere *trace-based*-egenskaber såsom konfidentialitet og autenticitet. Desværre er vi begrænset, når det kommer til *equivalence-based*-egenskaber såsom privatlivsegenskaber, der har særlig vægt på Internettet. Derfor udvidede vi (α, β)-privacy i denne afhandling. Den tilgang muliggør at udtrykke privatlivsmål som *reachability*-egenskaber og baner vej for vertikale sammensætningsresultater, som gælder for privatlivsmål.

Vores vigtigste bidrag inkluderer:

- formalisering af vertikale sammensætningsresultater for tilstandsfulde protokoller, der baseres på tidligere sammensætningsresultater for sideløbende protokoller op på velfunderet abstraktion til applikationsdata,
- formalisering i (α, β)-privacy af stemmehemmelighedsegenskaber såsom kvitteringsfri,
- formalisering af privatlivsegenskaber som *reachability* egenskaber, og
- en vedligeholdende udvidelse af (α, β)-privacy med sandsynligheder.

Preface

This thesis was prepared at Department of Applied Mathematics and Computer Science (DTU Compute) in fulfillment of the requirements for acquiring a PhD degree in Computer Science.

The research has been carried out under the supervision of Sebastian Mödersheim and Alberto Lafuente in the period from July 2018 to June 2021.

This research was supported by the Sapere-Aude project "Composed: Secure Composition of Distributed Systems", grant 4184-00334B of the Danish Council for Independent Research.

A substantial part of the work presented in this thesis is based on extensions to joint work with Sebastian Mödersheim and Luca Viganò, namely the following two published papers and a submitted paper: *Formalizing and Proving Privacy Properties of Voting Protocols using Alpha-Beta Privacy* [GM19], *Vertical Composition and Sound Payload Abstraction for Stateful Protocols* [GM21], and *Privacy as Reachability* [GMV21]. Chapter 2 is mainly based on [GM21] whereas Chapter 3 and Chapter 4 are based on [GM19], and Chapter 5 is based on [GMV21].

Lyngby, 30-June-2021

Sébastien Pierre Christophe Gondron

Acknowledgements

At hyggjandi sinni
skylit maðr hræsinn vera,
heldr gætinn at geði;
þá er horskr ok þögull
kemr heimisgarða til,
sjaldan verðr víti vörum,
því at óbrigðra vin
fær maðr aldregi
en mannvit mikit.

Eddukvæði, Hávamál

I am especially grateful to my supervisors Sebastian Alexander Mödersheim and Alberto Lluch Lafuente, for their encouragement and excellent guidance throughout this thesis. This has been the opportunity to look deeper into formal methods and verification of security protocols, areas that I grew attached to. I would also like to thank Luca Viganò for being my supervisor during my external stay at King's College London, even though this has taken place remotely due to the context.

I would also like to thank Jørgen Villadsen, Joshua D. Guttman and Christoph Sprenger for accepting to be part of my assessment committee. I am also grateful to my colleagues in the formal methods section at DTU and all the people I have shared lunches and discussions at the office or conferences. These discussions, always pleasant, have been fruitful in the development of my research.

On a personal note, I want to thank all my friends, be they in Copenhagen or elsewhere. In such difficult times, they have often been the reason I could continue this work. I would also like to thank the people at Peders that made the most advertised concept of danish *hygge* concrete and that, in a context where universities were closed, heard me rambling on the doubts and hopes around my research. Most importantly, I want to thank my family for always believing in me. I am sure this work will make them proud, and that a copy will find its way to a bookcase at home.

Notations

App	The application protocol	9.
Ch	The channel protocol	9.
$f_{(\ldots)}$	Message formats	9.
$\mathsf{Ch}^{\star}$	The protocol idealization, or the protocol interface	10.
$\mathsf{Ch}^{\sharp}$	The protocol abstraction	11.
Σ	The set of function symbols.	12.
$\mathcal{V}$	The set of variables.	12.
x, x_i	Variables.	12.
t, t_i	Terms.	12.
Σ^n	The symbols of Σ of arity n.	12.
$\mathcal{C}$	The set of constants, i.e., Σ^0.	12.
$\mathcal{T}(\Sigma, \mathcal{V})$	The set of terms over Σ and $\mathcal{V}$.	12.
$fv(t)$	The set of variables of a term t.	12.
$\sqsubseteq$	The subterm relation.	12.
σ,θ	Substitutions.	12.
$dom(\sigma)$	The substitution domain of the substitution σ	12.
$img(\sigma)$	The substitution image of the substitution σ	12.
$\mathcal{I}$	Interpretations.	12.
$\Sigma_{\boldsymbol{pub}}$	The public functions.	12.
Ana	The analysis function.	12.
$M \vdash t$	Denotes that the intruder can derive t given the messages in M.	13.
$\mathcal{P}, \mathcal{P}_1, \mathcal{P}_2$	Arbitrary protocols.	14.
R_i	Transaction rules.	14.
$\mathcal{S}$	Transaction strands with sets.	14.
$\mathsf{send}(t), \mathsf{receive}(t)$	Message transmission constraints.	14.
$t \doteq t'$	Equality constraints on messages.	14.
$\forall \bar{x}.\ t \not\doteq t'$	Inequality constraints on messages.	14.
$t \dot{\in} t'$	Positive set-membership constraints.	14.
$\forall \bar{x}.\ t \dot{\notin} t'$	Negative set-membership constraints.	14.
$\mathsf{insert}(t, t')$	Insertion of element t into database t'.	14.
$\mathsf{delete}(t, t')$	Deletion of element t into database t'.	14.
$t \rightarrow t'$	Syntactic sugar for $\mathsf{insert}(t, t')$.	14.
$t \leftarrow t'$	Syntactic sugar for $t \dot{\in} t'.\mathsf{delete}(t, t')$.	14.
$\xleftarrow{t}$	Syntactic sugar for $\mathsf{receive}(t)$.	14.
$\xrightarrow{t}$	Syntactic sugar for $\mathsf{send}(t)$.	14.
$trms(\mathcal{A})$	The set of terms occurring in the constraint $\mathcal{A}$.	15.
$\mathcal{A}$	Symbolic constraints.	15.

$setops(\mathcal{A})$	The set of set operations of the constraint $\mathcal{A}$. ....15.
$[\![M, D; \mathcal{A}]\!]\ \mathcal{I}$	Constraint semantics for stateful constraints. Denotes that $\mathcal{I}$ is a model of $\mathcal{A}$ given the initial intruder knowledge M and the initial database mapping D. ....15.
$\mathcal{I} \models \mathcal{A}$	Constraint semantics. Equivalent to $[\![\emptyset, \emptyset; \mathcal{A}]\!]\ \mathcal{I}$. . 16.
$fv(\mathcal{A})$	The free variables of th constraint $\mathcal{A}$. .....16.
$\Rightarrow$	The state transition relation. .....16.
$dual(\mathcal{S})$	The dual of the transaction strand $\mathcal{S}$. ..... 16.
$\mathfrak{s}$	Constraint steps. .....16.
$\Rightarrow^\star$	The transitive reflexive closure of $\Rightarrow$. .....17.
$\mathsf{attack}_{\mathcal{P}}$	The attack constant unique to the protocol $\mathcal{P}$. .. 17.
$\mathsf{outbox}(A, B)$,$\mathsf{inbox}(A, B)$	Message transmission sets between the principals A and B. .....17.
$\mathfrak{p}$	Abstract type, or Payload type. .....18.
$\mathfrak{T}_{\mathfrak{p}}$	The set of concrete payload types. ..... 19.
$\mathfrak{T}_a$	The set of atomic types. ..... 19.
Γ	The typing function. ..... 19.
τ	Arbitrary types. ..... 19.
$SMP(M)$	The sub-message patterns of the messages in the set M. .....20.
$\star$	The star label. .....21.
$\mathcal{P}^\star$	The idealization of the protocol $\mathcal{P}$. ..... 22.
$\mathcal{P}_1 \parallel \mathcal{P}_2$	The parallel composition of protocols $\mathcal{P}_1$ and $\mathcal{P}_2$. 22.
$\{t \mid \emptyset \vdash t\}$	The basic public terms. ..... 23.
Sec	The set of secret messages. .....23.
$GSMP(M)$	The ground sub-message patterns of the messages in M. .....23.
$\mathcal{DY}(M)$	The Dolev-Yao closure of the set of messages M. 23.
$traces(\mathcal{P})$	The traces of the protocol $\mathcal{P}$. .....24.
$\frac{\mathsf{App}}{\mathsf{Ch}}$	The vertical composition of protocols App and Ch. 24.
$\mathfrak{G}$	The infinite set of constants of type $\mathfrak{a}$. .....32.
$\mathfrak{a}$	The abstract type. .....32.
$\ulcorner\urcorner$	A block. ..... 36.
$\mathcal{A}(n)$	The n-th block of the constraint $\mathcal{A}$. ..... 36.
$\mathcal{A}(1, n)$	The n first blocks of the constraint $\mathcal{A}$. .....36.
tr, tr'	Traces. .....36.
$\mathsf{Ch}^{\mathsf{App}}$	The instantiated channel. ..... 37.
g	The abstraction function. ..... 39.
$GSMP_{\mathsf{App}}$	The ground sub-message patterns of the application protocol without the public terms. ..... 39.
status	The meta-function that returns the status of a constant g. .....39.
$\bowtie$	The compatibility relation. .....44.

$pos(t)$	The set of positions that exists in the term t. ....44.
Σ_f	The set of uninterpreted function symbols. ..... 77.
Σ_i	The set of interpreted function symbols. .....77.
Σ_r	The set of relation symbols. .....77.
$\mathcal{T}_{\Sigma_f}$	The set of ground terms that can be built using symbols in Σ_f. .....77.
$\approx$	A congruence relation on $\mathcal{T}_{\Sigma_f}$. .....77.
ϕ, ψ	Herbrand formulae. .....78.
Σ_{op}	The set of operators available to the intruder. ...78.
F, F_1, F_2	Arbitrary frames. .....78.
$concr$	The concrete knowledge frame. .....78.
$struct$	The structural knowledge frame. .....78.
gen	A unary relation symbol. .....78.
r, s	Arbitrary recipes. .....79.
Σ_0	The payload alphabet. .....80.
$MsgAna$	Message-Analysis problem .....81.
π, ρ	Permutations .....86.
θ_0	The truth. .....87.
θ_I	The intruder hypothesis. .....88.
Dan	The coerced voter. .....92.
$struct_{Dan}, concr_{Dan}$	The story of Dan. .....93.
ϕ_{lie}	The Lying Axiom. .....94.
γ	The truth formula. .....103.
δ	The sequence of conditional updates on the cells. 103.
η	The probability decision tree for the random variables.... .....103.
$\mathcal{P}_l$	Left processes. .....104.
$\mathcal{P}_r$	Right processes. .....104.
$(\alpha, \beta, \gamma, \delta, \eta)$	A state. .....107.
$(P, \phi, struct)$	A possibility. .....108.
$(\mathcal{S}, \mathcal{P})$	A configuration. .....108.
$[\mathcal{I}]$	An interpretation class. .....121.
$\mathcal{P}_{abs,\eta}([\mathcal{I}])$	The absolute probability of the interpretation class $[\mathcal{I}]$. .....121.
$\mathcal{P}_{\eta}([\mathcal{I}])$	The normalized probability of the interpretation class $[\mathcal{I}]$. .....121.

Contents

Chapter 1

Introduction

The work presented in this thesis is based on three major publications, each of which has a dedicated chapter in this thesis:

- *Formalizing and Proving Privacy Properties of Voting Protocols using Alpha-Beta Privacy* [GM19] by Gondron and Mödersheim, published at the 24th European Symposium on Research in Computer Security (ESORICS) in 2019.
- *Vertical Composition and Sound Payload Abstraction for Stateful Protocols* [GM21] by Gondron and Mödersheim, published at the 34th IEEE Computer Security Foundations (CSF) Symposium in 2021.
- *Privacy as Reachability* [GMV21] by Gondron, Mödersheim and Viganò, submitted to the 35th IEEE Computer Security Foundations (CSF) Symposium (first cycle).

Composition of Security Protocols

Communicating on networks, like the Internet, requires to run a wide variety of security protocols since such networks cannot be assumed to be safe. For instance, if Alice wants to visit the website of her bank to request the balance on her main account, she has to open a web browser, enter her login and her password on the login page of her bank website, and then only she can access the banking application. During this brief operation from everyday life, many security protocols were involved. The banking application is itself such a protocol that uses for security a login protocol and a TLS session. The specification and the analysis of the security properties of these protocols is a full-fledged research area.

One approach in this research area is to model cryptographic messages as formal terms subject to some equational theories representing attacker capabilities:

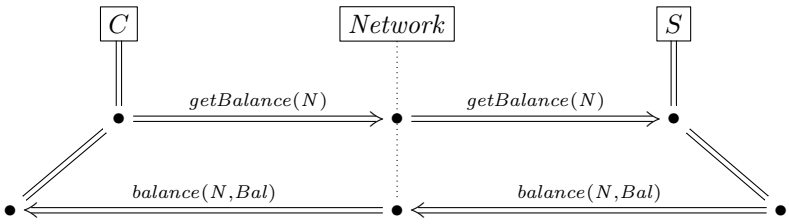

Figure 1.1: Banking protocol

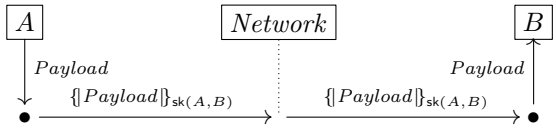

Figure 1.2: Channel protocol

the *symbolic model* originally proposed by Dolev and Yao [DY83]. The model aims at finding *logical* flaws, e.g., man-in-the-middle [Low95] or reflection attacks [BCM13] Over the years, a number of models have been adapted to this idea, e.g., the *applied-pi calculus* [ABF18], or *multiset rewriting* [Mit02]. This has resulted in tools such as ProVerif [Bla01], Avispa [Arm+05], Maude-NPA [EMM07], CPSA [Gut11], Tamarin [Mei+13], DeepSec [CKR18], or PSPSP [Hes+21].

However, despite their actual deployment within real environments, security protocols have been mostly and comprehensively formally verified in isolation. Only few work has been undertaken about their composition, i.e., when protocols run simultaneously or interact with each other. Indeed, being proven secure in isolation is not a guarantee that their composition is also secure. In the literature, three families of compositions are classically identified: parallel, sequential and vertical.

These compositionality results have a modular form: "given a suite of protocols that satisfy certain sufficient conditions and that are secure in isolation, then their composition is a secure system as well". These conditions should be easy to check statically. Most existing works on protocol composition have concentrated on parallel composition, i.e., when protocols run independently on the same network only sharing an infrastructure of fixed long-term keys [GT00; Gut09; CD09] with each other. [HMB18] is the first parallel compositionality result to support a wide variety of interactions between protocols: it allows for stateful protocols that maintain databases, shares them between protocols, and for the

declassification of long-term shared secrets.

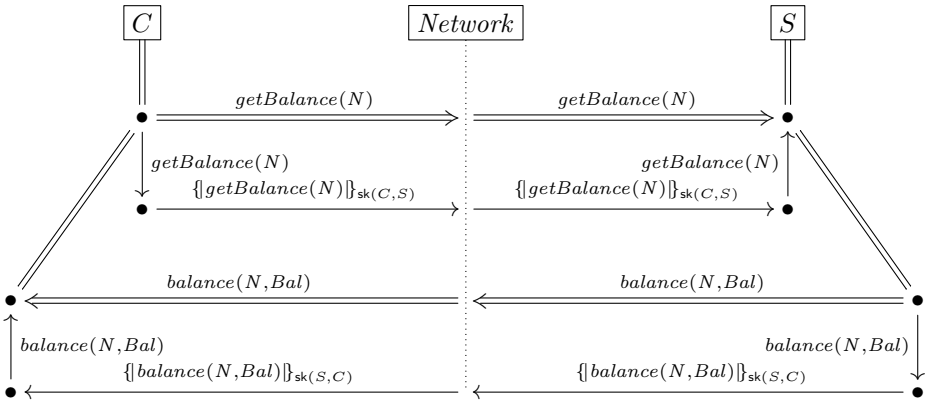

Figure 1.3: Vertical composition of a banking and channel protocols

In this thesis, we are particularly concerned with vertical composition. This is of great importance on the Internet, where protocols are often composed in a layered manner, i.e., one protocol transport "payload" for another. One can think of the numerous protocols that establish secure channels, such as TLS, and the protocols actually using these channels to run in a "secure" manner. For instance, the banking protocol from Figure 1.1 and the channel protocol from Figure 1.2 can be composed into the protocol in Figure 1.3, where the channel transports the messages from the banking application by replacing its "payload" message.

Mödersheim and Viganò [MV14] gave a first result on vertical composition limited to one transmitted message of the application protocol over the established channel, but the sufficient condition is semantical, i.e., cannot be checked statically. Mödersheim and Groß [GM11] introduce compositionality results for a larger class of protocol, including the stacking of several protocols on top of each other, introduced for instance by the vast deployment of VPNs and secure channels established by browsers but the interactions were limited.

Establishing a Vertical Compositionality Result for Stateful Protocol
In Chapter 2, we establish a vertical compositionality result for stateful protocol. This work is based on [GM21].

The first contribution of this chapter is the formalization of a general paradigm

for vertical composition of stateful protocols. We build on the results for parallel compositionality for stateful protocols of [Hes19]. The results of [Hes19] allow for stateful protocols that maintain databases (such as a keyserver), and the databases may even be shared between these protocols. This includes the possibility to declassify long-term secrets, e.g., to verify that a protocol is even secure if the intruder learns old private keys. We express a vertical composition as the parallel composition of two types of protocols, channel Ch and application App protocols, that are connected only by two families of sets outbox and inbox.

Let us take a look back at our examples in Figures 1.1 and 1.2, and consider them in our paradigm in Figure 1.4. When the client wants to send a request, $getBalance(N)$, to the bank server, the client inserts the message into the set $\mathsf{outbox}(C, S)$. The channel protocol retrieves the request from the set, transports it over the network, and on S's side, inserts it into the set $\mathsf{inbox}(C, S)$. The server can now retrieve the request from the inbox set. To send the response, the client and the server exchange the roles.

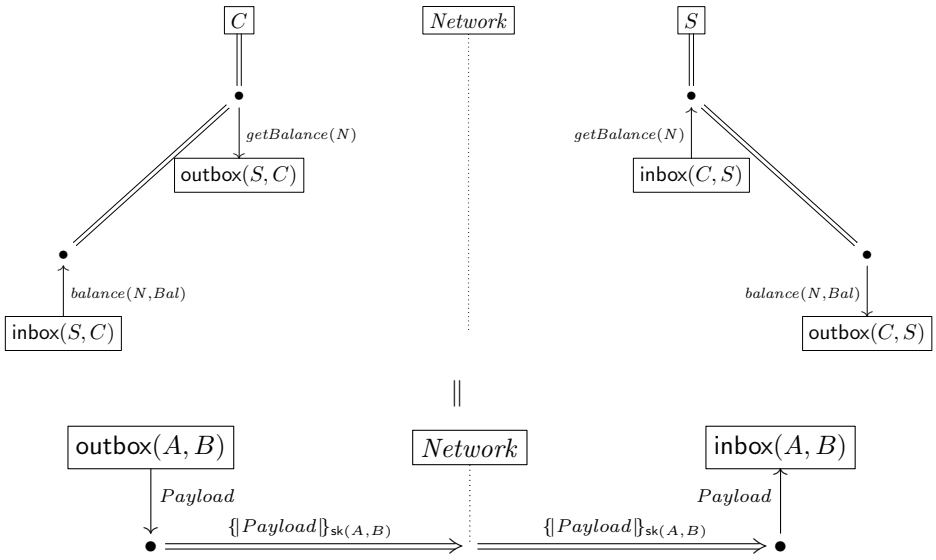

Figure 1.4: Vertical composition in our framework

We develop and proved a number of extensions of the definition and theorems of [Hes19]. In particular, we extend the typing result to take into account that messages from App can be handled by Ch when moved between sets and transmitted over the network as part of channel message transmissions by introducing a "payload type". These extensions were sufficient to allow for the verification of

the application protocol with respect to an idealization of the channel protocol $\mathsf{Ch}^\star$. A channel idealization models the security guarantees that a channel offers to an application. In Figure 1.5, we represent that an application relies on the security provided by $\mathsf{Ch}^\star$ to move messages from outbox to inbox sets.

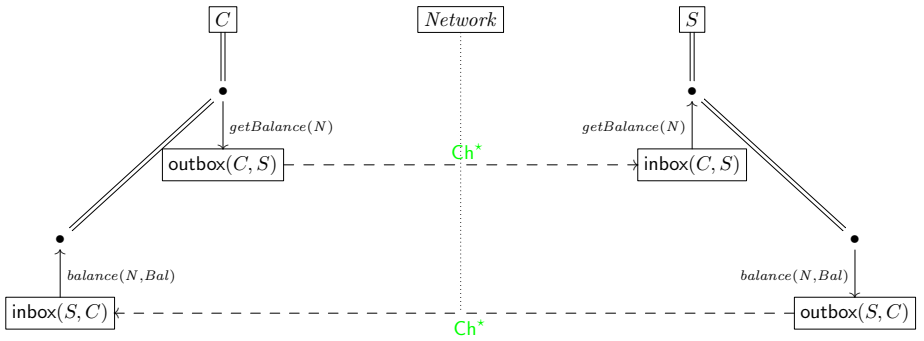

Figure 1.5: Login protocol with respect to the idealization of a channel protocol

However, in contrast with the application, our extensions to the results of [Hes19] are not sufficient for verifying channel protocols independent of an application, since outbox sets still contain all possible concrete terms from the application. The key contribution of this chapter is the development of a sound abstraction for payload variables, so the channel can be verified independent of a concrete application protocol. This abstraction substitutes, in a way, payload variables by constants that can either be fresh or previously used, and either known or unknown to the intruder. Many works that verify channel protocols such as TLS [Cre+17] model payloads as freshly generated nonces that are transmitted over the channel, and we prove in this chapter that this is a sound abstraction if done right.

Privacy

Beyond the classical security properties that we have considered in the first part of this thesis, such as confidentiality or authentication, a problem of large systems is to maintain privacy properties. When composing components, it is much harder to keep the overview of where an observer may be able to make links—and which links are even acceptable, i.e., considered pubic information. While we do not really develop compositionality results for privacy goals, we

follow here an approach to make feasible the specification and analysis of privacy goals in large systems where it is hard to keep track of all details. Formulating privacy "positively" by what information has deliberately been released eases the work for the system designer and reduces the risk of forgetting about links that an attacker can make.

Privacy-type properties of security and voting protocols are traditionally specified as *trace equivalence of two processes* in some process calculus, such as the Applied-π calculus [ABF18; BAF08; CRZ07; DRS08]. While such approaches have uncovered vulnerabilities in a number of protocols, they rely on asking whether the intruder can distinguish two variants of a process. It is a rather technical way to encode the privacy goals of a protocol.

To fill the gap between intuitive ideas of the privacy goals and the mathematical notions used to formalize and reason about them, (α, β)-*privacy* has been proposed in [MV19]. It is a declarative approach based on specifying two formulae α and β in first-order logic with Herbrand universes. α formalizes the *payload* i.e., the "non-technical" information that we intentionally release to the intruder, and β describes the "technical" information that he has, i.e., his "actual knowledge": what (names, keys, etc.) he initially knows, which actual cryptographic messages he observed and what he infers from them. He may be unable to decrypt a message, but know anyway that it has a certain format and contains certain (protected) information, e.g., a vote.

Formalizing Voting Privacy Properties in Alpha-Beta Privacy In Chapter 4, we formalize voting privacy properties, including receipt-freeness, in (α, β)-privacy. This work is based on [GM19].

This chapter gives a model-theoretic way to formalize and reason about voting privacy and receipt-freeness. As explained, we build on the framework of (α, β)-privacy [MV19], that defines privacy as a state-based reachability property.

Our contribution is a general methodology to modeling voting privacy and receipt-freeness with (α, β)-privacy that can be applied for a variety of protocols. This in particular includes a proof methodology that allows for simple model-theoretic arguments, suitable for manual proofs and proof assistants like Isabelle and Coq. We illustrated this practically at hand of the FOO'92 protocol as an example.

Formalizing Privacy Properties as Reachability Properties In the first part of Chapter 5, we express privacy as a reachability problem in a state tran-

sition system. This work is based on [GMV21].

The main contribution of this part of Chapter 5 is to lift (α, β)-privacy from a static approach to a dynamic one. We define a transaction-process formalism for distributed systems; the semantics of these transactions is specified by a transition system. Our formalism includes privacy variables that can be non-deterministically chosen from finite domains, can work also with long-term mutable states, and allows one to specify the consciously released information. Every state of our defined transition system is an (α, β)-privacy problem, i.e., a pure reachability problem.

Extending Alpha-Beta Privacy to a Probabilistic Approach Finally, we extend both the static and dynamic approach of (α, β)-privacy by probabilities in the second part of Chapter 5. This work is based on [GMV21].

Many approaches (e.g., quantitative information flow [Alv+20; CHM01; DHW04; Gru08; LMT10; Low02; Smi09], differential privacy [Dwo08; YCW17], etc.) reason about privacy by considering quantitative aspects and probabilities. Trace equivalence approaches are instead purely qualitative and possibilistic, and so is (α, β)-privacy; this is appropriate for many scenarios, but we give examples where probability distributions play a crucial role (i.e., in a purely possibilistic setting, there is no attack, but with probabilities there is).

The contribution of the second part of Chapter 5 is a conservative extension of (α, β)-privacy by probabilistic variables. As for non-deterministic variables (used when probabilities are irrelevant or when the intruder does not know the distribution), probabilistic variables can be sampled from a finite domain with a probability distribution, which may depend on probabilistic variables that were chosen earlier. As proof-of-concept, we consider some examples, e.g., the well-known problem of vote copying (e.g., in Helios, where a dishonest voter can copy an honest voter's vote). Finally, we also study in details how (α, β)-privacy relates to trace-equivalence and information flow approaches.

Chapter 2

Vertical Composition and Sound Payload Abstraction for Stateful Protocols

With *vertical composition*, we mean that a high-level protocol called *application*, or App for short, uses for message transport a low-level protocol called *channel*, or Ch for short. For instance, a banking application may be run over a channel established by TLS. For concreteness, let us consider a simple running example of a login protocol over a unilaterally authenticated channel as shown in semi-formal notation in Figure 2.1. Here $[C]P$ represents a client C that is not authenticated but acting under an alias (pseudonym) P, which is simply a public key, and only C knows the corresponding private key $\mathsf{inv}(P)$. Clients can have any number of aliases, and thus choose in every session to either work under a new identity or use the same alias, and thereby link the sessions. The setup of the channel has the client generate a new session key K, sign it with $\mathsf{inv}(P)$ and encrypt it for a server S. The functions $f_{(\ldots)}$ represent message formats like XML that structure data and distinguish different kinds of messages. This gives us a secure key between P and S: the server S is authenticated w.r.t. its real name while the client is only authenticated w.r.t. alias P—this is somewhat similar to the typical deployment of TLS where P would correspond to the unauthenticated Diffie-Hellman half-key of the client. We can transmit messages on the channel by encrypting with the established key, and the login protocol now uses this channel for authenticating the client. For simplicity, the client is computing a MAC on a challenge N from the server with a shared secret. This models the second factor in the Danish NemID [Net21] service where each user has a personal key-card to look up the response for a given challenge N. The first factor, a password, we just omit for simplicity.

Most existing works on protocol composition have concentrated on *parallel* composition, i.e., when protocols run independently on the same network only sharing an infrastructure of fixed long-term keys [HT96; GT00; And+08; Gut09; CD09; CC10; Che+13; ACD15; Alm+15]. In contrast, we want to compose here components that *interact* with each other, namely an application App that hands messages to a channel Ch for secure transmission. [GM11; CCW17] allow

Channel protocol Ch:
Setup:

$$\begin{aligned} [C]P\colon &\quad \mathsf{new}\ K \\ [C]P \to S\colon &\quad \mathsf{crypt}(\mathsf{pk}(S), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,S,K))) \end{aligned}$$

Transport:

$$\begin{aligned} \text{For } [C]P \xrightarrow{\mathsf{Ch}} S\colon &\quad X,\ \text{transmit } \mathsf{scrypt}(K, f_{pseudo}(P,S,X)) \\ \text{For } S \xrightarrow{\mathsf{Ch}} [C]P\colon &\quad X,\ \text{transmit } \mathsf{scrypt}(K, f_{pseudo}(S,P,X)) \end{aligned}$$

Login protocol App:

$$\begin{aligned} S\colon &\quad \mathsf{new}N \\ S \xrightarrow{\mathsf{Ch}} [C]P\colon &\quad f_1(N,S) \\ [C]P \xrightarrow{\mathsf{Ch}} S\colon &\quad f_2(\mathsf{mac}(\mathsf{secret}(C,S),N)) \end{aligned}$$

Figure 2.1: Running Example

for interaction between the protocols that are being composed, albeit specialized to a particular form of interaction. [HMB18] is the first parallel compositionality result to support arbitrary interactions between protocols: it allows for stateful protocols that maintain databases, shares them between protocols, and for the declassification of long-term secrets.

As a first contribution, we build upon these results a general paradigm for vertical composition: we use such databases to connect channel Ch and application App protocols. For instance, when the application wants to send a message from A to B, then it puts it into the shared set $\mathsf{outbox}(A,B)$ where the channel protocol fetches it, encrypts and transmits it, and puts it in the corresponding inbox of B where the application can pick it up.

As a second contribution, we extend the typing result from [Hes19] to take into account that messages from App can be manipulated by Ch. Thus, in our paradigm, Ch and App are arbitrary protocols from a large class of protocols that synchronize via shared sets inbox and outbox and fulfills a number of simple syntactic conditions.

Compared to refinement approaches that "compose" a particular application with a specific channel, our vertical compositionality result is much more general: from the definition of a channel protocol Ch, we extract an idealized behavior $\mathsf{Ch}^\star$, the protocol *interface*, that hides how the channel is actually implemented and offers only a high-level interface for the application, e.g., guaranteeing con-

fidential and/or authentic transmission of messages. The application can be then verified against this interface, and so we can at any time replace Ch by any other channel Ch' that implements the same interface $\mathsf{Ch}^\star$ without verifying the application again. However, the third and core contribution of this chapter is a solution for the converse question, namely how to also verify the channel independently of the application, so that the channel Ch can be used with *any* application App that relies only on the properties guaranteed by the interface $\mathsf{Ch}^\star$.

If we look for instance at the formal verification of TLS in [Cre+17; BBK17], all the payload messages that are transmitted over the channel (corresponding to X in Figure 2.1) are just modeled as fresh nonces. One could say this paper verifies that TLS is correct if the application sends only fresh nonces. In reality, the messages may neither be fresh nor unknown to the intruder, and in fact they may be composed terms that could interfere with the channel context they are embedded in. For a well-designed channel protocol, this is unlikely to cause trouble, but wouldn't it be nice to formally prove that?

The core contribution of this chapter is a general solution for this problem: we develop an abstraction of the payload messages and prove its soundness. The abstraction is indeed similar to the fresh-nonce idea, but taking into account that they represent structured messages that may be (partially) known to the intruder, and that applications may transmit the same message multiple times over a channel. This gives rise to a translation from the concrete Ch to an abstract version $\mathsf{Ch}^\sharp$ that uses nonces as payloads—a concept that all standard protocol verification tools support. The soundness means that it is sufficient to verify $\mathsf{Ch}^\sharp$ in order to establish that Ch is secure in that it fulfills its interface $\mathsf{Ch}^\star$, and is securely composable with *any* application App that expects interface $\mathsf{Ch}^\star$ (and given that some syntactic conditions between App and Ch are met, like no interference between their message formats). In the example, after verification, we know that not only this composition is secure, but that Ch is secure for any application that requires a unilaterally authenticated channel, and App can securely run on any channel providing the same interface. Thus, the composition may not only reduce the complexity of verification, breaking it into smaller problems, but also make the verification result more general.

Organization: in §2.1, we introduce the framework to model stateful protocols. In §2.2, we describe our paradigm for vertical composition, and we extend a typing result to support an abstract payload type. In §2.3, we prove that our abstraction of payloads is sound, and that our vertical composition result can be used with a wide variety of channel and application protocols. §2.4 gives the proof of the main theorems. §2.5 gives the full extension of the typing results. §2.6 show how to apply the vertical composability definition to our running example. §2.7 gives a battery of examples to illustrate the scope of our results.

In §2.8, we show how our results can be used to study channel bindings. Finally, we relate our work to others and conclude in §2.9.

2.1 Preliminaries

Most of the content of this section is adapted from [Hes19].

2.1.1 Terms and substitutions

We consider a countable signature Σ and a countable set $\mathcal{V}$ of variable symbols disjoint from Σ. We do not fix a particular set of cryptographic operators, and our theory is parametrized over an arbitrary Σ. A term is either a variable $x \in \mathcal{V}$ or a composed term of the form $f(t_1, \ldots, t_n)$ where $f \in \Sigma^n$, the t_i are terms, and Σ^n denotes the symbols in Σ of arity n. We define the set of constants $\mathcal{C}$ as Σ^0. We denote the set of terms over Σ and $\mathcal{V}$ as $\mathcal{T}(\Sigma, \mathcal{V})$. We denote the set of variables of a term t as $fv(t)$, and if $fv(t) = \emptyset$ then t is *ground*. We extend these notions to sets of terms. We denote the *subterm* relation by $\sqsubseteq$.

We define substitutions as functions from variables to terms. $dom(\sigma) \equiv \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is the domain of a substitution σ, i.e., the set of variables that are not mapped to themselves by σ. We then define the substitution image, $img(\sigma)$, as the image of $dom(\sigma)$ under σ: $img(\sigma) \equiv \sigma(dom(\sigma))$, and we say σ is ground if its image is ground. An *interpretation* is defined as a substitution that assigns a ground term to every variable: $\mathcal{I}$ is an interpretation iff $dom(\mathcal{I}) = \mathcal{V}$ and $img(\mathcal{I})$ is ground. Substitutions are extended to functions on terms and set of terms as expected. Finally, a substitution σ is a *unifier* of terms t and t' iff $\sigma(t) = \sigma(t')$.

2.1.2 The Intruder Model

We use a Dolev-Yao-style intruder model, i.e., cryptography is treated as a black-box where the intruder can encrypt and decrypt terms when he has the respective keys, but he cannot break cryptography. In order to define intruder deduction in a model where the set of operators Σ is not fixed, one first needs to also specify what the intruder can compose and decompose. To that end, we denote as $\Sigma^n_{pub} \subseteq \Sigma$ the *public* functions, which are available to the intruder, of Σ of arity n, and we define a function Ana that takes a term t and returns a

pair (K, T) of sets of terms. This function specifies that, from the term t, the intruder can obtain the terms T, if he knows all the "keys" in the set K. For example, if scrypt is a public function symbol to represent symmetric encryption, we may define $\mathsf{Ana}(\mathsf{scrypt}(k, m)) = (\{k\}, \{m\})$ for any terms k and m. We define the relation $\vdash$, where $M \vdash t$ means that an intruder who knows the set of terms M can derive the message t as follows:

Definition 2.1 (Intruder Model [Hes19])**.** *We define* $\vdash$ *as the least relation that includes the knowledge, and is closed under composition with public functions and under analysis with* Ana*:*

$$\frac{}{M \vdash t} \quad t \in M \quad (\mathit{Axiom}), \qquad \frac{M \vdash t_1 \dots M \vdash t_n}{M \vdash f(t_1, \dots, t_n)} \quad f \in \Sigma^n_{pub} \quad (\mathit{Compose}),$$

$$\frac{M \vdash t \quad M \vdash k_1 \dots M \vdash k_n}{M \vdash t_i} \quad \begin{array}{l}(\mathit{Decompose}),\\ \mathsf{Ana}(t) = (K, T),\\ t_i \in T, K = \{k_1, \dots, k_n\}\end{array}$$

(*Axiom*) says that the intruder can derive everything in his knowledge. (*Compose*) says that the intruder can compose messages by applying public function symbols to derivable messages. (*Decompose*) says that the intruder can decompose, i.e., analyze, messages if he can derive the keys specified by Ana. The specification of Ana must satisfy the following requirements for the typing and compositionality results from [Hes19] to hold:

1. $\mathsf{Ana}(t) = (K, T)$ implies that K is finite and $fv(K) \subseteq fv(t)$,
2. $\mathsf{Ana}(x) = (\emptyset, \emptyset)$ for variables $x \in \mathcal{V}$,
3. $\mathsf{Ana}(f(t_1, \dots, t_n)) = (K, T)$ implies $T \subseteq \{t_1, \dots, t_n\}$, and
4. $\mathsf{Ana}(f(t_1, \dots, t_n)) = (K, T)$ implies $\mathsf{Ana}(\sigma(f(t_1, \dots, t_n))) = (\sigma(K), \sigma(T))$.

Ana is defined for arbitrary terms, including terms with variables (though the standard Dolev-Yao deduction is normally used on ground terms only). The first requirement restricts the set of keys K to be finite and to not introduce any new variables, but the keys otherwise do not need to be subterms of the term being decomposed. The second requirement says that we cannot analyze a variable. The third requirement says that the result of the analysis are *immediate* subterms of the term being analyzed. The fourth requirement says that Ana is invariant under instantiation.

Example 2.1. *Let* scrypt, crypt *and* sign *be public function symbols, representing respectively symmetric encryption, asymmetric encryption and signatures, and let* inv *be a private function symbol mapping public keys to the corresponding private key. We characterize these symbols with the following* Ana

theory: $\mathsf{Ana}(\mathsf{scrypt}(k,m)) = (\{k\},\{m\})$, $\mathsf{Ana}(\mathsf{crypt}(k,m)) = (\{\mathsf{inv}(k)\},\{m\})$, $\mathsf{Ana}(\mathsf{sign}(k,m)) = (\emptyset,\{m\})$. *To model message formats, we define a number of* transparent *functions, e.g.,* f_1 *that the intruder can open without knowing any keys:* $\mathsf{Ana}(f_1(t,t')) = (\emptyset,\{t,t'\})$. *For all other terms* t*:* $\mathsf{Ana}(t) = (\emptyset,\emptyset)$.

This model of terms and the intruder is not considering algebraic properties such as the ones needed for Diffie-Hellman-based protocols. Since handling algebraic properties is making everything more complicated, while being largely orthogonal to the points of this chapter, for simplicity, we stick with this free term algebra model.

2.1.3 Stateful Protocols

We introduce a strand-based protocol formalism for *stateful* protocols. The idea is to extend strands with a concept of *sets* to model long-term mutable state information of *stateful protocols*. The semantics is defined by a symbolic transition system where constraints are built up during transitions. The models of the constraints then constitute the concrete protocol runs.

Protocols are defined as sets $\mathcal{P} = \{R_1,\ldots\}$ of *transaction rules* of the form: $R_i = \forall x_1 \in T_1,\ \ldots,\ x_n \in T_n.\ \mathsf{new}\ y_1,\ldots,y_m.\mathcal{S}$ where $\mathcal{S}$ is a *transaction strand with sets*, i.e. of the form $\mathsf{receive}(t_1).\ldots.\mathsf{receive}(t_k).\ \phi_1.\ldots.\phi_{k'}.\ \mathsf{send}(t'_1).\ldots.\mathsf{send}(t'_{k''})$ where t and t' ranges over terms and $\bar{x}$ over finite sequences $x_1,\ldots,x_n$ of variables from $\mathcal{V}$:

$$\phi ::= \ t \doteq t' \mid \forall \bar{x}.\ t \not\doteq t' \mid t \dot{\in} t' \mid \forall \bar{x}.\ t \dot{\notin} t' \mid \mathsf{insert}(t,t') \mid \mathsf{delete}(t,t')$$

As syntactic sugar, we may write $t \not\doteq t'$ and $t \dot{\notin} t'$ in lieu of $\forall \bar{x}.\ t \not\doteq t'$ and $\forall \bar{x}.\ t \dot{\notin} t'$ when $\bar{x}$ is the empty sequence. We may also write $t \rightarrow t'$ for $\mathsf{insert}(t,t')$ and $t \leftarrow t'$ for $t \dot{\in} t'.\mathsf{delete}(t,t')$. We may also write $\xleftarrow{t}$ for $\mathsf{receive}(t)$ and $\xrightarrow{t}$ for $\mathsf{send}(t)$ when writing rules. The prefix $\forall x_1 \in T_1,\ldots,x_n \in T_n$ denotes that the transaction strand $\mathcal{S}$ is applicable for instantiations σ of the x_i variables where $\sigma(x_i) \in T_i$. The construct $\mathsf{new}\ y_1,\ldots,y_m$ represents that the occurrences of the variables y_i in the transaction strand $\mathcal{S}$ are instantiated with fresh constants.

Example 2.2. *In Figure 2.2, we formalize the* App *from Figure 2.1; we now look at a few rules as examples and discuss the others later. Note that each step of a rule is labeled by either label* App *or* $\star$ *which we also introduce below. The rule* App_3 *models an honest server* S *who first generates a new nonce* N*, stores it in a set of active nonces* $\mathsf{sent}(S,P)$ *where* P *is an identifier (alias) for*

a currently unauthenticated agent. It then adds the message $f_{challenge}(N, S)$ *to a set* $\mathsf{outbox}(S, P)$ *for being sent on a secure channel to* P*. Here,* $f_{challenge}$ *is just a format to structure the message. In* App_4*, this is received by a client* A *in its* $\mathsf{inbox}(S, P)$*, where the relation between the client* A *and its pseudonym is ensured by the positive check* $P \dot{\in} \mathsf{alias}(A)$*. The client then sends a more complex message as a reply.*

We call all variables that are introduced by a quantifier or new the *bound* variables of a transaction, and all other variables *free*. We say a transaction rule is *well-formed* if all free variables first occur in a receive step or a positive check, and the bound variables are disjoint from the free variables (over the entire protocol). For the rest of this chapter we restrict ourselves to well-formed transaction rules.

2.1.4 Stateful Symbolic Constraints

The semantics of a stateful protocol is defined as in terms of a symbolic transition system of *intruder constraints*. The intruder constraints are also represented as strands, essentially a sequence of transactions where parameters and new variables are instantiated, and are formulated from the intruder's point of view, i.e., a message sent in a transaction becomes a received message in the intruder constraint and vice-versa. We first define the semantics of constraints and then how a protocol induces a set of reachable constraints.

By $trms(\mathcal{A})$ we denote the set of terms occurring in the constraint $\mathcal{A}$. The *set of set operations* of $\mathcal{A}$, called $setops(\mathcal{A})$, is defined as follows where we assume a binary symbol $(\cdot, \cdot) \in \Sigma^2_{pub}$:

$$setops(\mathcal{A}) \equiv \{(t, s) \mid \mathsf{insert}(t, s) \text{ or } \mathsf{delete}(t, s) \text{ or } t \dot{\in} s \text{ or } \forall \bar{x}.t \dot{\notin} s \text{ occurs in } \mathcal{A}\}$$

We extend $trms(\cdot)$ and $setops(\cdot)$ to transaction strands, rules and protocols as expected. For the semantics of constraints, we first define a predicate $[\![M, D; \mathcal{A}]\!]\, \mathcal{I}$, where M is a ground set of terms (the intruder knowledge), D is a ground set

of tuples (the state of the sets), $\mathcal{A}$ is a constraint and $\mathcal{I}$ is an interpretation:

$$
\begin{array}{lll}
[\![M, D; 0]\!]\,\mathcal{I} & \text{iff} & \mathsf{true} \\
[\![M, D; \mathsf{send}(t).\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & M \vdash \mathcal{I}(t) \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; \mathsf{receive}(t).\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & [\![\{\mathcal{I}(t)\} \cup M, D; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; t \doteq t'.\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & \mathcal{I}(t) = \mathcal{I}(t') \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; (\forall \bar{x}.\ t \not\doteq t').\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \text{ and} \\
\multicolumn{3}{l}{\quad \big(\mathcal{I}(\sigma(t)) \neq \mathcal{I}(\sigma(t')) \text{ for all ground substitutions } \sigma \text{ with domain } \bar{x}\big)} \\
[\![M, D; \mathsf{insert}(t, s).\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & [\![M, \{\mathcal{I}((t, s))\} \cup D; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; \mathsf{delete}(t, s).\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & [\![M, D \setminus \{\mathcal{I}((t, s))\}; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; t \dot{\in} t'.\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & \mathcal{I}((t, s)) \in D \text{ and } [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \\
[\![M, D; (\forall \bar{x}.\ t \dot{\notin} t').\mathcal{A}]\!]\,\mathcal{I} & \text{iff} & [\![M, D; \mathcal{A}]\!]\,\mathcal{I} \text{ and} \\
\multicolumn{3}{l}{\quad \big(\mathcal{I}(\sigma((t, s))) \notin D \text{ for all ground substitutions } \sigma \text{ with domain } \bar{x}\big)}
\end{array}
$$

$\mathcal{I}$ is called a *model* of $\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, iff $[\![\emptyset, \emptyset; \mathcal{A}]\!]\,\mathcal{I}$. We define again free and bound variables as for transactions, and say a constraint is *well-formed* if every free variable first occurs in a send step or a positive check and free variables are disjoint from bound variables. We denote the free variables of a constraint $\mathcal{A}$ by $fv(\mathcal{A})$. In contrast, in a transaction we defined free variables must first occur in a receive step or a positive check; this is because constraints are formulated from the intruder's point of view. For the rest of the chapter we consider only well-formed constraints without further mention.

2.1.5 Reachable Constraints

Let $\mathcal{P}$ be a protocol. We define a state transition relation $\Rightarrow$ where states are constraints and the initial state is the empty constraint 0. First the *dual* of a transaction strand $\mathcal{S}$, written $dual(\mathcal{S})$ means "swapping" the direction of the sent and received messages of S: $dual(\mathsf{send}(t).\mathcal{S}) = \mathsf{receive}(t).dual(\mathcal{S})$, $dual(\mathsf{receive}(t).\mathcal{S}) = \mathsf{send}(t).dual(\mathcal{S})$ and otherwise $dual(\mathfrak{s}.\mathcal{S}) = \mathfrak{s}.dual(\mathcal{S})$ for any other step $\mathfrak{s}$. The transition $\mathcal{A} \Rightarrow \mathcal{A}.dual(\alpha(\sigma(\mathcal{S})))$ is possible if the following conditions are met:

1. $(\forall x_1 \in T_1, \ldots, x_n \in T_n.\mathsf{new}\ y_1, \ldots, y_m.\mathcal{S})$ is a transaction of $\mathcal{P}$,
2. $dom(\sigma) = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$,
3. $\sigma(x_i) \in T_i$ for all $i \in \{1, \ldots, n\}$,
4. $\sigma(y_i)$ is a fresh constant for all $i \in \{1, \ldots, m\}$, and
5. α is a variable-renaming of the variables of $\sigma(\mathcal{S})$ with fresh variables.

Note that by these semantics, each transaction is atomic (we do not allow partial application of a transaction), and each transaction rule can be taken arbitrarily often, thus allowing for an unbounded number of "sessions".

We say that a constraint $\mathcal{A}$ is *reachable* in protocol $\mathcal{P}$ if $0 \Rightarrow^\star \mathcal{A}$ where $\Rightarrow^\star$ is the transitive reflexive closure of $\Rightarrow$. Note that we consider only well-formed transactions and thus every reachable state is a well-formed constraint.

To model goal violations of a protocol $\mathcal{P}$ we first fix a special non-public constant unique to $\mathcal{P}$, e.g. $\mathsf{attack}_\mathcal{P}$. We can then formulate transactions that check for violations of the goal and if so, send out the message $\mathsf{attack}_\mathcal{P}$. A protocol *has an attack* if there exists a satisfiable reachable constraint of the form $\mathcal{A}. \xrightarrow{\mathsf{attack}_\mathcal{P}}$, otherwise the protocol is *secure*. This allows for modeling all security properties expressible in the geometric fragment [Alm+15; Gut14], e.g., standard reachability goals like secrecy and authentication, but not for instance privacy-type properties. We give attack rules in our examples in Example 2.3 and Example 2.4.

2.2 Stateful Vertical Composition

The compositionality result of Hess et al. [Hes19; HMB20] allows for the *parallel* composition of stateful protocols. The protocols being composed may share sets. An example would be a server that maintains a database and runs several protocols that access and modify this database.[1] After specifying an appropriate interface how these protocols may access and modify the database, one can verify each protocol individually with respect to this interface and obtain the security of the composed system.

A simple idea is to re-use this result for *vertical* composition of protocols as follows (but we explain later why this is not enough). We consider a channel protocol Ch and an application protocol App that wants to transmit messages over this channel. We regard them as running in parallel and sharing two families of sets as an interface, called inbox and outbox. In the application, if A wants to send a message to B over the channel, she inserts it into $\mathsf{outbox}(A, B)$. The channel protocol on A's side retrieves the message from $\mathsf{outbox}(A, B)$, encrypts it appropriately and transmits it to B, where it is decrypted and delivered into $\mathsf{inbox}(A, B)$. The application on B's side can now receive the message from this inbox.

[1] One could also use sets to model an abstract synchronous communication channel between participants, but that is not what we will consider here: we will only use sets that belong to one single agent who may engage in several protocols.

This paradigm is very general: the application can freely transmit messages over the channel, similar to sending on the normal network; there are no limitations on the number of messages that can be sent. Similarly, we can model a wide variety of channels and the protections they offer, e.g., our running example considers a channel where only one side is authenticated like in the typical TLS deployment. Moreover, the channel may have a handshake that establishes one or more keys that are used in the transport, where we can model both that the same key is used for several message transmissions, and that we can establish any number of such keys.

Nevertheless, there are three challenges to overcome. First, the compositionality result of [Hes19; HMB20] relies on a typing result, and this typing result is not powerful enough for our paradigm of vertical composition, due to the payload messages from the application that are inserted on the channel. The extension is in fact our first main contribution in §2.2.1. Note that §2.2.2 comes mainly from [Hes19; HMB20] but we include it here because we need to incorporate our extension of the typing result, and we need to update several definitions to take into account the specific features of vertical composition. The second challenge in §2.2.3 is to define an appropriate interface between channel and application, i.e., which security properties the channel ensures that the application can rely on. This interface allows for verifying the application completely independent of the channel, in particular, the channel can then be replaced by any other channel that implements the same interface without verifying the application again. Finally, the third and main challenge (in §2.3) is a sound abstraction of the payload messages of the application so that the channel can also be verified independent of the application.

2.2.1 Typed Model and Payloads

As already mentioned, the typing result of [Hes19; HMB20] is not general enough for our purposes: since we want to define a channel protocol independent of the application that uses the channel, we would like the messages that the channel transports to be of an abstract type $\mathfrak{p}$ (*payload*) that can, during composition, be instantiated by the concrete message types of the application protocol.

This requires, however, a substantial extension of the typing system and the typing result, since from the point of view of the channel protocol, the payload is a variable that is embedded into a channel message, e.g., a particular way to encrypt the payload. The fact that the payload is a variable reflects that the channel is indeed "agnostic" about the content that it is transporting. This is, however, incompatible with the typing result from [Hes19; HMB20], because the instantiation of the payload type with several concrete message types from the

application protocol implies that, amongst the channel, messages are unifiable message patterns of different types, which is precisely what [Hes19; HMB20] forbid.

The main idea to overcome this problem is as follows. Let $\mathfrak{T}_\mathfrak{p}$ be the set of *concrete payload types* of a given application, i.e., the types of messages the application transmits over the channel. Essentially, we want to exclude that there can ever be an ambiguity over the type of a transmitted message, i.e., that one protocol recipient sends a message of type $\tau_1 \in \mathfrak{T}_\mathfrak{p}$ and the recipient receives it as some different type $\tau_2 \in \mathfrak{T}_\mathfrak{p}$. Such ambiguity can for instance be prevented by using a distinct format for each type (e.g., using a tag).

This allows us to extend the typing and the depending compositionality results from [Hes19; HMB20] such that every instantiation of the abstract payload type $\mathfrak{p}$ with a type of $\mathfrak{T}_\mathfrak{p}$ counts as well-typed. We now introduce all concepts in the notation of [Hes19] and mark our extensions; the proof of the results under the extensions is given in Section 2.5.

Type expressions are terms built over a finite set $\mathfrak{T}_a$ of *atomic* types like Agent and Nonce and the function symbols of Σ without constants. Our extensions are the special abstract payload type $\mathfrak{p}$ and a finite non-empty set $\mathfrak{T}_\mathfrak{p}$ of concrete payload types where $\mathfrak{T}_\mathfrak{p} \subset \mathcal{T}(\Sigma \setminus \mathcal{C}, \mathfrak{T}_a)$.

Let Γ be a given type specification for all variables and constants, i.e., $\Gamma(c) \in \mathfrak{T}_a$ for every constant c and $\Gamma(x) = \tau \in \mathcal{T}(\Sigma \setminus \mathcal{C}, \mathfrak{T}_a) \cup \{\mathfrak{p}\}$ such that τ does not contain an element of $\mathfrak{T}_\mathfrak{p}$ as a subterm.

The restriction that τ does not contain an element of $\mathfrak{T}_\mathfrak{p}$ is our new addition: it prevents that the application (or the channel) uses any variables of a payload type (or variables that can be instantiated with a term that contains a payload-typed subterm). This is to prevent that we can have unifiers between terms of distinct types. Similarly, observe that $\mathfrak{p}$ can only be the type of a variable, and that it cannot occur as a proper subterm in a type expression. The type system leaves the protocol only two choices for handling payloads: either abstractly (in the channel) as a variable of type $\mathfrak{p}$ or concretely (in the application) as a non-variable term of $\mathfrak{T}_\mathfrak{p}$ type.

The typing function is extended to composed terms as follows: $\Gamma(f(t_1, \ldots, t_n)) = f(\Gamma(t_1), \ldots, \Gamma(t_n))$ for every $f \in \Sigma^n \setminus \mathcal{C}$ and terms t_i. Further, it is required that for every atomic type $\beta \in \mathfrak{T}_a$, the intruder has an unlimited supply of these terms, i.e., $\{c \in \mathcal{C} \mid c \in \Sigma_{pub}, \Gamma(c) = \beta\}$ is infinite for each atomic type β. This is needed to find solutions to inequalities.

For the payload extension, we define a partial order on types, formalizing that

the abstract payload is a generalization of the types in $\mathfrak{T}_{\mathfrak{p}}$:

- $\mathfrak{p} > \tau$ for all $\tau \in \mathfrak{T}_{\mathfrak{p}}$,
- $\tau \geq \tau'$ iff $\tau = \tau' \vee \tau > \tau'$, and
- $f(\tau_1, \ldots, \tau_n) \geq f(\tau'_1, \ldots, \tau'_n)$ iff $\tau_1 \geq \tau'_1 \wedge \cdots \wedge \tau_n \geq \tau'_n$.

We say that two types τ and τ' are *compatible* when they can be compared with the partial order. We say a substitution σ is *well-typed* iff $\Gamma(x) \geq \Gamma(\sigma(x))$ for all $x \in \mathcal{V}$. This is a generalization of [Hes19] which instead requires $\Gamma(x) = \Gamma(\sigma(x))$, i.e., we allow here the instantiation of $\mathfrak{p}$ with types from $\mathfrak{T}_{\mathfrak{p}}$. The central theorem for extending [Hes19] with payload types is that, for any two unifiable terms s and t with $\Gamma(s) \geq \Gamma(t)$, their most general unifier is well-typed:

Theorem 2.2. *Let s, t be unifiable terms with $\Gamma(s) \geq \Gamma(t)$. Then their most general unifier is well-typed.*

The modifications to the following definitions and results with respect to [Hes19] are minor: we use our updated notion of well-typed, and we use the notion of compatible types instead of the same type. We give the definitions as an almost verbatim quote without pointing out these minor differences each time.

The typing result is essentially that the messages and sub-messages of a protocol have different form whenever they do not have compatible types. Thus, given a set of messages M that occur in a protocol, define the set of sub-message patterns $SMP(M)$ as:

Definition 2.3 (Sub-message patterns [Hes19])**.** *The sub-message patterns for a set of messages M is denotes as $SMP(M)$ and is defined as the least set satisfying the following rules:*

1. *$M \subseteq SMP(M)$.*
2. *If $t \in SMP(M)$ and $t' \sqsubseteq t$ then $t' \in SMP(M)$.*
3. *If $t \in SMP(M)$ and σ is a well-typed substitution then $\sigma(t) \in SMP(M)$.*
4. *If $t \in SMP(M)$ and $\mathsf{Ana}(t) = (K, T)$ then $K \subseteq SMP(M)$.*

It is sufficient for the typing result that the non-variable sub-message patterns have no unifier unless they have compatible types:

Definition 2.4 (Type-flaw resistance (extended from [Hes19]))**.** *We call a term* t generic for a set of variables X, *if* $t = f(x_1, \ldots, x_n)$, $n > 0$ *and* $x_1, \ldots, x_n \in X$.

We say a set M *of messages is type-flaw resistant iff* $\forall t, t' \in SMP(M) \setminus \mathcal{V}.\ (\exists \sigma.\sigma(t) = \sigma(t')) \rightarrow \Gamma(t) \geq \Gamma(t') \vee \Gamma(t) \leq \Gamma(t')$. *We call a constraint* $\mathcal{A}$ *type-flaw resistant iff the following holds:*

- $trms(\mathcal{A}) \cup setops(\mathcal{A})$ *is type-flaw resistant,*
- *for all* $t \doteq t'$ *occurring in* $\mathcal{A}$*: if* t *and* t' *are unifiable then* $\Gamma(t) \leq \Gamma(t')$ *or* $\Gamma(t) \geq \Gamma(t')$,
- *for all* $\forall \bar{x}.\, t \dot{\neq} t'$ *occurring in* $\mathcal{A}$*, no subterm of* (t, t') *is generic for* $\bar{x}$*, and*
- *for all* $\forall \bar{x}.t \dot{\notin} t'$ *occurring in* $\mathcal{A}$*, no subterm of* (t, t') *is generic for* $\bar{x}$.

We say that a protocol $\mathcal{P}$ *is type-flaw resistant iff the set* $trms(\mathcal{P}) \cup setops(\mathcal{P})$ *is type-flaw resistant and all the transactions of* $\mathcal{P}$ *are type-flaw resistant.*

Our extension of the type system with the payload types requires an update of the typing result of [Hes19]. Most of this is straightforward and Theorem 2.2 is the only new theorem. In a nutshell, the typing result shows that the intruder never needs to make any ill-typed choice to perform an attack, and thus if there is an attack, then there is a well-typed one:

Theorem 2.5 ((extended from [Hes19]))**.** *If* $\mathcal{A}$ *is a well-formed, type-flaw resistant constraint, and if* $\mathcal{I} \models \mathcal{A}$*, then there exists a well-typed interpretation* $\mathcal{I}_\tau$ *such that* $\mathcal{I}_\tau \models \mathcal{A}$.

The typing requirements essentially imply that messages with different meaning should be made discernable, and this is indeed a good engineering practice. However, since we will below require that channel and application messages are also distinguishable, we will not be able to stack several layers of the same channel.

2.2.2 Parallel Compositionality

We review and adapt the *parallel composition* result from [Hes19]. The compositionality result ensures that attacks cannot arise from the composition itself. To keep track of where a step originated in a constraint, each step in a transaction is labeled with the name of the protocol, or with a special label $\star$. This $\star$ labels

$\mathsf{App}_1 \colon \forall C \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.$

App: $P \dot{\notin} \mathsf{taken}.$
App: $P \rightarrow \mathsf{taken}.$
App: $P \rightarrow \mathsf{alias}(C)$

$\mathsf{App}_2 \colon \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.$

$\star$: $\xrightarrow{\mathsf{inv}(P)}$

$\mathsf{App}_3 \colon \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, \mathsf{new}\ N.$

App: $N \rightarrow \mathsf{sent}(S, P).$
$\star$: $f_{challenge}(N, S) \rightarrow \mathsf{outbox}(S, P)$

$\mathsf{App}_4 \colon \forall S \in \mathsf{Agent}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\star$: $f_{challenge}(N, S) \leftarrow \mathsf{inbox}(S, P).$
App: $P \dot{\in} \mathsf{alias}(C).$
$\star$: $f_{response}(\mathsf{mac}(\mathsf{secret}(C, S), N)) \rightarrow \mathsf{outbox}(P, S)$

$\mathsf{App}_5 \colon \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, C \in \mathsf{Agent}.$

$\star$: $f_{response}(\mathsf{mac}(\mathsf{secret}(C, S), N)) \leftarrow \mathsf{inbox}(P, S).$
App: $N \leftarrow \mathsf{sent}(S, P)$

$\mathsf{App}_6 \colon \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\star$: $f_{response}(\mathsf{mac}(\mathsf{secret}(C, S), N)) \leftarrow \mathsf{inbox}(P, S).$
App: $N \dot{\in} \mathsf{sent}(S, P).$
App: $P \dot{\notin} \mathsf{alias}(C).$
App: $\xrightarrow{\mathsf{attack}_{\mathsf{App}}}$

Figure 2.2: Example of a login protocol

all those steps of a protocol that are relevant to the other: when the protocols to compose share any sets, then all checks and modifications to these sets must be labeled $\star$. One may always label even more steps with $\star$ to make them visible to the other protocol (this may be necessary to ensure well-formedness of the interface). From this labeling, one can obtain an interface between the protocols to compose as follows. Define the *idealization* $\mathcal{P}^\star$ of a protocol $\mathcal{P}$ as removing all steps from $\mathcal{P}$ that are not labeled $\star$. The compositionality result essentially says that the parallel composition $\mathcal{P}_1 \parallel \mathcal{P}_2$ is secure, if $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ are secure (and some syntactic conditions hold), i.e., each protocol can be verified in isolation against the idealization of the other. In the special case that no sets are shared between the two protocols, these idealizations are empty.

The protocols to compose should, to some extent, have separate message spaces, e.g., by tagging messages uniquely for each protocol. In fact, messages (or

sub-messages) that occur in both protocols must be given special attention. Unproblematic are *basic public terms* $\{t \mid \emptyset \vdash t\}$, i.e., all messages that the intruder initially knows. All other messages that can occur in more than one protocol must be part of a set *Sec* of messages that are initially considered secret. A secret may be explicitly declassified by a transaction that sends it on the network with a $\star$ label, e.g., when an agent sends a message to a dishonest agent, this message has to be explicitly declassified. For instance, *Sec* can contain all public and private keys, and then declassify all public keys and the private keys of dishonest agents. Of course it counts as an attack if any protocol leaks a secret that has not been declassified.

Formally, the *ground sub-message patterns* (*GSMP*) of a set of terms M is defined as $GSMP(M) \equiv \{t \in SMP(M) \mid fv(t) = \emptyset\}$. For a constraint $\mathcal{A}$, we define $GSMP_{\mathcal{A}} \equiv GSMP(trms(\mathcal{A}) \cup setops(\mathcal{A}))$, and similarly for protocols. It is required for composition that two protocols are disjoint in their ground sub-message except for basic public terms and shared secrets:

Definition 2.6 (GSMP disjointness [Hes19])**.** *Given two sets of terms M_1 and M_2, and a ground set of terms Sec (the shared secrets), we say that M_1 and M_2 are Sec-GSMP* disjoint *iff $GSMP(M_1) \cap GSMP(M_2) \subseteq Sec \cup \{t \mid \emptyset \vdash t\}$.*

For declassification, we extend the definition from [Hes19]: we close the declassified messages under intruder deduction. We denote the Dolev-Yao closure of a set of messages M by $\mathcal{DY}(M) = \{t \mid M \vdash t\}$. We now define that what the intruder can derive from declassified messages is also declassified:[2]

Definition 2.7 (Declassification (extended from [Hes19]))**.** *Let $\mathcal{A}$ be a labeled constraint and $\mathcal{I}$ a model of $\mathcal{A}$. Then the set of declassified secrets of $\mathcal{A}$ under $\mathcal{I}$ is $declassified_{\mathcal{DY}}(\mathcal{A}, \mathcal{I}) \equiv \mathcal{DY}(\{t \mid \star\colon \xleftarrow{t} \text{ occurs in } \mathcal{I}(\mathcal{A})\})$.*

This modification requires the update of several definitions and proofs in [Hes19]. We provide the details of this extension in Section 2.5.

If the intruder learns a secret that has not been declassified then it counts as an attack. We say that the protocol $\mathcal{P}$ *leaks* a secret s if there is a reachable satisfiable constraint $\mathcal{A}$ where the intruder learns s before it is declassified:

Definition 2.8 (Leakage ([Hes19]))**.** *Let Sec be a set of secrets and $\mathcal{I}$ be a model of the labeled constraint $\mathcal{A}$. $\mathcal{A}$ leaks a secret from Sec under $\mathcal{I}$ iff there exists $s \in Sec \setminus declassified_{\mathcal{DY}}(\mathcal{A}, \mathcal{I})$ and a protocol-specific label l such that $\mathcal{I} \models \mathcal{A}|_l.\mathsf{send}(s)$ where $\mathcal{A}|_l$ is the projection of $\mathcal{A}$ to the steps labeled l or $\star$.*

[2] Each protocol can define more refined secrecy goals to catch unintended declassifications (so it is not a restriction in the protocols we can model), while the Dolev-Yao closure of declassification is necessary since later after abstraction of payload messages, we cannot reason about deductions from these payload messages anymore.

We define the *traces* of a protocol $\mathcal{P}$ as the "solved" ground instances of reachable constraints: $\mathit{traces}(\mathcal{P}) \equiv \{\mathcal{I}(\mathcal{A}) \mid 0 \Rightarrow^\star \mathcal{A}\wedge\mathcal{I} \models \mathcal{A}\}$. Next is the compositionality requirement on protocols that ensures that all traces are parallel composable:

Definition 2.9 (Parallel composability [Hes19])**.** *Let $\mathcal{P}_1 \parallel \mathcal{P}_2$ be a composed protocol and let Sec be a ground set of terms. Then $(\mathcal{P}_1, \mathcal{P}_2, \mathit{Sec})$ is parallel composable iff*

1. *$\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is Sec-GSMP disjoint from $\mathcal{P}_1^\star \parallel \mathcal{P}_2$,*
2. *for all $s \in \mathit{Sec}$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in \mathit{Sec}$,*
3. *for all $l\colon (t, s)$, $l'\colon (t', s') \in \mathit{labeledsetops}(\mathcal{P}_1 \parallel \mathcal{P}_2)$, if (t, s) and (t', s') are unifiable then $l = l'$,*
4. *$\mathcal{P}_1 \parallel \mathcal{P}_2$ is type-flaw resistant and $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_1^\star$ and $\mathcal{P}_2^\star$ are well-formed.*

where $\mathit{labeledsetops}(\mathcal{P}) \equiv \{l\colon (t, s) \mid l\colon \mathsf{insert}(t, s)$ or $l\colon \mathsf{delete}(t, s)$ or $l\colon t \dot{\in} s$ or $l\colon (\forall \bar{x}.t \dot{\notin} s)$ occurs in $\mathcal{P}\}$.

Composition of secure, parallel composable protocols is secure:

Theorem 2.10 (Parallel Composition [Hes19])**.** *If $(\mathcal{P}_1, \mathcal{P}_2, \mathit{Sec})$ is parallel composable and $\mathcal{P}_1 \parallel \mathcal{P}_2^\star$ is well-typed secure in isolation, and $\mathcal{P}_1^\star \parallel \mathcal{P}_2$ does not leak a secret under any well-typed model, then all goals of $\mathcal{P}_1$ hold in $\mathcal{P}_1 \parallel \mathcal{P}_2$.*

2.2.3 Channels and Applications

As our second contribution in this chapter, we propose a general paradigm for expressing vertical composition problems as parallel composition of a channel protocol Ch and an application protocol App that transmits messages over the channel. We employ the parallel compositionality result from [Hes19], where we connect the two protocols with each other via shared sets inbox and outbox. We may even denote this by using the notation $\frac{\mathsf{App}}{\mathsf{Ch}}$, emphasizing it is essentially a parallel composition. Let us first look more closely to the application protocols:

Definition 2.11 (Application Protocol)**.** *Let* inbox *and* outbox *be two families of sets (e.g., parametrized over agent names). An* application protocol App *is a protocol that does not contain any normal sending and receiving step, but may insert messages into sets of the* outbox *family, and retrieve messages from sets of the* inbox *family and perform no other operations on these sets. The* inbox *and* outbox *steps are labeled $\star$ (since these sets are shared with the channel protocol),*

and no other operations are labeled $\star$— except potentially set operation steps needed to ensure well-formedness of the idealization $\mathsf{App}^\star$*, whose sets are only accessed by the application. The set of concrete payload types* $\mathfrak{T}_\mathfrak{p}$ *of the type system is determined to contain exactly those message types that are inserted into an* outbox *or received from an* inbox *by the application. Finally, let the set Sec of shared secrets contain all application messages.*

This definition does not specify what guarantees the application can get from the channel (like secure transmission). This will in fact be formalized next as part of the channel protocol. Recall also that our type system requires that no variable may have a type in which a $\mathfrak{T}_\mathfrak{p}$ type occurs as a subterm.

Example 2.3. *We formalize the running example from Figure 2.1, i.e., a login protocol, as an application that runs over a secure channel where one side is not* yet *authenticated. As explained, we formalize the unauthenticated endpoint of a channel using an alias P, which is an unauthenticated public key and the owner is the person who created P and knows the corresponding private key* $\mathsf{inv}(P)$*. Thus let* Names *be a set of the public constants that is further partitioned into a subset* Agent*, representing real names of agents, and a subset* Alias*, representing the aliases. The set* Names *is further partitioned between honest principals* Hon *and dishonest principals* Dis*. We write for example* $\mathsf{Agent}|_{\mathsf{Hon}}$ *when we restrict the agent set to the honest principals. Since global constants cannot be freshly created, the rules* App_1 *and* App_2 *formalize that every agent can assume any alias P that has not yet been taken, mark it as taken, and insert it into its set of aliases. For the honest users, the knowledge of the corresponding* $\mathsf{inv}(P)$ *is implicitly understood, for the dishonest agents, we declassify* $\mathsf{inv}(P)$*. P is public anyway, and by obtaining* $\mathsf{inv}(P)$ *the intruder will be able to use alias P.*[3] *Note that, in this way, the protocol can simply distinguish between pseudonyms belonging to honest and dishonest agents—which of course is not visible to any agent. Note also that we do not need to explicitly specify that the honest agents also know* $\mathsf{inv}(P)$ *to every P they pick.*

The actual protocol begins with App_3*, and it assumes a secure channel between some server S and some unauthenticated client under some alias P. Here, the server S generates a fresh nonce N (of type* Nonce*) and inserts it into its set* $\mathsf{sent}(S, P)$ *of unanswered challenges. Then, S uses the channel to P by inserting $f_{challenge}(N, S)$ into its outbox for P, where $f_{challenge}$ is message format, i.e., a transparent function. The rule* App_4 *describes how this message is received by the unauthenticated client C who is the owner of P. The client computes a MAC of the challenge N with a secret pre-shared with the server,* $\mathsf{secret}(C, S)$*. Here,*

[3] This declassification step is in principle forbidden by Definition 2.11. However, as we see below at the channel protocol, the channel will automatically declassify all payloads sent to a dishonest recipient, and thus, we can see declassification of $\mathsf{inv}(P)$ in the application as syntactic sugar for $\forall P \in \mathsf{Alias}_{\mathsf{Dis}}, C \in \mathsf{Agent}|_{\mathsf{Dis}}.\star\colon \mathsf{inv}(P) \to \mathsf{outbox}(C, C)$.

mac *is a public function, whereas* secret *is a private function. This in fact models a personal code card where agents can look up the answer to a challenge* N *from a server.* C *inserts its response,* $f_{response}(\mathsf{mac}(\mathsf{secret}(C,S),N))$ *where* $f_{response}$ *is another message format distinct from* $f_{challenge}$*, into its* $\mathsf{outbox}(P,S)$*. In the rule* App_5*, an honest server can retrieve* C*'s message from its set* $\mathsf{inbox}(P,S)$*, where* $N \leftarrow \mathsf{sent}(S,P)$ *means that the server both checks that* N *is an active challenge for* P *and removes it from the set. At this point,* S *accepts* C *as authenticated, i.e.,* S *believes that* C *is indeed the owner of alias* P*, and thus the other endpoint of the secure channel. Consequently,* App_6 *defines that it counts as an attack if that is actually not the case: this rule can fire when a server could accept the login (with* App_5*) while* P *is actually* not *owned by* C*. Note that in this rule, we limit* C *and* S *to honest agents, similar to standard authentication goals (if the intruder authenticates under the name of any dishonest agents, there are no security guarantees for such sessions).* App_6 *is in fact a non-injective authentication goal (it does not check for replay); we discuss such examples in Section 2.7.*

The payload types of this application are

$$\mathfrak{T}_{\mathfrak{p}} = \{f_{challenge}(\mathsf{Nonce},\mathsf{Agent}), f_{response}(\mathsf{mac}(\mathsf{secret}(\mathsf{Agent},\mathsf{Agent}),\mathsf{Nonce}))\}.$$

Observe that the example protocol would indeed have an attack if we implemented the channel as simply transmitting the payload messages in clear text through the network. The application obviously needs the channel to implement some properties in order to be secure, and this is indeed now part of the formalization of the channel itself:

Definition 2.12 (Channel Protocol)**.** *Let again* inbox *and* outbox *be families of sets. A* channel protocol *is a protocol that uses these families only in a particular way: it only retrieves from* outbox *as variable* X *of the abstract payload type* $\mathfrak{p}$ *and only inserts to* inbox *also with* X *of type* $\mathfrak{p}$*, and these steps must be labeled star.*

Example 2.4 (Unilaterally authenticated secure channel)**.** *We now model the channel protocol from Figure 2.1 in our framework as a unilaterally authenticated secure channel, similar to what TLS without client authentication would establish. We consider the same sets of agents that we used in Example 2.3. Additionally, we have a function* $\mathsf{pk}(A)$ *to model an authenticated public key of a server* A *and the corresponding private key is* $\mathsf{inv}(\mathsf{pk}(A))$*. We define all these public keys and the private keys of any dishonest* A *as public terms.*

In the first rule Ch_1 *in Figure 2.3, an honest client with alias* P *generates a session key* K *(of type* Key*) for talking to an agent* B*, stores it in* $\mathsf{sessKeys}(P,B)$*, and signs it with the private key* $\mathsf{inv}(P)$ *of their alias, and encrypts it with*

$$\frac{\mathsf{Ch}_1 : \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.}{\begin{array}{l}\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(P, B).\\ \mathsf{Ch} : \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}\end{array}}$$

$$\frac{\mathsf{Ch}_2 : \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{Ch} : \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}.\\ \mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(B, P)\end{array}}$$

$$\frac{\mathsf{Ch}_3 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star : X \leftarrow \mathsf{outbox}(A, B).\\ \mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).\\ \star : X \rightarrow \mathsf{secCh}(A, B).\\ \mathsf{Ch} : \xrightarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,X))}\end{array}}$$

$$\frac{\mathsf{Ch}_4 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,X))}.\\ \mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(B, A).\\ \star : X \mathrel{\dot\in} \mathsf{secCh}(A, B).\\ \star : X \rightarrow \mathsf{inbox}(A, B)\end{array}}$$

$$\frac{\mathsf{Ch}_5 : \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.}{\begin{array}{l}\star : X \leftarrow \mathsf{outbox}(A, B).\\ \star : \xrightarrow{X}\end{array}}$$

$$\frac{\mathsf{Ch}_6 : \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.}{\begin{array}{l}\star : \xleftarrow{X}.\\ \star : X \rightarrow \mathsf{inbox}(A, B)\end{array}}$$

$$\frac{\mathsf{Ch}_7 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,X))}.\\ \mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).\\ \star : X \mathrel{\dot\notin} \mathsf{secCh}(A, B).\\ \mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}\end{array}}$$

Figure 2.3: Example for an unilaterally authenticated pseudonymous secure channel

the public key $\mathsf{pk}(B)$ *of* B*. Note that a similar protocol for a mutually secure channels would just instead of* P *use a real name* A*, and use* $\mathsf{inv}(\mathsf{pk}(A))$ *for signing, but this would require clients to have an authenticated public key. Also note that this implicitly assumes that all users know the public keys of all servers, and in Section 2.7, we consider variants where this is actually communicated using key certificates.*

In Ch_2*, an honest agent* B *is receiving a session key* K *encrypted with his public key and signed by an agent under an alias* P*. They insert* K *into* $\mathsf{sessKeys}(B, P)$*. Note that this is a minimal key exchange protocol for simplicity (that does not protect against replay). One may in fact here install a more complicated protocol that also uses* sessKeys *as an interface to the other rules* $\mathsf{Ch}_3 \ldots \mathsf{Ch}_7$ *as a sequential composition.*

The following rules use the session keys, and they do not distinguish whether endpoints are real names (from the set Agent*) or aliases (from the set* Alias*), and instead use the union set* Names*. In* Ch_3*, an honest A can transmit a payload message X that an application protocol has inserted into an* outbox *set using for encryption any session key K that was established for that recipient. The term X bears the type payload* $\mathfrak{p}$*, and in a composition,* $\mathfrak{p}$ *will be instantiated with all the concrete payload types from the application, like* $\mathfrak{T}_{\mathfrak{p}}$ *in Example 2.3. Let us ignore the insertion into the set* secCh *for a moment.*

In Ch_4*, an honest B can receive the encrypted payload X from A, provided it is encrypted correctly with a key K that has been established with A. Both A and B can be a real name or an alias. It is inserted into* $\mathsf{inbox}(A, B)$ *to make it available on an application level. We ignore again the* secCh*.*

Rules Ch_5 *and* Ch_6 *describe symmetrically the sending and the receiving operations for a dishonest principal, i.e., the intruder can receive message directed to any dishonest recipient, and send messages under the identity of any dishonest sender, where recipient and sender can both be real names or aliases. Note also that* Ch_5 *means declassifying the payload X: the message was directed to a dishonest agent, so if it was a secret so far, it cannot be considered one anymore.*

For formulating goals, and especially the interface to the application, we introduce the set $\mathsf{secCh}(A, B)$ *that represents all messages ever sent by an honest A for an honest B. Note the similarity between rules* Ch_4 *and* Ch_7*: they are applicable when a message that looks like a legitimate message from honest A to honest B with the right session key arrives at B.* Ch_4 *can fire if the corresponding X was indeed sent by A for B, i.e.,* secCh *holds, and otherwise we have an authentication attack, and* Ch_7 *fires. This expresses that the channel ensures non-injective agreement of the payload messages: recipient B can be sure it came from A, but we do not check for replay here. In fact, in this simple channel, the intruder can simply replay the encrypted message so that B can receive a payload more often than it was sent. For an example of a channel offering replay protection, see Section 2.7.*

Now consider the idealization $\mathsf{Ch}^\star$ *of the protocol, i.e., the restriction to $\star$-labeled steps of the* Ch *protocol as in Figure 2.4: this describes abstractly every changes that the channel can ever do to the sets* outbox *and* inbox *that it shares with the application (given that the channel protocol does not have an attack, i.e.,* Ch_7 *can never fire): all messages sent by honest A to honest B move to a set* $\mathsf{secCh}(A, B)$ *and from there into the inbox of B, and the intruder can read messages directed to a dishonest B and send messages as any dishonest A.*

Observe how interface and attack declaration complement each other: when a message arrives at an honest B coming apparently from an honest A, either this

$\underline{\mathsf{Ch}_3^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}$
$\star\colon X \leftarrow \mathsf{outbox}(A,B).$
$\star\colon X \rightarrow \mathsf{secCh}(A,B).$

$\underline{\mathsf{Ch}_4^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}$
$\star\colon X \mathrel{\dot\in} \mathsf{secCh}(A,B).$
$\star\colon X \rightarrow \mathsf{inbox}(A,B)$

$\underline{\mathsf{Ch}_5^\star\colon \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.}$
$\star\colon X \leftarrow \mathsf{outbox}(A,B).$
$\star\colon \xrightarrow{X}$

$\underline{\mathsf{Ch}_6^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.}$
$\star\colon \xleftarrow{X}.$
$\star\colon X \rightarrow \mathsf{inbox}(A,B)$

Figure 2.4: Idealization of the channel protocol from Figure 2.3

is true (and rule Ch_4 *is applicable), or not (and rule* Ch_7 *is applicable). The former case is what the interface advertises, while if the latter can ever happen, the verification of the channel fails.*

Secrecy is specified implicitly: recall that all messages from the application are part of the set Sec of shared secrets and it counts as an attack if a protocol leaks a secret that has not been explicitly declassified. Here we only declassify messages that are directed at a dishonest agent ($\mathsf{Ch}_5^\star$*), i.e., the interface advertises that it will keep all messages secret (if they are not public anyway) except those sent to dishonest recipients.*

Example 2.5 (Perfect Forward secrecy)**.** *Figure 2.5 shows a modification of our running example that also provides perfect forward secrecy for the channel, i.e., even when the private key of the server* B *is given to the intruder, it does not compromise past sessions. For this reason, we have a new special rule* Ch_{0a} *that gives* $\mathsf{inv}(\mathsf{pk}(B))$ *to the intruder and marks* B *as compromised. The transactions of the key-exchange (*Ch_{0b}*,* Ch_1 *and* Ch_2*) require that* B *is uncompromised; however, after the key* K *is established, the channel allows for transactions with a compromised* B*. In our running example, the channel would not provide forward secrecy because the intruder could learn all session keys* K *any client has established with* B*, and thus decrypt all traffic with* B*. We have a slightly more complicated key exchange: in* Ch_{0b} *the server generates a new (ephemeral) public key* PK *and signs it.* Ch_1 *is similar to the running example, except that the key* K *is now encrypted with* PK *instead of* $\mathsf{inv}(\mathsf{pk}(B))$*. This is somewhat simulating an aspect of Diffie-Hellman, since both* PK *and* P *play the role of ephemeral keys, and later discovery of the authentication key* $\mathsf{inv}(\mathsf{pk}(B))$ *does not reveal the session key* K*. We do not need to even update the specification of the goals, because the channel should provide exactly the same interface to the application: it keeps the secrecy of all payload messages that have not been explicitly declassified (either by the application or by sending to a dishonest agent with* Ch_5*). It is merely a change on the channel level that long-term*

$\underline{\mathsf{Ch}_{0a}\colon \forall B \in \mathsf{Agent}.}$

$\mathsf{Ch}\colon B \to \mathsf{compromized}$

$\mathsf{Ch}\colon \xrightarrow{\mathsf{inv}(\mathsf{pk}(B))}$

$\underline{\mathsf{Ch}_{0b}\colon \forall P \in \mathsf{Alias}_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ PK.}$

$\mathsf{Ch}\colon B \dot{\notin} \mathsf{compromized}$

$\mathsf{Ch}\colon PK \to \mathsf{tmpK}(B,P)$

$\mathsf{Ch}\colon \xrightarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)),f_{PK}(B,P,PK))}$

$\underline{\mathsf{Ch}_1\colon \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.}$

$\mathsf{Ch}\colon B \dot{\notin} \mathsf{compromized}$

$\mathsf{Ch}\colon \xleftarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)),f_{PK}(B,P,PK))}$

$\mathsf{Ch}\colon K \to \mathsf{sessKeys}(P,B).$

$\mathsf{Ch}\colon \xrightarrow{\mathsf{crypt}(PK,\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}$

$\underline{\mathsf{Ch}_2\colon \forall A \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.}$

$\mathsf{Ch}\colon B \dot{\notin} \mathsf{compromized}$

$\mathsf{Ch}\colon \xleftarrow{\mathsf{crypt}(PK,\mathsf{sign}(\mathsf{inv}(P)),f_{newSess}(P,B,K))}$.

$\mathsf{Ch}\colon PK \leftarrow \mathsf{tmpK}(B,P)$

$\mathsf{Ch}\colon K \to \mathsf{sessKeys}(B,P)$

Figure 2.5: Example for a channel with perfect forward secrecy.

private keys may be lost.[4]

Let us take stock. We can define application and channel protocols App and Ch that interact with each other via the inbox and outbox sets, and the idealization of the channel protocol $\mathsf{Ch}^\star$ describes abstractly the properties that the channel guarantees, such as authentication or secrecy properties, and in fact, one can use this for more complicated properties like preserving the order of transmissions. Verifying the application now essentially means to verify that $\mathsf{App} \parallel \mathsf{Ch}^\star$ is secure, i.e., that the application has no attack as long as the channel does not manipulate the inbox and outbox sets in any other way than described in $\mathsf{Ch}^\star$ and does not leak any messages except those explicitly declassified in $\mathsf{Ch}^\star$. The first main point of composition is here that this verification $\mathsf{App} \parallel \mathsf{Ch}^\star$ is independent of the concrete implementation Ch: *any* channel Ch' with $\mathsf{Ch}'^\star = \mathsf{Ch}^\star$ would work! In fact, using Theorem 2.10 we can derive:

Theorem 2.13 (Vertical Composition (with unabstracted payload))**.** *Given a channel protocol* Ch *and an application protocol* App *w.r.t. a ground set Sec of*

[4]Note that in our specification the public-key infrastructure is only used by the channel. If the application were to use them, then $\mathsf{inv}(\mathsf{pk}(B))$ would have to be part of *Sec*, and thus declassified in Ch_{0a}, and similarly compromized would have to be a shared set (i.e., operations labeled $\star$).

terms where the only shared sets are the inbox *and* outbox *sets*[5] *s.t.* (Ch, App, *Sec*) *is parallel composable. If both* App ‖ Ch* *and* App* ‖ Ch *are secure and do not leak secrets (in the typed model) then the vertical composition* $\frac{\mathsf{App}}{\mathsf{Ch}}$ *is secure (even in the untyped model).*

The verification of App ‖ Ch* is now independent of the concrete channel, however the verification of App* ‖ Ch is still depending largely on the concrete messages of App, especially if, to achieve well-formedness, almost everything in App has to be labeled $\star$. The next section is solving exactly this.

2.3 Abstracting the Payload

As the third and core contribution of this chapter, we show how to verify the channel *independent* of the payload messages of a particular application. After recasting the vertical composition as a parallel composition, the problem is that a concrete execution of Ch ‖ App* has the concrete messages from the application at least in the outbox and inbox sets and as subterms of the messages that the channel transmits. There are two reasons why we want to do this independently of App: it should be simpler (we do not want the complexity of the messages of App) and more general (we do not want to have to verify the channel again when considering a different application).

We show a transformation of the problem, at the end of which we have a completely App-independent protocol Ch$^\sharp$ such that each transformation is sound (if there is an attack, then so there is after the transformation). If we manage to verify Ch$^\sharp$, then we have also verified Ch ‖ App* and (with the results of the previous section) the vertical composition $\frac{\mathsf{App}}{\mathsf{Ch}}$. In fact, the requirements for automated verification tools to handle Ch$^\sharp$ are modest: besides whatever the modeling of the channel itself requires, our result will only require a supply of fresh constants that can be used as payloads and which can occasionally be given to the intruder—and the tool needs to be able to track which ones are still secret.

[5] Note: $\star$-labeled set operations on other sets (like secCh in the example) are *not* forbidden by this as long as each set is mentioned in only one of the protocols. This then simply means that the respective set is not "hidden" by the interface.

2.3.1 Abstract Constants

At the core of the transformation is the idea to replace the concrete payload messages that can be inserted on the channel by abstract constants in a sound way. The intuition is as follows: the precise form of the messages of the application should not matter as long as we can ensure that they do not interfere with the form of the messages of the channel. For that purpose, let $\mathfrak{G} \subseteq Sec$ be an infinite set of constants disjoint from $GSMP_{\mathsf{Ch}}$ and from $GSMP_{\mathsf{App}}$. All elements of $\mathfrak{G}$ are elements of a new type $\mathfrak{a}$ that does not occur in App or Ch. We now define a protocol $\mathsf{Ch}^\sharp$ where we replace payload messages X of type $\mathfrak{p}$ by variables of this type $\mathfrak{a}$, and where we remove the outbox and inbox sets.

Moreover, we introduce two new sets, closed and opened. We use these two sets during transactions to keep track of which constants from $\mathfrak{G}$ have been declassified, namely they are in closed if they have not been declassified, in opened otherwise.

2.3.2 Translation to the abstract channel

We now explain formally the transformation of Ch into the protocol $\mathsf{Ch}^\sharp$. As explained, in the rules of $\mathsf{Ch}^\sharp$, the payload messages of type $\mathfrak{p}$ have been replaced by variables of type $\mathfrak{a}$, thus allowing us to verify the channel without considering the concrete terms from the application. Furthermore, since after this abstraction we do not need the interface with the application anymore, we drop the steps with outbox and inbox sets. We prove later in this chapter that $\mathsf{Ch}^\sharp$ has an attack if $\mathsf{Ch} \parallel \mathsf{App}^\star$ has, i.e., this abstraction is sound.

Definition 2.14 (Transformation of rules of Ch to rules of $\mathsf{Ch}^\sharp$)**.** *Given a channel rule* Ch_i*, its translation to* $\mathsf{Ch}^\sharp$ *rules is as follows.*

- *we remove all the steps containing* outbox *or* inbox *sets,*
- *if the rule contains any variable X of type $\mathfrak{p}$, we make a case split into two rules: one containing the positive check* $(\star\colon X \mathrel{\dot\in} \mathsf{opened})$ *and the other containing* $(\star\colon X \mathrel{\dot\in} \mathsf{closed})$*, and X is now of type $\mathfrak{a}$. We repeat this case splitting until there is no more variable of type $\mathfrak{p}$, and*
- *for every rule that contains both* $(\star\colon X \mathrel{\dot\in} \mathsf{closed})$ *and* $(\star\colon \xrightarrow{X})$*, we replace these two steps by* $(\star\colon X \leftarrow \mathsf{closed}.\ \star\colon X \rightarrow \mathsf{opened}.\ \star\colon \xrightarrow{X})$.

Finally, we add the special rule: $\mathsf{Ch}^\sharp_{\mathsf{new}}\colon \mathsf{new}\ G.\star\colon G \rightarrow \mathsf{closed}$ *for creating new constants.*

The idea of the special rule ($\mathsf{Ch}^\sharp_{\text{new}}$) is that any "new" abstract constant is first inserted in closed, and that they are moved to opened and reveled to the intruder whenever they represent a payload that is declassified. Note that this setup handles both payloads that are secret to the intruder and payloads that are known to the intruder, and further they can be fresh or they can be a repetition. We now give as an example the translation of the rules from Figure 2.3:

$$\frac{\mathsf{Ch}^\sharp_1 \colon \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.}{\begin{array}{l}\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(P, B).\\ \mathsf{Ch} : \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_2 \colon \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{Ch} : \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}.\\ \mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(B, P)\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{3a,b} \colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star : G \dot{\in} \mathsf{opened}/G \dot{\in} \mathsf{closed}.\\ \mathsf{Ch} : K \dot{\in} \mathsf{sessKeys}(A, B).\\ \star : G \rightarrow \mathsf{secCh}(A, B).\\ \mathsf{Ch} : \xrightarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,G))}\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{4a,b} \colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star : G \dot{\in} \mathsf{opened}/G \dot{\in} \mathsf{closed}.\\ \mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,G))}.\\ \mathsf{Ch} : K \dot{\in} \mathsf{sessKeys}(B, A).\\ \star : G \dot{\in} \mathsf{secCh}(A, B).\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{5a} :}{\begin{array}{l}\star : G \dot{\in} \mathsf{opened}.\\ \star : \xrightarrow{G}\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{5b} :}{\begin{array}{l}\star : G \leftarrow \mathsf{closed}\\ \star : G \rightarrow \mathsf{opened}.\\ \star : \xrightarrow{G}\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{6a,b} :}{\begin{array}{l}\star : G \dot{\in} \mathsf{opened}/G \dot{\in} \mathsf{closed}.\\ \star : \xleftarrow{G}.\end{array}}$$

$$\frac{\mathsf{Ch}^\sharp_{\mathsf{new}} \colon \mathsf{new}\ G.}{\star : G \rightarrow \mathsf{closed}}$$

$$\frac{\mathsf{Ch}^\sharp_{7a,b} \colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star : G \dot{\in} \mathsf{opened}/G \dot{\in} \mathsf{closed}.\\ \mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,G))}.\\ \mathsf{Ch} : K \dot{\in} \mathsf{sessKeys}(A, B).\\ \star : G \dot{\notin} \mathsf{secCh}(A, B).\\ \mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}\end{array}}$$

Figure 2.6: Abstraction for our example channel Ch from 2.3

Example 2.6 (Abstraction of the channel from Example 2.4)**.** *In Figure 2.6, we give the set of rules of* $\mathsf{Ch}^\sharp$ *transformed from the set of rules given in Figure 2.3 following Definition 2.14 where we have actually renamed the payload variables*

X into G to emphasize that they now bear the type $\mathfrak{a}$. We consider the same set of agents that we used in Example 2.3. We write $\star\colon G \mathrel{\dot\in} \mathsf{opened}/G \mathrel{\dot\in} \mathsf{closed}$ as a syntactic sugar to avoid writing two rules, one with $(\star\colon G \mathrel{\dot\in} \mathsf{opened})$ and one with $(\star\colon G \mathrel{\dot\in} \mathsf{closed})$, when all other things are equal.

Ch_1 and Ch_2 are not affected by the transformation since they do not deal with any payload messages. These two rules can be seen as "pure" channel rules since they are already independent of any application protocol. Thus $\mathsf{Ch}_1^\sharp$ and $\mathsf{Ch}_2^\sharp$ are identical to the original rules.

A payload message X occurs in Ch_3, thus we need to divide this rule into two rules. The rule $\mathsf{Ch}_{3a}^\sharp$ contains the positive check $(\star\colon G \dot\in \mathsf{opened})$ at the beginning, whereas the rule $\mathsf{Ch}_{3b}^\sharp$ contains $(\star\colon G \mathrel{\dot\in} \mathsf{closed})$. The further transformations are similar for the two rules since there is no declassification step for the payload. The step containing the set operation for outbox is dropped. The payload message inserted into the secCh set is replaced by the variable G of type $\mathfrak{a}$, as is the payload message in the transmitted message. The transformations for the rule Ch_4 are very similar. It needs to be split into two rules, the inbox step is dropped and the payload messages X are replaced by a variable G of type $\mathfrak{a}$.

The rule Ch_5 also has to be split into two rules. Since the payload is declassified upon transmission to the intruder, the transformations are different for the two rules. In $\mathsf{Ch}_{5a}^\sharp$, we add the positive check $(\star\colon G \dot\in \mathsf{opened})$. We then simply remove the step with outbox and replace the payload message by the variable G of type $\mathfrak{a}$. In $\mathsf{Ch}_{5b}^\sharp$, we add the positive check $(\star\colon G \mathrel{\dot\in} \mathsf{closed})$. We also remove the step with outbox. Since, the remaining step, after replacing the payload with the variable of type $\mathfrak{a}$, is the declassification of that variable, and since that G is in closed, we need to replace the previously added positive check and the declassification step by $(\star\colon G \leftarrow \mathsf{closed}.\star\colon G \rightarrow \mathsf{opened}.\star\colon \xrightarrow{G})$. We correctly abstracted the declassification of the original payload.

The rule Ch_6 has to be be split into two rules. The step with the set inbox is removed and the payload is replaced by a variable of type $\mathfrak{a}$ in both rules. Note that these rules become superfluous (since they contain only a check and a receive) but we keep them here to illustrate the transformation. We also add the rule $\mathsf{Ch}_{new}^\sharp$ that me mentioned before. Finally, Ch_7 has also to be split into two rules. Further, in both rules, the payload variable X is replaced by the variable G of type $\mathfrak{a}$.

Recall that the parallel composability of Ch and App requires that $GSMP_{\mathsf{Ch}} \cap GSMP_{\mathsf{App}^\star} \subseteq Sec \cup \{t \mid \emptyset \vdash t\}$, and that the definition of an application requires that $GSMP_{\mathsf{App}} \subseteq Sec \cup \{t \mid \emptyset \vdash t\}$. For the abstraction of the payload we actually need something even stronger, namely that the application is completely

disjoint from the channel without payloads. Having defined $\mathsf{Ch}^\sharp$, we can specify this simply as $GSMP_{\mathsf{Ch}^\sharp} \cap GSMP_{\mathsf{App}} \subseteq \{t \mid \emptyset \vdash t\}$, i.e., the only terms common to the channel and the application are public. This allows us to label any ground term and subterm of a channel and an application protocol in any well-typed instantiation in a unique way either as *Pub* (when it is in $\{t \mid \emptyset \vdash t\}$), Ch (when it is in $GSMP_{\mathsf{Ch}^\sharp}$ or a variable of type $\mathfrak{p}$) or App (when it is in $GSMP_{\mathsf{App}}$). We require that when $f(t_1, \ldots, t_n)$ is a message of $GSMP_{\mathsf{Ch}}$ and $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ that none of the keys in K or their subterms are labeled App, i.e., the channel never uses payload messages in key positions. This is because application payloads are abstracted and thus application payload messages cannot be used to encrypt channel messages. In fact, a violation of this rule would be a poor practice of protocol design.

Let us now collect all the conditions we stated for vertical composition in the following notion of vertical composability:

Definition 2.15 (Vertical Composability)**.** *Let* Ch *be a channel protocol,* App *an application protocol w.r.t. a ground set Sec of terms. Then* $(\mathsf{Ch}, \mathsf{App}, Sec)$ *is* vertical composable *iff*

1. $(\mathsf{Ch}, \mathsf{App}, Sec)$ *is parallel composable,*
2. $GSMP_{\mathsf{App}} \subseteq Sec \cup \{t \mid \emptyset \vdash t\}$*,*
3. $GSMP_{\mathsf{Ch}^\sharp} \cap GSMP_{\mathsf{App}} \subseteq \{t \mid \emptyset \vdash t\}$*, and*
4. *none of the keys in* K *or their subterms in an analysis rule for a channel term s.t.* $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ *are labeled* App*.*

The first condition was also required in Theorem 2.13. Conditions (2)–(3) give the disjointness requirements. Condition (4) requires that the keys or their subterms are not labeled App. We now can give the main theorem:

Theorem 2.16. *Let* Ch *be a channel protocol and* App *an application protocol w.r.t. a ground set Sec of terms that are vertical composable. If there is an attack in* $\mathsf{Ch}||\mathsf{App}^\star$*, then there is one in the protocol* $\mathsf{Ch}^\sharp$*.*

The proof is given in Section 2.4 and the proof idea is as follows. First, we define an intermediate channel protocol $\mathsf{Ch}^{\mathsf{App}}$ where the payloads are instantiated by arbitrary concrete ground terms from the application and where we delete the steps with the sets outbox and inbox. We show that this protocol has an attack if $\mathsf{Ch} \parallel \mathsf{App}^\star$ has. Then we define a translation of ground traces of $\mathsf{Ch}^{\mathsf{App}}$ that replaces the concrete payloads with abstract ones, keeping track of which are

declassified, and show that the resulting trace is a trace of $\mathsf{Ch}^\sharp$. Again, we show that all attacks are preserved.

This last result allows us to conclude on the security of the vertical composition of a channel and an application protocol:

Corollary 2.17. *Let* Ch *be a channel protocol and* App *an application protocol w.r.t. a ground set Sec of terms. If* $(\mathsf{Ch}, \mathsf{App}, Sec)$ *is vertical composable, and* $\mathsf{Ch}^\sharp$ *and* $\mathsf{Ch}^\star \parallel \mathsf{App}$ *are both secure in isolation, then the composition* $\frac{\mathsf{App}}{\mathsf{Ch}}$ *is also secure.*

To summarize, in order to prove the security of $\frac{\mathsf{App}}{\mathsf{Ch}}$ w.r.t. a ground set *Sec* of terms, one has first to prove that $(\mathsf{Ch}, \mathsf{App}, Sec)$ is vertical composable (Definition 2.15). This means that one has to prove $(\mathsf{Ch}, \mathsf{App}, Sec)$ is parallel composable (Definition 2.9) and $\mathsf{Ch} \parallel \mathsf{App}$ is type-flaw resistant (Definition 2.4). Then, one has to make sure that all the terms from $GSMP_{\mathsf{App}}$ are shared secrets or public terms, and that none of the keys used in the channel or their subterms are labeled App, to avoid them being abstracted. Finally, one has to check that $GSMP_{\mathsf{App}}$ and $GSMP_{\mathsf{Ch}^\sharp}$ only shares public terms. All these requirements are syntactical conditions. Provided that $\mathsf{Ch}^\sharp$ and $\mathsf{Ch}^\star \parallel \mathsf{App}$ are secure in isolation, one can conclude with Corollary 2.17. We show how to apply the results to the protocols from our main examples in Section 2.6.

2.4 Proofs

We give in this section the proofs of our results. We first introduce a new notation. Each constraint can be seen as a sequence of blocks with each block being an application of a transaction rule. We sometimes write a block between $\ulcorner\urcorner$. For $n > 0$, we write $\mathcal{A}(n)$ when we consider the n-th block or $\mathcal{A}(1, n)$ when we consider the n first blocks of the constraint. We adopt the following convention for $n = 0$: $\mathcal{A}(0)$ and $\mathcal{A}(1, 0)$ are the empty constraints. Given a constraint $\mathcal{A}(1, n)$, we define $M(\mathcal{A}(1, n)) = \{m \mid \xleftarrow{m} \text{ occurs in } \mathcal{A}(1, n)\}$ as the intruder knowledge until the n-th block of the constraint $\mathcal{A}$. We use later the same notations for traces, i.e., $tr(1, n)$ and $tr(n)$. We designed in Section 2.3 a transformation from a channel protocol Ch to the protocol $\mathsf{Ch}^\sharp$. To prove the main theorem of this work, we first want to introduce an intermediary transformation. In order to lower the complexity of verifying the protocol $\mathsf{Ch} \parallel \mathsf{App}^\star$, we want to reduce the problem of solving an intruder constraint representing a protocol execution of $\mathsf{Ch} \parallel \mathsf{App}^\star$ containing set operations coming from the idealization of the application protocol $\mathsf{App}^\star$ to solving an intruder constraint

without these set operations — namely set operations dealing with the outbox and inbox sets. We call the protocol that we obtain at the end of this transformation an instantiated channel and we denote it by $\mathsf{Ch}^{\mathsf{App}}$.

Definition 2.18 (Transformation of rules of Ch to rules of $\mathsf{Ch}^{\mathsf{App}}$)**.** *Given a pure channel rule* Ch_i*, its translation into an instantiated channel rule is given as follows. If the rule contains a step of the form* $(X \leftarrow \mathsf{outbox}(A, B))$*, where* X *is a payload variable of type* $\mathfrak{p}$*, it is split into a rule for every* $t \in GSMP_{\mathsf{App}}$ *s.t.* $\mathsf{Ch}_{i,t}^{\mathsf{App}} = \mathsf{Ch}_i[X \mapsto t]$ *where the payload is instantiated with a ground subterm from the application. All other steps with a set operation for a set of the set families* inbox *or* outbox *are dropped.*

Note that if Ch_i is a well-formed rule, then $\mathsf{Ch}_{i,t}^{\mathsf{App}}$, for every $t \in GSMP_{\mathsf{App}}$, are also well-formed rules. Indeed, instead of retrieving a variable from an outbox set, we instantiate it with a ground term from $GSMP_{\mathsf{App}}$. By Definition 2.15, a channel protocol can only retrieve from an outbox set, no other set operation is allowed on this set family. Similarly, a channel protocol can only insert into an inbox set, no other set operation is allowed on this set family. We now state a soundness theorem for this transformation.

Theorem 2.19. *Let* $\mathcal{A}$ *be a constraint of the protocol* $\mathsf{Ch} \parallel \mathsf{App}^\star$ *and* $\mathcal{I}$ *an interpretation s.t.* $\mathcal{I} \models \mathcal{A}$*. Then, there exists a constraint* $\mathcal{A}'$ *of the protocol* $\mathsf{Ch}^{\mathsf{App}}$ *s.t.* $\mathcal{I} \models \mathcal{A}'$*. Furthermore, if there is an attack against* $\mathsf{Ch} \parallel \mathsf{App}^\star$ *then there is an attack against* $\mathsf{Ch}^{\mathsf{App}}$*.*

Proof. Let $\mathcal{A}$ be a constraint of $\mathsf{Ch} \parallel \mathsf{App}^\star$ and $\mathcal{I}$ an interpretation s.t. $\mathcal{I} \models \mathcal{A}$. We define the following translation and then prove it is a constraint of $\mathsf{Ch}^{\mathsf{App}}$. The translation of the constraint $\mathcal{A}$, that we denote $\mathcal{A}'$, is obtained by the following operations:

- for every block $\mathfrak{b}$ being an application of an $\mathsf{App}^\star$ rule, drop the block $\mathfrak{b}$,
- for every step $(X \leftarrow \mathsf{outbox}(A, B))$, instantiate X with $\mathcal{I}(X)$ in the whole constraint, and
- for every step $\mathfrak{s}$ where a set of the set family outbox or inbox occurs, drop the step $\mathfrak{s}$ (but not the entire block to which this step belongs to).

We show that $\mathcal{A}'$ is a constraint of the protocol $\mathsf{Ch}^{\mathsf{App}}$ defined in Definition 2.18 and that $\mathcal{I} \models \mathcal{A}'$. We proceed by induction where the induction hypothesis $\mathcal{H}(n)$ is concerned with the first n blocks of the constraints $\mathcal{A}(1, n)$ and $\mathcal{A}'(1, n)$:

(a) $\mathcal{A}'(1, n)$ is a valid constraint of the protocol $\mathsf{Ch}^{\mathsf{App}}$,

(b) the knowledge of the intruder is the same in both constraints, i.e. $M(\mathcal{I}(\mathcal{A}(1, n))) = M(\mathcal{I}(\mathcal{A}'(1, n)))$,

(c) the state of the sets, except the set from the set families outbox and inbox and sets only accessed from $\mathsf{App}^\star$ are the same in both traces, and

(d) $\mathcal{I} \models \mathcal{A}'(1, n)$

For $n = 0$, i.e., the empty constraints, it is obvious. Let us now assume that $\mathcal{H}(n)$ holds for every blocks until $n \geq 0$, let us prove it holds also for $n + 1$ blocks. We have to distinguish if the block $n + 1$ is the application of a Ch or an $\mathsf{App}^\star$ transaction.

First, consider the case when the block $n+1$ is the application of an $\mathsf{App}^\star$ rule. Then, it is dropped from the constraint, so $\mathcal{A}'(1, n+1) = \mathcal{A}'(1, n)$. By induction hypothesis, $\mathcal{I} \models \mathcal{A}'(1, n)$ so $\mathcal{I} \models \mathcal{A}'(1, n + 1)$ (d). Since the only set operations allowed in $\mathsf{App}^\star$ are set operations involving sets from the set families inbox and outbox or sets only accessed by the application, the requirement on the state of sets holds (c). There are no sent messages in $\mathsf{App}^\star$, thus we also have that $M(\mathcal{I}(\mathcal{A}(1, n + 1))) = M(\mathcal{I}(\mathcal{A}'(1, n + 1)))$ (b). Also, by induction hypothesis $\mathcal{A}'(1, n)$ is a valid constraint of $\mathsf{Ch}^{\mathsf{App}}$ and thus so is $\mathcal{A}'(1, n + 1)$ (a).

Second, consider the case when the block $n + 1$ is an application of a Ch rule. If the block $n + 1$ contains a step $(X \leftarrow \mathsf{outbox}(A, B))$, since by induction hypothesis $\mathcal{I} \models \mathcal{A}(1, n)$ and $\mathcal{I}(X) \in GSMP_{\mathsf{App}}$, it is possible to instantiate X with $\mathcal{I}(X)$ in the whole constraint. Then, all the steps where a set of the set family inbox or outbox occur are dropped. Since by induction hypothesis, the constraint until now did not contain any of these sets, removing these steps does not affect the satisfiability of the constraint, i.e. $\mathcal{I} \models \mathcal{A}'(1, n + 1)$ (d). Following this argument, the knowledge of the intruder remains the same after the translation, so $M(\mathcal{I}(\mathcal{A}(n + 1))) = M(\mathcal{I}(\mathcal{A}'(n + 1)))$ (b) and besides sets of the set families inbox and outbox, the state of sets remains the same (c). Also, during the translation of this Ch block, we instantiated X with a ground term from $GSMP_{\mathsf{App}}$ and remove the sets from the set families inbox and outbox, so we obtain a valid block of a constraint of $\mathsf{Ch}^{\mathsf{App}}$. Thus $\mathcal{A}'(1, n + 1)$ is a valid constraint of $\mathsf{Ch}^{\mathsf{App}}$ (a).

By induction we proved that there exists a constraint $\mathcal{A}'$ of the protocol $\mathsf{Ch}^{\mathsf{App}}$ s.t. $\mathcal{I} \models \mathcal{A}'$. It entails that if there is an attack against $\mathsf{Ch} \parallel \mathsf{App}^\star$, there is an attack against $\mathsf{Ch}^{\mathsf{App}}$. □

We are now ready to take it to the level of the protocol $\mathsf{Ch}^\sharp$. We want to define a ground trace of $\mathsf{Ch}^\sharp$ from a translation of a ground trace of $\mathsf{Ch}^{\mathsf{App}}$. For that

purpose, we define an *abstraction function* denoted g that takes terms from $GSMP^{\bullet}_{\mathsf{App}} = GSMP_{\mathsf{App}} \setminus \{t \mid \emptyset \vdash t\}$ — namely the terms that are labeled App — and abstract from them by replacing them by a fresh constant g from $\mathfrak{G}$ — the infinite set of constants that we defined in Section 2.3.1. This function leaves unchanged the terms labeled Ch or *Pub*:

Definition 2.20 (g function)**.** *Let g be an injective function from $GSMP^{\bullet}_{\mathsf{App}}$ to $\mathfrak{G}$ (i.e., $\forall s,t \in GSMP^{\bullet}_{\mathsf{App}}.\ g(s) = g(t) \Rightarrow s = t$). We extend g to a function from $\mathcal{T}_\Sigma \to \mathcal{T}_\Sigma$ by setting $g(f(t_1,\ldots,t_n)) = f(g(t_1),\ldots,g(t_n))$ whenever $f(t_1,\ldots,t_n) \notin GSMP^{\bullet}_{\mathsf{App}}$.*

If we apply this function to all steps of a ground trace of $\mathsf{Ch}^{\mathsf{App}}$, we can abstract from the terms introduced by the application. We use this function to defined a ground trace tr' that we later prove to be a valid trace of $\mathsf{Ch}^{\sharp}$:

Definition 2.21 (Translated trace tr')**.** *We define the meta function* status *on the abstract constants of a trace tr':*

$$\mathsf{status}(g, tr'(1,n)) = \begin{cases} (g \mathrel{\dot\in} \mathsf{opened}) & \text{if } (g \to \mathsf{opened}) \in tr'(1,n) \\ (g \mathrel{\dot\in} \mathsf{closed}) & \text{otherwise} \end{cases}$$

For a given ground trace tr of $\mathsf{Ch}^{\mathsf{App}}$ and $n >= 1$, we define the translated ground trace tr' by:

$$tr'(0) = \{\ulcorner g \to \mathsf{closed} \urcorner \mid g \in g(GSMP(M(tr))) \cap \mathfrak{G}\}$$
$$tr'(n) = \ulcorner \{\mathsf{status}(g, tr'(a, n-1)) \mid g \in g(tr(n)) \cap \mathfrak{G}\}.g(tr(n)) \urcorner$$

Besides, if $g \in declassified_{\mathcal{DY}}(tr'(n)) \cap \mathfrak{G}$, i.e., g is declassified in the n-th block in a step $(\star\colon \xleftarrow{g})$, and $(g \mathrel{\dot\in} \mathsf{closed}) \in tr'(n)$, then these two steps are replaced by $(g \leftarrow \mathsf{closed}.g \to \mathsf{opened}.\star\colon \xleftarrow{g})$.

We now show that the declassified terms of a ground trace of $\mathsf{Ch}^{\sharp}$ are just the abstraction of the declassified terms of the original ground trace of $\mathsf{Ch}^{\mathsf{App}}$:

Lemma 2.22. *The declassified* Payload *messages of the translated trace coincides with the ones of the original trace modulo g, i.e., $g(declassified_{\mathcal{DY}}(tr(1,n) \cap GSMP^{\bullet}_{\mathsf{App}}) = declassified_{\mathcal{DY}}(tr'(0,n))) \cap \mathfrak{G}$.*

Proof. Let $g \in g(declassified_{\mathcal{DY}}(tr(1,n)) \cap GSMP^{\bullet}_{\mathsf{App}})$. By Definition 2.20, $g \in \mathfrak{G}$. If $g \in \mathsf{closed}$, then it is going to be declassified and inserted in opened during the translation of original trace as defined in Definition 2.21 and then $g \in declassified_{\mathcal{DY}}(tr'(0,n))$. If $g \in \mathsf{opened}$, then it means it has

been declassified before because abstraction constants can only be inserted in an opened during declassification and then again $g \in \mathit{declassified}_{\mathcal{DY}}(tr'(0,n))$. Thus, $g \in \mathit{declassified}_{\mathcal{DY}}(tr'(0,n)) \cap \mathfrak{G}$.

Let $M = \mathit{declassified}_{\mathcal{DY}}(tr(1,n))$ and $M' = \mathit{declassified}_{\mathcal{DY}}(tr'(0,n))$, i.e., the declassified messages of each trace without restriction to payloads, for the other direction. First, observe that for every $s \in M'$ there is a t with $M \vdash t$ and $g(t) = s$; this is because M' contains only messages that are the translation $g(t)$ of a message t declassified in tr, or that have been opened, i.e., $t \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n)) \cap \mathit{GSMP}^{\bullet}_{\mathsf{App}}$. Let $M' \vdash s$, then there is a corresponding derivation $M \vdash t$ with $g(t) = s$, because we can replace every constant from $\mathfrak{G}$ in the proof $M' \vdash s$ with the corresponding term from M. Thus $\mathit{declassified}_{\mathcal{DY}}(tr'(0,n)) \subseteq g(\mathit{declassified}_{\mathcal{DY}}(tr(1,n)))$.

We thus proved that the declassified Payload messages of the translated trace coincides with the ones of the original trace modulo g, i.e., $\mathit{declassified}_{\mathcal{DY}}(tr'(0,n)) \cap \mathfrak{G} = g(\mathit{declassified}_{\mathcal{DY}}(tr(1,n)) \cap \mathit{GSMP}^{\bullet}_{\mathsf{App}})$. □

Lemma 2.23. *Let tr be a ground trace from $\mathsf{Ch}^{\mathsf{App}}$ of length at least $n+1$. Let $M^+ = M(tr(1,n+1))$ and $M^\star = M(tr'(0,n+1))$. If $g(\mathcal{DY}(M(tr(1,n)))) \subseteq \mathcal{DY}(g(M(tr'(0,n))))$ then also $g(\mathcal{DY}(M^+)) \subseteq \mathcal{DY}(M^\star)$ or there exists $g \in \mathcal{DY}(M^\star)$ s.t. $(g \to \mathsf{closed})$ occurs in $tr'(0,n+1)$ and not $(g \to \mathsf{opened})$.*

Proof. Let $tr, tr', n, M^+, M^\star$ given as in the statement. Let us say that $tr(1,n+1)$ *leaks payload* if there is a message $t \in \mathit{GSMP}^{\bullet}_{App} \setminus \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1))$ such that $M^+ \vdash t$, and similarly, say that $tr'(0,n+1)$ *leaks payload* if there is a message $g \in \mathfrak{G} \setminus \mathit{declassified}_{\mathcal{DY}}(tr'(0,n+1))$ such that $M^\star \vdash g$. If $tr'(0,n+1)$ leaks payload, then this lemma holds, because $M^\star \vdash g$ for some $g \in \mathfrak{G}$ (so $g \to$ closed occurs in the trace) and $g \notin \mathit{declassified}_{\mathcal{DY}}(tr'(0,n+1))$ (so $g \to$ opened does not). Thus, for the remainder of this proof we can assume that $tr'(0,n+1)$ does not leak payload.

Note that, if $g \in \mathit{declassified}_{\mathcal{DY}}(tr'(1,n+1)) \cap \mathfrak{G}$, then by Lemma 2.22, there exists a $t \in \mathit{declassified}_{\mathcal{DY}}(tr(0,n+1)) \cap \mathit{GSMP}^{\bullet}_{\mathsf{App}}$ such that $g(t) = g$.

We proceed by structural induction over the derivation $M^+ \vdash t$ (see Definition 2.1). Our induction hypothesis (for $m \in \mathbb{N}$) is: $\varphi(m) \equiv \forall t.\ M^+ \vdash^m t \implies M^\star \vdash g(t)$ where $\vdash^m$ denotes the derivation in at most m steps.

The initial case $\varphi(0)$ coincides with the (Axiom) case: $\dfrac{}{M^+ \vdash t}$, $t \in M^+$. By definition of the translated trace tr', $g(t) \in M^\star$ thus $M^\star \vdash t$.

For the induction step $\varphi(m) \implies \varphi(m+1)$: we have either a composition or a

decomposition step.

For the (Compose) derivation, we have that $t = f(t_1, \ldots, t_p)$ for some $f \in \Sigma^p_{pub}$ and $\dfrac{M^+ \vdash^m t_1 \quad \cdots \quad M^+ \vdash^m t_p}{M^+ \vdash^{m+1} f(t_1, \ldots, t_p)}$. By induction, we have that $M^\star \vdash g(t_1)$,...,$M^\star \vdash g(t_p)$. We further distinguish two cases:

1. $f(t_1, \ldots, t_p) \in GSMP^\bullet_{\mathsf{App}}$: we have $g(f(t_1, \ldots, t_p)) \in \mathfrak{G}$, and $t_i \in GSMP_{\mathsf{App}}$ for $1 \leq i \leq p$. For each $1 \leq i \leq p$, we have either $t_i \in \{t \mid \emptyset \vdash t\}$, then $g(t_i) = t_i$, otherwise $g(t_i) \in \mathfrak{G}$. In that case, $g(t_i) \in \mathit{declassified}_{\mathcal{DY}}(tr'(0, n{+}1)) \cap \mathfrak{G}$ since $tr'(0, n{+}1)$ does not leak, and thus $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n{+}1)) \cap GSMP^\bullet_{\mathsf{App}}$ by Lemma 2.22. Thus, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n{+}1))$ for all $1 \leq i \leq p$ (including public t_i). Thus, by $\mathcal{DY}$-closure also $f(t_1, \ldots, t_p) \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1))$, and since also $f(t_1, \ldots, t_p) \in GSMP^\bullet_{\mathsf{App}}$, again by Lemma 2.22, we have $g(f(t_1, \ldots, t_p)) \in \mathit{declassified}_{\mathcal{DY}}(tr'(1, n+1)) \cap \mathfrak{G}$ and thus $g(f(t_1, \ldots, t_p)) \in M^\star$ by the construction of tr'. We thus have $M^\star \vdash g(f(t_1, \ldots, t_p))$ and therefore $\varphi(m+1)$ holds.
2. $f(t_1, \ldots, t_p) \notin GSMP^\bullet_{\mathsf{App}}$: then by definition of g, $g(f(t_1, \ldots, t_p)) = f(g(t_1), \ldots, g(t_p))$. Since $M^\star \vdash g(t_i)$ by induction, also $M^\star \vdash f(g(t_1), \ldots, g(t_p))$ and thus $\varphi(m+1)$ holds.

(Decompose): then there is $t_0 = f(t_1, \ldots, t_q)$ such that $t \in \{t_1, \ldots, t_q\}$ and

$$\frac{M^+ \vdash^m t_0 \quad M^+ \vdash^m k_1 \quad \ldots \quad M^+ \vdash^m k_p}{M^+ \vdash^{m+1} t} \quad \begin{array}{l}\mathsf{Ana}(t_0) = (\{k_1, \ldots, k_p\}, \{t\} \cup T),\\ p \leq q\end{array}$$

By the form of Ana rules, $\{k_1, \ldots, k_p\} \subseteq \{t_1, \ldots, t_q\}$. W.l.o.g. we can assume that the keys are the first p positions of f, i.e. $t_1 = k_1, \ldots, t_p = k_p$. By induction, we have that $M^\star \vdash g(t_0), M^\star \vdash g(k_1), \ldots, M^\star \vdash g(k_p)$. To show: $M^\star \vdash g(t)$. We distinguish further two the cases:

1. $t_0 \in GSMP^\bullet_{\mathsf{App}}$: we have that $g(t_0) \in \mathfrak{G}$, and $t, t_1, \ldots, t_q \in GSMP_{\mathsf{App}}$. For each $0 \leq i \leq q$, we have either $t_i \in \{t \mid \emptyset \vdash t\}$, then $g(t_i) = t_i$, otherwise $t_i \in GSMP^\bullet_{\mathsf{App}}$ and thus $g(t_i) \in \mathfrak{G}$. In that case, $g(t_i) \in \mathit{declassified}_{\mathcal{DY}}(tr'(0, n+1)) \cap \mathfrak{G}$, since $tr'(1, n+1)$ does not leak, and thus by Lemma 2.22, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1)) \cap GSMP^\bullet_{\mathsf{App}}$. Thus, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1))$ for all $0 \leq i \leq q$ (including public t_i). Thus by $\mathcal{DY}$, also $t \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1))$. If $t \in \{t \mid \emptyset \vdash t\}$, then trivially $M^\star \vdash g(t)$, otherwise since $t \in GSMP_{\mathsf{App}}$, we have $t \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1)) \cap GSMP^\star_{\mathsf{App}}$, and thus again by Lemma 2.22, $g(t) \in \mathit{declassified}_{\mathcal{DY}}(tr'(0, n+1)) \cap \mathfrak{G}$, and thus $g(t) \in M^\star$ by construction. Therefore $M^\star \vdash g(t)$ and therefore $\varphi(m+1)$ holds.

2. $t_0 \notin GSMP^{\bullet}_{\mathsf{App}}$: excluding the trivial case $t_0 \in \{t \mid \emptyset \vdash t\}$, t_0 is thus labeled channel and thus by our assumptions so are also the keys $t_1, \ldots, t_p$, i.e., they cannot be part of $GSMP^{\bullet}_{\mathsf{App}}$ either. Thus, $g(t_0) = g(f(t_1, \ldots, t_q)) = f(g(t_1), \ldots, g(t_q)) = f(t_1, \ldots, t_p, g(t_{p+1}), \ldots, g(t_q))$. By induction, we have that $M^\star \vdash g(t_0)$ and $M^\star \vdash g(t_i) = t_i$ for $1 \leq i \leq q$. Thus the corresponding analysis step is possible in $M^\star$, yielding $M^\star \vdash g(t)$.

□

Theorem 2.16. *Let* Ch *be a channel protocol and* App *an application protocol w.r.t. a ground set Sec of terms that are vertical composable. If there is an attack in* $\mathsf{Ch} \parallel \mathsf{App}^\star$*, then there is one in the protocol* $\mathsf{Ch}^\sharp$*.*

Proof. Let us consider a constraint $\mathcal{A}$ of $\mathsf{Ch} \parallel \mathsf{App}^\star$ and an interpretation $\mathcal{I}$ s.t. $\mathcal{I} \models \mathcal{A}$. By Theorem 2.19, there exists a constraint $\mathcal{A}'$ of $\mathsf{Ch}^{\mathsf{App}}$ s.t. $\mathcal{I} \models \mathcal{A}'$. $\mathcal{I}(\mathcal{A}')$ is a ground trace of $\mathsf{Ch}^{\mathsf{App}}$. We note it tr and we consider its translation following Definition 2.21.

We proceed now by induction, where the induction hypothesis $\mathcal{H}(n)$ is concerned with the first n blocks of the original trace $tr(1, n)$ and the $n+1$ blocks of steps of the translated trace $tr'(0, n)$ defined in Definition 2.21:

- either $tr'(0, n)$ is a valid trace of $\mathsf{Ch}^\sharp$, and we have that $g(\mathcal{DY}(M(tr(1, n)))) \subseteq \mathcal{DY}(M(tr'(0, n)))$,
- either $\mathcal{DY}(M(tr'(0, n))) \cap (g(Sec \backslash declassified_{\mathcal{DY}}(tr(1, n))) \cup \{\mathsf{attack}_{\mathsf{Ch}}\}) \neq \emptyset$

The second conjunction holds for $n = 0$. We show that $tr'(0, 0)$ is a valid trace of $\mathsf{Ch}^\sharp$. It was defined in Definition 2.21 that initially a number of g-values are inserted in the closed set. These steps can be generated by the rule $\mathsf{Ch}^\sharp_{\mathsf{new}}$. There is initially no declassified values.

Suppose the induction hypothesis holds for some number $n \geq 0$, and the number of blocks of steps in both traces is at least $n + 1$. Note that once the second disjunction is true for n, it is also true for all $n' > n$. Thus we suppose the second disjunction does not hold until n. We start by showing that the translation of every new block is the application of a valid rule of $\mathsf{Ch}^\sharp$. Note that as specified in Definition 2.21, all the constants $g \in \mathfrak{G}$ occurring in $tr'(0, n + 1)$ have been inserted in the set closed at the start of the trace. The function status only inserts positive checks at the beginning of the blocks, as in every rule of $\mathsf{Ch}^\sharp$.

There are already no outbox or inbox in $\mathsf{Ch}^{\mathsf{App}}$. We then apply the function g to the block that replace every ground term from the application, that replaced payload variables, by an abstract constant from $\mathfrak{G}$. We also specify how to correctly declassify the constants from $\mathfrak{G}$. Thus we obtain a valid application of a rule of $\mathsf{Ch}^{\sharp}$. Then, we can now show that the induction hypothesis holds for $n+1$. We distinguish the following cases according to the kind of blocks of steps that we are concerned with at the block $n+1$. In the following, we consider that the second disjunction is not true until n, otherwise the induction is trivially true as we explained earlier.

- new messages are received but not the constant $\mathsf{attack}_{\mathsf{Ch}}$: it means the knowledge of the intruder is augmented by the set of new received messages, i.e. $M(tr(1,n+1)) = M(tr(1,n)) \cup M(tr(n+1))$. By induction hypothesis, we can apply Lemma 2.23 and we have $g(\mathcal{DY}(M(tr(1,n+1)))) \subseteq \mathcal{DY}(g(M(tr'(0,n+1))))$ or there exists $g \in \mathcal{DY}(g(M(tr(1,n+1))))$ s.t. $(g \mathrel{\dot\in} \mathsf{closed})$ occurs in $tr'(0,n+1)$. Therefore, either of the disjunction of the induction hypothesis holds and $\mathcal{H}(n+1)$ holds.

- no new messages are received: the knowledge of the intruder stays the same, i.e. $M(tr(1,n+1)) = M(tr(1,n))$. We can use the induction hypothesis and apply Lemma 2.23, either of the disjunction holds and $\mathcal{H}(n+1)$ holds.

- the constant $\mathsf{attack}_{\mathsf{Ch}}$ is received in the original trace: as explained in Definition 2.20, the constant $\mathsf{attack}_{\mathsf{Ch}}$ is not abstracted. This means the constant $\mathsf{attack}_{\mathsf{Ch}}$ is also received in the translated trace. Therefore the second disjunction holds in the block $n+1$ and $\mathcal{H}(n+1)$.

By induction, we proved the theorem. □

Finally, the composition of vertical composable and secure application and channel protocols is secure:

Corollary 2.17. *Let* Ch *be a channel protocol and* App *an application protocol w.r.t. a ground set Sec of terms. If* $(\mathsf{Ch}, \mathsf{App}, Sec)$ *is vertical composable, and* $\mathsf{Ch}^{\sharp}$ *and* $\mathsf{Ch}^{\star} \parallel \mathsf{App}$ *are both secure in isolation, then the composition* $\frac{\mathsf{App}}{\mathsf{Ch}}$ *is also secure.*

Proof. This is a direct consequence of Theorem 2.16 and Theorem 2.10. □

2.5 Extension of the typing results

2.5.1 Extension of the Typing Result [Hess18]

We define a compatibility relation as the least reflexive and symmetric relation such that:

- $\tau_1 \bowtie \tau_2$ if $\tau_1 \leq \tau_2$, and
- $f(\tau_1, \ldots, \tau_n) \bowtie f(\tau'_1, \ldots, \tau'_n)$ if $\tau_1 \bowtie \tau'_1 \wedge \ldots \wedge \tau_n \bowtie \tau'_n$

Note that $\bowtie$ is not transitive, e.g. $\mathfrak{p} \bowtie f_3(\mathsf{Nonce})$ and $f_1(\mathsf{Nonce}, \mathsf{Agent}) \bowtie \mathfrak{p}$, but $f_3(\mathsf{Nonce}) \not\bowtie f_1(\mathsf{Nonce}, \mathsf{Agent})$.

We silently assume in the following that all substitutions are idempotent, i.e. variables of the domain do not occur in the image. Note that all unifiers of terms can be made idempotent.

As it is standard we define the composition $\theta \circ \sigma$ of two substitutions σ and θ as function composition. Note that the result is in general not idempotent (e.g. $\sigma = [x \mapsto f(y)]$, $\theta = [y \mapsto f(x)]$), therefore, we silently assume in the following that $fv(img(\theta)) \cap dom(\sigma) = \emptyset$ (which is the case for all constructions we make in this chapter).

Lemma 2.24. *Given well-typed σ and θ, then $\theta \circ \sigma$ is well-typed.*

Proof. Consider any variable x, we have to show $\Gamma(x) \geq \Gamma(\sigma(x)) \geq \Gamma(\theta(\sigma(x)))$.

By well-typedness of σ follows already $\Gamma(x) \geq \Gamma(\sigma(x))$. If $\sigma(x)$ is a variable, then also $\Gamma(\sigma(x)) \geq \Gamma(\theta(\sigma(x)))$ follows by well-typedness of θ. If $\sigma(x)$ is not a variable, no proper subterm of $\sigma(x)$ can have type $\mathfrak{p}$ or a type from $\mathfrak{T}_\mathfrak{p}$ (because otherwise $\Gamma(x)$ would contain $\mathfrak{p}$ or $\mathfrak{T}_\mathfrak{p}$ as a proper subterm). Thus $\Gamma(y) = \Gamma(\theta(y))$ for every variable y in $\sigma(x)$, and thus $\Gamma(\sigma(x)) \leq \Gamma(\theta(\sigma(x)))$. □

Lemma 2.25. *Let s, t be terms such that $\Gamma(s) \bowtie \Gamma(t)$, and $\theta = [x \mapsto u]$ with $\Gamma(x) = \mathfrak{p}$ and $\Gamma(u) \in \mathfrak{T}_\mathfrak{p}$ such that $\theta(s)$ and $\theta(t)$ can be unified. Then $\Gamma(\theta(s)) \bowtie \Gamma(\theta(t))$.*

Proof. Consider $P = pos(s) \cap pos(t)$, the set of positions that exist in both s and t. Note that $\Gamma(s|p) \bowtie \Gamma(t|p)$ for all $p \in P$. Consider any position p where x occurs in s or t, say in s. We consider two cases:

- $p \in P$. Since $\Gamma(s|p) \bowtie \Gamma(t|p)$, $t|p$ is:
  - either a variable of type $\mathfrak{p}$ (thus $\Gamma(\theta(s|p)) \bowtie \Gamma(\theta(t|p))$)
  - or a composed term of a type in $\mathfrak{T}_\mathfrak{p}$. Since $\theta(s|p)$ can be unified with $\theta(t|p)$, follows $\Gamma(t|p) = \Gamma(u)$ (thus $\Gamma(\theta(s|p)) = \Gamma(\theta(t|p))$).
- $p \notin P$. Let p_0 be the longest prefix of p with $p_0 \in P$. Since $\Gamma(s|p_0) \bowtie \Gamma(t|p_0)$, $t|p_0$ must be variable (otherwise a strictly longer prefix of p would be in P). However, that also means $\Gamma(t|p_0)$ contains as a subterm either $\mathfrak{p}$ or an element of $\mathfrak{T}_\mathfrak{p}$ — that contradicts the requirements on the typing system.

For all other positions $p \in P$ (including $p = \epsilon$), it follows $\Gamma(\theta(s|p)) \bowtie \Gamma(\theta(t|p))$ from the definition of $\leq$. □

Theorem 2.2. *Let s, t be unifiable terms with $\Gamma(s) \geq \Gamma(t)$. Then their most general unifier is well-typed.*

Proof. We show an invariant for the standard unification algorithm where a state of the algorithm is characterized by a set of pairs of terms $\{(s_1, t_1), \ldots, (s_n, t_n)\}$ to unify (initially this set consists of the given pair $\{(s, t)\}$) and a current substitution σ (initially the identity). The invariant is that $\Gamma(s_i) \bowtie \Gamma(t_i)$ and σ is well-typed. We exploit during the proof the fact that the given s and t are unifiable, and the standard unification algorithm is correct, i.e., it will return a unifier σ_f that is most general, and at each state, the current σ will have σ_f as an instance, and the (s_i, t_i) pairs all have σ_f as a unifier.

The algorithm picks any pair (s_i, t_i) to unify first and we distinguish the cases:

- $\Gamma(s_i) = \mathfrak{p}$. Then s_i is a variable (since $\mathfrak{p}$ is not an atomic or composed type). Since $\Gamma(s_i) \bowtie \Gamma(t_i)$, we have one of the following two cases:
  - $\Gamma(t_i) = \mathfrak{p}$, and hence t_i is a variable. Then $\theta = [s_i \mapsto t_i]$ (or $\theta = [t_i \mapsto s_i]$, depending on the algorithm's preference) is well-typed, thus $\theta \circ \sigma$ is also well-typed by Lemma 2.24. Moreover, θ does not change the type of any term it is applied to, i.e. $\Gamma(\theta(s_j)) = \Gamma(s_i) \bowtie \Gamma(t_j) = \Gamma(\theta(t_j))$.
  - $\Gamma(t_i) \in \mathfrak{T}_\mathfrak{p}$, and hence t_i is not a variable. Thus $\theta = [s_i \mapsto t_i]$ is well-typed since $\Gamma(s_i) > \Gamma(t_i)$. Moreover, $\theta \circ \sigma$ is well-typed by Lemma 2.24. By Lemma 2.25, we have that $\Gamma(\theta(s_j)) \bowtie \Gamma(\theta(t_j))$ for the all pairs to unify.

This takes care of the case that one of the terms to unify is a variable of type $\mathfrak{p}$.

- If s_i is a variable, but not of type $\mathfrak{p}$, then it cannot contain $\mathfrak{p}$ or any element of $\mathfrak{T}_{\mathfrak{p}}$ as subterm of the type. Thus actually $\Gamma(s_i) = \Gamma(t_i)$ and the substitution $\theta = [s_i \mapsto t_i]$ does not change any types and thus preserves the invariant. The case that t_i is a variable is handled symmetrically.
- Otherwise we have $s_i = f(u_1, \ldots, u_k)$ and $t_i = f(v_1, \ldots, v_k)$ for some operator f and $\Gamma(u_i) \bowtie \Gamma(v_i)$ by construction, thus also here the invariant is preserved recursively.

□

2.5.2 Extending the Results of [Hess18]

We now describe how the definition and results from [Hes19] can be extended to the payload data type that we have introduced. We just summarize definitions and proofs that do not require any changes, and we omit aspects that we do not need for our result.

The thesis [Hes19] develops the typing result in several stages, namely in Sec. 3.2 first on intruder constraints without analysis (so the intruder can only compose and unify) and without set operations and conditions; this is extended in Sec. 3.3 to transition systems (where analysis is also handled), and then in Sec. 4, this is extended to stateful constraints (augmenting with set operations and conditions). The reason is that Sec. 3.3 and Sec. 4 are done by reduction to problems of Sec. 3.2; since all the extensions on all levels are similar, we will sometimes group this together for the different levels.

Note that the typing result we have formalized so far is a conservative extension of the typing system in [Hes19] (Sec. 3.2) in the sense that our system allows strictly more types, considers strictly more substitutions as well-typed and leads to a strictly larger *SMP* for a given protocol (since it is closed under well-typed substitutions).

Our notion of type-flaw resistance is thus more liberal than [Hes19] (Def. 3.17 and 4.12) except for the requirements on inequalities: here [Hes19] gives two possible ways to satisfy the requirements: either all free variables of the inequality are of atomic type or no subterm of the inequality is generic. We have opted to specify only the second choice since we never practically needed the

first and wanted to make the notion of type-flaw resistance not unnecessarily complicated — the result holds however even when allowing both choices.

In section [Hes19], a sound, complete and terminating procedure for constraint reduction is introduced. This procedure is applied to a pair $(\mathcal{A}, \theta)$ where $\mathcal{A}$ is a well-formed constraint and θ a substitution for variables that have been already instantiated (so $dom(\theta) \cap fv(\mathcal{A}) = \emptyset$); initially θ is the identity. This procedure is unaffected by our extensions. The core of the typing result is the following lemma:

Lemma 2.26 ([Hes19] (Lemma 3.18))**.** *If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is type-flaw resistant, θ is well-typed, and from $(\mathcal{A}, \theta)$ the constraint reduction can reach $(\mathcal{A}', \theta)$, then $\mathcal{A}'$ is type-flaw resistant and θ' is well-typed.*

Proof. The constraint reduction can do any of the following steps:

- it can unify a received term s and a term t that the intruder has sent. Neither s nor t may be a variable (but may contain variables). By type-flaw resistance and since $s, t \in SMP(\mathcal{A})$ and are not variables, if they have a unifier, then either $\Gamma(s) \leq \Gamma(t)$ or $\Gamma(s) \geq \Gamma(t)$, and thus by Theorem 2.2, their most general unifier is well-typed. Thus, the invariants are satisfied on the resulting constraint. This is in fact the main point why our extension of the typing result is correct.
- for equality we already have that the two terms must have compatible types, and thus again their most general unifier is well-typed, and
- the other two operators are composition and analysis (in [Hes19] (Sec. 3.3)), both just move to subterms or key-terms under which *SMP* is closed.

□

Thus, as far this procedure is concerned, the intruder never needs to make an ill-typed choice to launch an attack. The constraint reduction terminates with so-called *simple* constraints: all messages left to send for the intruder are variables with satisfiable inequalities. Without any inequality constraints, this is trivially satisfiable, because the intruder can pick just any value for the remaining variables. For constraints with inequalities [Hes19] (Lemma 3.7) (that is independent of the typing) tells us to pick a fresh value for every remaining variable and check the inequalities; if they are unsatisfiable, then they are unsatisfiable for every choice, otherwise we have a solution. The point is that this pick can be done also well-typed:

Lemma 2.27 ([Hes19] (Lemma 3.19))**.** *If $(\mathcal{A}, \theta)$ is well-formed, $\mathcal{A}$ is simple, and θ is well-typed, then $(\mathcal{A}, \theta)$ has a well-typed model.*

Proof. The point is that the intruder has unbounded reservoir of public constants of any atomic type; for composed types with a public function symbol, he can just apply the function symbol to fresh terms of the corresponding subtypes. Private function symbols are in fact just syntactic sugar: a private function symbol $f \in \Sigma^n$ is encoded as a public function symbol $f_p \in \Sigma^{n+1}_{\mathsf{pub}}$, where the first argument for all terms of a protocol is a special constant the intruder does not have, so he cannot compose "interesting" private terms of the protocol (like private keys), but something that satisfies the typing. For our extension with payload types, there is only one item to consider: the abstract payload type $\mathfrak{p}$. Here, we have to pick a value for some type of $\mathfrak{T}_\mathfrak{p}$. This requires that $\mathfrak{T}_\mathfrak{p} \neq \emptyset$, which is however not a restriction as we actually do want to use it with concrete payloads. □

From this follows immediately the typing result for constraints [Hes19] (Theorem 3.20) that a type-flaw resistant constraint has a solution iff it has a well-typed solution.

As part of lifting the result to the protocol level, the analysis rules integrated in [Hes19] (Sec. 3.3.3); this is relevant since the analysis rules are untyped, while we have to handle this in way compatible with the typing result. The construction is to allow analysis steps for every subterm in the intruder knowledge except variables. This is sufficient since the intruder does not have to analyze any term that does not occur as a construction in his knowledge (because all other constructions are by the intruder, so he already knows the subterms). In fact, this proof also works when we integrate payloads: either a variable of type $\mathfrak{p}$ is never instantiated (so the intruder does not have to analyze it) or it is instantiated with a more concrete term that already occurs in the constraint (then it is already covered by the construction). Thus there is in fact no modification for obtaining the result [Hes19] (Theorem 3.27) for the typing result on the protocol level: every satisfiable reachable constraint of a protocol has a well-typed solution.

In [Hes19] (Sec. 4), the result is lifted to stateful constraints by a reduction proof that maps set operations and check into equality and inequality checks. In fact, the result theorem [Hes19] (Theorem 4.15) requires only the updates to type-flaw resistance already discussed, i.e. for type-flaw resistant stateful protocols, every satisfiable reachable constraint of a protocol has a well-typed solution.

2.5.3 Update of the Parallel Composability Result

[Hes19] defines the declassified messages as those that are sent by an honest agent in a $\star$-labeled step, i.e., those that some component explicitly wishes to declassify. This mechanism was initially intended to apply only to a few distinguished secrets used in both protocols, e.g., public and private keys. The construction in this chapter, however, treats all messages of the payload and their submessages (as far as they are not public) as members of *Sec*. For instance, in a pair-style function (i.e., the intruder can both compose and decompose it), we would have for instance that the intruder also immediately knows the components, and also that he can build other messages of the protocol with these components. It is unfeasible to explicitly declassify all these messages, because the intruder may even compose well-typed messages from components that he learned in different declassification steps. Therefore, we consider in this chapter a variant of declassification that is closed under Dolev-Yao deduction.

2.5.4 Declassification (extended from [Hess18])

Let $\mathcal{A}$ be a labeled constraint and $\mathcal{I}$ a model of $\mathcal{A}$. Then $\mathit{declassified}_{\mathcal{DY}}(\mathcal{A}, \mathcal{I}) \equiv \mathcal{DY}(\{t \mid \star\colon \xleftarrow{t} \text{ occurs in } \mathcal{I}(\mathcal{A})\})$ is the set of declassified secrets of $\mathcal{A}$ under $\mathcal{I}$.

First, one may wonder if this is going to far, i.e., that this closure includes some declassifications that the designer of the protocol did not intend and is not aware of. However, this is in our opinion not the case, since the declassification does in general *not* play the role of secrecy goals, but only the one of an interface in the composition of protocols where they are guaranteeing each other not to leak. In other words, the declassification plays the role of a "contract" between two protocols, each of them should have their own policy about secrecy, and it is then part of the verification of each protocols goals, that said contract is sufficient to guarantee these goals.

As a concrete example, let us consider the famous attack on Needham-Schroeder Public-Key Protocol (NSPK) [Low95]. Suppose we have NSPK without cryptography as an application running over confidential channels. In the attack, an honest a in role A first sends a message $f_1(na, a)$ to the intruder, i.e., to a dishonest recipient. Note that we use here a format f_1 instead of pairs to ensure type-flaw resistance. This means that we declassify $f_1(na, a)$, and thus na and later also $f_2(na, nb)$ which would be a type-correct message of the second step. The intruder forwards this message on a confidential channel to an honest b in role B, who answers with $f_2(na, nb)$ on a confidential channel to a, so this is *not*

declassified. In fact, nb is now considered by NSPK as a secret between a and b, and is made explicit by putting $nb \to \mathsf{secret}(a, b)$ (with the corresponding rule $(\xleftarrow{M} .M \in \mathsf{secret}(A, B).\mathsf{attack})$ for all honest A and B). Further, a, who believes to be talking to the intruder, sends the reply $f_3(nb)$ on a confidential channel to the intruder. From a's point of view this is fine, and $f_3(nb)$ and nb get declassified as they are deliberately sent to a dishonest recipient. Now, the intruder does indeed know the declassified nb and can trigger the violation of the secrecy goal for b. This illustrates that the attack exploits a discrepancy between a's understanding of the protocol (in particular $na, nb \in \mathsf{secret}(a, i)$), and thus sending this message to the intruder in accordance with the protocol, and thereby revealing a value that b considers as a secret. However, this does not violate the contract between channel and application: in the steps where a has sent messages to a dishonest recipient, she has just released the channel from the obligation to keep these messages secret.

It seems intuitive that all proofs of the compositionality result [Hes19] still work with this modified notion of the declassification, because these proofs do not actually depend on what message the particular protocols choose to declassify; the only crucial property — that the intruder can derive all declassified messages — still holds with this definition.

However, the way the proofs are constructed in [Hes19] on the constraint level does not work with this update directly. Like in the typing result, [Hes19] (Sec. 5) first establishes all properties on a constraint level, and for all notions considers only messages that occur on the constraint level, but not all messages that may occur in a given set of protocols. In contrast, our new notion of declassification deliberately considers terms that have not yet occurred at a particular point in a constraint, but that the intruder may want to use later.

However this "scoping" issue can be overcome by the following change to a number the definitions in [Hes19]: where the scope is limited to *GSMP* of a particular trace, we replace it with the *GSMP* of the entire protocol. This is actually a substantial change to a number of definitions and lemmas, but the intuition why this works is quite simple: suppose we add to a constraint at the start that the intruder should send messages covering every message pattern occurring in the protocol (using fresh variables). This does not constrain the intruder really —since he is always able to generate messages in the form of the protocol— and the *GSMP* of the constraint would then indeed cover the *GSMP* of the entire protocol we consider. However, this is only the intuition why this change works, and we now provide proper definitions.

First, we have the definition given in the main text, i.e., the set *Sec*; the definition of *GSMP* for a set of messages, for a trace, and for a protocol; and

GSMP-disjointness. We also define $\mathit{declassified}_{\mathcal{DY}}(\mathcal{A},\mathcal{I})$ as in the main text as the Dolev-Yao closure of the $\star$-labeled messages received by the intruder in $\mathcal{I}(\mathcal{A})$. Thus, $\mathit{declassified}_{\mathcal{DY}}(\mathcal{A},\mathcal{I}) = \mathcal{DY}(\mathit{declassified}(\mathcal{A},\mathcal{I}))$ for the notion of *declassified* in [Hes19] (Def. 5.2). We define leakage with respect to our notion of declassification.

The definition of parallel composability is also changed w.r.t. [Hes19] (Def. 5.4 and 5.6) on the constraint level and on the protocol level: we omit the requirement that a transaction cannot have $\star$-labeled receive steps (i.e., sending from the intruder's point of view). In fact, the authors of [HMB20] have discovered in the meantime that the proofs can be conducted without this requirement. Another change is that we of course now use the notion of type-flaw resistance that we have defined in this chapter earlier (and the parallel compositionality only relies on the typing result itself, i.e., that a well-typed attack exists if an attack exists).

To prove the updated parallel compositionality result, let us fix a few terms for the remainder of this section: suppose $\mathcal{P}_1,\ldots,\mathcal{P}_n$ are protocols that are parallel composable with respect to a set *Sec* of secrets. Let $\mathcal{P} = \mathcal{P}_1 \parallel \ldots \parallel \mathcal{P}_n$. The first definitions and lemmata consider once again only constraints on the stateless level (i.e., without set operations and set conditions).

Definition 2.28 (Update of [Hes19] (Def. 5.9))**.** *Let $\mathcal{A}$ be a constraint from $\mathcal{P}$.*

- *A term t is* i-specific *iff $t \in \mathit{GSMP}_{\mathcal{P}_i} \setminus (\mathit{Sec} \cup \{t \mid \emptyset \vdash t\})$ for a label i.*
- *A term t is* heterogeneous *iff there exists protocol-specific labels $l_1 \neq l_2$ and subterms t_1 and t_2 of t such that each t_i is l_i-specific.*
- *A term t is* homogeneous *iff it is not heterogeneous.*

We have then (with the same proof idea):

Lemma 2.29 (Update of [Hes19] (Lemma 5.10))**.** *If $\mathcal{A}$ is a constraint of $\mathcal{P}$, then every $t \in \mathit{GSMP}_{\mathcal{A}}$ is homogeneous.*

An important next step is that the intruder never needs to construct heteroge-

neous terms. For that we first define homogeneous intruder deduction:

$$\frac{}{M \vdash_{\text{hom}} t}\ t \in M$$

$$\frac{M \vdash_{\text{hom}} t_1 \quad \dots \quad M \vdash_{\text{hom}} t_n}{M \vdash_{\text{hom}} f(t_1, \dots, t_n)}\quad \begin{array}{l} f \in \Sigma^n_{\mathsf{pub}}, \\ f(t_1, \dots, t_n) \text{ homogeneous}, \\ f(t_1, \dots, t_n) \in \mathit{GSMP}_{\mathcal{P}} \end{array}$$

$$\frac{M \vdash_{\text{hom}} t \quad M \vdash_{\text{hom}} k_1 \quad \dots \quad M \vdash_{\text{hom}} k_n}{M \vdash_{\text{hom}} t_i}\quad \begin{array}{l} \mathsf{Ana}(t) = (k_1, \dots, k_n, T), \\ t_i \in T \end{array}$$

Lemma 2.30 (Update of [Hes19] (Lemma 5.12))**.** *Given a finite set of messages $M \subset \mathit{GSMP}_{\mathcal{P}}$ and a term $t \in \mathit{GSMP}_{\mathcal{P}}$, then $M \vdash t$ iff $M \vdash_{\text{hom}} t$.*

Proof. This is only a minor update in the proof; the essential idea is that we can do proof normalization. If the proof tree for $M \vdash t$ contains a composition $f(t_1, \dots, t_n)$ from known t_i that is being analyzed to obtain one of the t_i again, then we can simplify the proof for that t_i. Further, since for homogeneous t, $\mathsf{Ana}(t) = (K, T)$ has that also K and T are all homogeneous terms, and the goal term t of the statement must be homogeneous, no heterogeneous term in the derivation remains after proof normalization. □

From homogeneity follows, where $\mathsf{ik}(\mathcal{A})$ is the intruder knowledge in $\mathcal{A}$:

Lemma 2.31 (Update of [Hes19] (Lemma 5.13))**.** *Given a constraint $\mathcal{A}$ of $\mathcal{P}$ and a well-typed model $\mathcal{I}$ such that $\forall s \in \mathit{Sec} \setminus \mathit{declassified}_{\mathcal{DY}}(\mathcal{A}, \mathcal{I})$. $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \not\vdash_{\text{hom}} s$ for any label i, (i.e., none of the protocols in isolation leaks a classified secret in $\mathcal{I}(\mathcal{A})$). Let $\mathsf{ik}(\mathcal{I}(\mathcal{A})) \vdash_{\text{hom}} t$, then $t \notin \mathit{Sec} \setminus \mathit{declassified}_{\mathcal{DY}}(\mathcal{A}, \mathcal{I})$ and if $t \in \mathit{GSMP}(\mathcal{A}|_i)$ for some i, then $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \vdash_{\text{hom}} t$.*

Thus, these lemmata together give that if the protocols do not leak secrets, then every derivation of a term that belongs to protocol i can be achieved in the projection to protocol i of the constraint:

Lemma 2.32 (Update of [Hes19] (Lemma 5.14))**.** *Given a constraint $\mathcal{A}$ of $\mathcal{P}$ and a well-typed model $\mathcal{I}$. Then $\mathcal{A}$ leaks a secret from Sec or for every $\mathsf{ik}(\mathcal{I}(\mathcal{A})) \vdash t \in \mathit{GSMP}(\mathcal{A}|_i)$, we have $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \vdash t$ where i is a protocol-specific label.*

The next step is:

Lemma 2.33 (Update of [Hes19] (Lemma 5.15))**.** *Given a constraint $\mathcal{A}$ of $\mathcal{P}$ and a well-typed model $\mathcal{I}$. Then either some prefix of $\mathcal{A}$ leaks a secret, or $\mathcal{I} \models \mathcal{I}(\mathcal{A}|_i)$ for every label i.*

Proof. This proof actually does not need an update with respect to our modification of declassification, but one in order to lift the original requirement that no receiving step in a transaction is labeled $\star$.

Suppose $\mathcal{A} = \mathcal{A}'.(l\colon \mathfrak{s})$ for a constraint $\mathcal{A}'$ on which the statement already holds and a step $\mathfrak{s}$ labeled l. If a prefix of $\mathcal{A}'$ leaks a secret, then so does a prefix of $\mathcal{A}$. Suppose thus, $\mathcal{I} \models \mathcal{I}(\mathcal{A}'|_i)$ for every label i, and we have to show that also $\mathcal{I} \models \mathcal{I}(\mathcal{A}|_i)$ for all i. We do this by case distinction of l and $\mathfrak{s}$.

- $\mathfrak{s}$ is a receive step of $\mathcal{A}$ (thus, a send in the corresponding transaction): this is not problematic as it only augments the intruder knowledge and, if $\star$-labeled, also the declassified terms, but cannot invalidate $\mathcal{I}$.

- $\mathfrak{s} = \xrightarrow{t}$ (thus a receive of a transaction) and l is a protocol specific label: since $\mathsf{ik}(\mathcal{I}(\mathcal{A}')) \vdash \mathcal{I}(t)$ (since $\mathcal{I} \models \mathcal{A}'$), we have by Lemma 2.32 that also $\mathsf{ik}(\mathcal{I}(\mathcal{A}'|_i)) \vdash \mathcal{I}(t)$.

- The difficult part is if $\mathfrak{s} = \xrightarrow{t}$ and the label l is $\star$. Avoiding this case in the proof was indeed the reason for forbidding transaction with a star-labeled receive (i.e. $\star$-labeled send in the resulting constraints). However, it can be proved without as seen in [HMB20]. First, $\mathcal{I}(t)$ must be in $Sec \cup \{t \mid \emptyset \vdash t\}$, because it occurs in the projections $\mathcal{I}(\mathcal{A}|_i)$ for all labels i and this would violate GSMP-disjointness if $\mathcal{I}(t)$ had any i-specific subterms. If it is public (i.e., in $\{t \mid \emptyset \vdash t\}$) or declassified (i.e., in $\mathit{declassified}_{\mathcal{DY}}(\mathcal{I}(\mathcal{A}'))$), then the statement easily follows. There remains the most tricky case: $\mathcal{I}(t) \in Sec \setminus \mathit{declassified}_{\mathcal{DY}}(\mathcal{I}(\mathcal{A}'))$. Since $\mathcal{I} \models \mathcal{A}$, we have that $\mathsf{ik}(\mathcal{I}(\mathcal{A}')) \vdash \mathcal{I}(t)$. To show: one of the $\mathcal{P}_i$ is to blame for leaking this classified secret (and thus concluding this case).

  Consider the normalized derivation proof for $\mathsf{ik}(\mathcal{I}(\mathcal{A}')) \vdash \mathcal{I}(t)$.

  - If the root operation is a compose step, i.e., producing a term of the form $f(t_1, \ldots, t_n)$, then the t_i are in $Sec \cup \{t \mid \emptyset \vdash t\}$. Then, also one of the t_i must be in $Sec \setminus \mathit{declassified}_{\mathcal{DY}}(\mathcal{I}(\mathcal{A}'))$ (otherwise, if all t_i are public or declassified, then also $f(t_1, \ldots, t_n) \in \mathit{declassified}_{\mathcal{DY}}(\mathcal{I}(\mathcal{A}'))$). In this case, we shall continue with the respective subterm. By repeatedly applying this argument we thus arrive at a node in the derivation tree that is not a composition, but an (Axiom) or (Decompose) step, and such that this term is in $Sec \cup \{t \mid \emptyset \vdash t\}$ as handled by the following cases.
  - We assume thus we have a term $t_0 \in Sec \cup \{t \mid \emptyset \vdash t\}$ and $\mathsf{ik}(\mathcal{I}(\mathcal{A}')) \vdash t_0$, and root node of the derivation tree is (Axiom) or (Decompose). In the case of analysis, note that the term being analyzed cannot be obtained by a (Compose) step due to proof normalization, so it

is either itself obtained by (Axiom) or (Decompose). We follow this chain of analysis steps until we reach a message that was obtained by (Axiom). In any case this message contains t_0 as a subterm and was sent by some protocol $\mathcal{P}_i$ and must be homogeneous. Thus all keys that have been used to obtain the analysis steps along the path to t_0 must be labeled i or $\star$. Thus, by Lemma 2.32, we have either a leak or $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \vdash t_0$, which is then also a leak.

- Equalities and inequalities are also not problematic as they are all already satisfied since $\mathcal{I}$ is a model of $\mathcal{A}$.

□

[Hes19] (Lemma 5.16) does not change in statement or proof: if no protocol leaks, then every constraint $\mathcal{A}$ with model $\mathcal{I}$ has a well-typed model that is a model of every project $\mathcal{A}|_i$.

Again, [Hes19] (Sec. 5.4.2) lifts the previous result from ordinary constraints (without set operations and conditions) to the stateful level by a translation from stateful to ordinary constraints and [Hes19] (Sec. 5.4.3) lifts it to the protocol level. This requires no changes for our modification of declassification.

2.6 Application of the theorems

In this section, we want to show in detail how to apply our results to the vertical composition $\frac{\mathsf{App}}{\mathsf{Ch}}$ from our running example (see Figures 2.2 and 2.3) as summarized in the end of Section 2.3. Thus, following Definition 2.15, we need to show that:

1. $(\mathsf{Ch}, \mathsf{App}, \mathit{Sec})$ is parallel composable,
2. $\mathit{GSMP}_{\mathsf{App}} \subseteq \mathit{Sec} \cup \{t \mid \emptyset \vdash t\}$,
3. $\mathit{GSMP}_{\mathsf{Ch}^\sharp} \cap \mathit{GSMP}_{\mathsf{App}} \subseteq \{t \mid \emptyset \vdash t\}$, and
4. none of the keys in K or their subterms in an analysis rule for a channel term s.t. $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ are labeled App.

Let us start with showing that $(\mathsf{Ch}, \mathsf{App}, \mathit{Sec})$ is parallel composable (Definition 2.9). This means that we have to show that:

a) $\mathsf{Ch} \parallel \mathsf{App}$ is *Sec-GSMP* disjoint from $\mathsf{Ch}^\star \parallel \mathsf{App}$,

b) for all $s \in Sec$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in Sec$,

c) for all $l\colon (t, s)$, $l'\colon (t', s') \in \mathit{labeledsetops}(\mathsf{Ch} \parallel \mathsf{App})$, if (t, s) and (t', s') are unifiable then $l = l'$, and

d) $\mathsf{Ch} \parallel \mathsf{App}$ is type-flaw resistant and $\mathsf{Ch}, \mathsf{App}, \mathsf{Ch}^\star$ and $\mathsf{App}^\star$ are well-formed.

We give here the basis for the demonstration, for more details on how to prove the parallel composability, we refer the reader to [Hes19].

Following Definition 2.4, let us first start by proving the type-flaw resistance (1d) of $\mathsf{Ch} \parallel \mathsf{App}$, i.e., we can prove the type-flaw resistance of the following set of steps M that subsumes the steps of $\mathsf{Ch} \parallel \mathsf{App}$ as well-typed instances, where $\Gamma(\{P\}) = \{\mathsf{Alias}\}$, $\Gamma(\{C, S\}) = \{\mathsf{Agent}\}$, $\Gamma(\{N\}) = \{\mathsf{Nonce}\}$, $\Gamma(\{K\}) = \{\mathsf{Key}\}$, $\Gamma(\{A, B\}) = \{\mathsf{Names}\}$ and $\Gamma(\{X\}) = \{\mathfrak{p}\}$:

$$
\begin{aligned}
M = \{ & P \mathrel{\dot\notin} \mathsf{taken}, P \to \mathsf{taken}, P \to \mathsf{alias}(C), \xrightarrow{P}, N \to \mathsf{sent}(S, P), \\
& f_{challenge}(N, S) \to \mathsf{outbox}(S, P), f_{challenge}(N, S) \leftarrow \mathsf{inbox}(S, P), \\
& N \leftarrow \mathsf{sent}(S, P), P \mathrel{\dot\in} \mathsf{alias}(C), K \to \mathsf{sessKeys}(P, B), \\
& f_{response}(\mathsf{mac}(\mathsf{secret}(C, S), N)) \to \mathsf{outbox}(P, S), \\
& f_{response}(\mathsf{mac}(\mathsf{secret}(C, S), N)) \leftarrow \mathsf{inbox}(P, S), (\forall C.P \mathrel{\dot\notin} \mathsf{alias}(C)), \\
& \xrightarrow{\mathsf{attack}_{\mathsf{App}}}, \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}, \\
& \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(f_{newSess}(P,B,K)))}, K \to \mathsf{sessKeys}(B, P), \\
& X \leftarrow \mathsf{outbox}(A, B), K \mathrel{\dot\in} \mathsf{sessKeys}(A, B), X \to \mathsf{secCh}(A, B), \\
& \xrightarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,X))}, \xleftarrow{\mathsf{scrypt}(K, f_{pseudo}(A,B,X))}, \\
& K \mathrel{\dot\in} \mathsf{sessKeys}(B, A), X \mathrel{\dot\in} \mathsf{secCh}(A, B), X \to \mathsf{inbox}(A, B), \\
& \xrightarrow{X} . \xleftarrow{X}, (\forall A, B.X \mathrel{\dot\notin} \mathsf{secCh}(A, B)), \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}} \}
\end{aligned}
$$

All variables have atomic types. Besides the non-constant, non-variable sub-message patterns of M consist of the composed terms and subterms closed under well-typed variable renaming and well-typed instantiation of the variables with constants. It is easy to see that each pair of non-variable terms amongst these composed sub-message patterns have compatible types if they are unifiable. There are no inequality checks in M, there remains just the conditions for negative checks to fulfill. There are only three negative checks, $(P \mathrel{\dot\notin} \mathsf{taken})$, $(P \mathrel{\dot\notin} \mathsf{alias}(C))$ and $(X \mathrel{\dot\notin} \mathsf{secCh}(A, B))$, and none of their subterms are generic for any set of variables.

It is easy to see that Ch, App, $\mathsf{Ch}^\star$ and $\mathsf{App}^\star$ are well-formed. Now that we proved that $\mathsf{Ch} \parallel \mathsf{App}$ is type-flaw resistant, we need to prove the other conditions for parallel composability. First let us look at the GSMP disjointness of $\mathsf{Ch} \parallel \mathsf{App}^*$ and $\mathsf{Ch}^\star \parallel \mathsf{App}$ (1a). The set $GSMP_{\mathsf{Ch}^\star \parallel \mathsf{App}}$ consists of the following set closed under subterms:

$$\begin{aligned}
&\{\mathsf{attack}_{\mathsf{App}}, (p, \mathsf{alias}(a)), (n_1, \mathsf{sent}(s, p)),\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{outbox}(p, s)),\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{inbox}(p, s)),\\
&\ (f_{challenge}(n_1, s), \mathsf{inbox}(p, s)), (f_{challenge}(n_1, s), \mathsf{outbox}(p, s)), \mathsf{inv}(p),\\
&\ (x, \mathsf{outbox}(a, b)), (x, \mathsf{secCh}(a, b)), (x, \mathsf{inbox}(a, b)), x,\\
&\ (p, \mathsf{taken}) \mid n_1, a, s, p, b, c, x \in \mathcal{C}, \Gamma(\{n_1\}) = \{\mathsf{Nonce}\},\\
&\ \Gamma(\{c, s\}) = \{\mathsf{Agent}\}, \Gamma(\{p\}) = \{\mathsf{Alias}\},\\
&\ \Gamma(\{a, b\}) = \{\mathsf{Names}\}, \Gamma(\{x\}) = \{\mathfrak{p}\}\},
\end{aligned}$$

and $GSMP_{\mathsf{Ch} \parallel \mathsf{App}^\star}$ consists of the following set closed under subterms:

$$\begin{aligned}
&\{\mathsf{attack}_{\mathsf{Ch}}, (k, \mathsf{sessKeys}(a, b)), \mathsf{scrypt}(k, f_{pseudo}(A, B, X)),\\
&\ (x, \mathsf{outbox}(b, c)), (x, \mathsf{secCh}(b, c)), (x, \mathsf{inbox}(b, c)), x,\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{outbox}(p, s)),\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{inbox}(p, s)),\\
&\ (f_{challenge}(n_1, s), \mathsf{inbox}(p, s)), (f_{challenge}(n_1, s), \mathsf{outbox}(p, s)), \mathsf{inv}(p),\\
&\ (k, \mathsf{sessKeys}(s, p)) \mid n_1, a, s, p, b, c, x, k \in \mathcal{C},\\
&\ \Gamma(\{n_1\}) = \{\mathsf{Nonce}\}, \Gamma(\{c, s\}) = \{\mathsf{Agent}\},\\
&\ \Gamma(\{p\}) = \{\mathsf{Alias}\}, \Gamma(\{a, b\}) = \{\mathsf{Names}\},\\
&\ \Gamma(\{x\}) = \{\mathfrak{p}\}, \Gamma(\{k\}) = \{\mathsf{Key}\}.\}
\end{aligned}$$

The terms occurring in the intersection of the *GSMP* are included in the following set *Sec*:

$$\begin{aligned}
&\{x, (x, \mathsf{secCh}(a, b)), (x, \mathsf{outbox}(a, b)), (x, \mathsf{inbox}(a, b)),\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{outbox}(p, s)),\\
&\ (f_{response}(\mathsf{mac}(\mathsf{secret}(c, s), n_1)), \mathsf{inbox}(p, s)),\\
&\ (f_{challenge}(n_1, s), \mathsf{inbox}(p, s)), (f_{challenge}(n_1, s), \mathsf{outbox}(p, s)), \mathsf{inv}(p),\\
&\ \mathsf{secret}(c, s), n_1\\
&\ \mid n_1, a, s, p, b, c, x \in \mathcal{C}, \Gamma(\{n_1\}) = \{\mathsf{Nonce}\},\\
&\ \Gamma(\{c, s\}) = \{\mathsf{Agent}\}, \Gamma(\{p\}) = \{\mathsf{Alias}\},\\
&\ \Gamma(\{a, b\}) = \{\mathsf{Names}\}, \Gamma(\{x\}) = \{\mathfrak{p}\}\}
\end{aligned}$$

The second condition (1b) is satisfied since any subterm of a term from *Sec* is either in *Sec* or an agent name. Finally, for the third condition (1c), we indeed have for all $l\colon (t,s), l'\colon (t',s') \in \mathit{labeledsetops}(\mathsf{Ch} \parallel \mathsf{App})$, if (t,s) and (t',s') are unifiable then $l = l'$.

Let us now look at the remaining conditions for the vertical composability following definition 2.15.

$\mathit{GSMP}_{\mathsf{App}}$ consists of the following set closed under subterms:

$$\begin{aligned}
&\{\mathsf{attack}_{\mathsf{App}}, (p, \mathsf{alias}(a)), (n_1, \mathsf{sent}(s,p)),\\
&\ (f_2(\mathsf{mac}(\mathsf{secret}(a,s), n_1)), \mathsf{inbox}(p,s)),\\
&\ (f_2(\mathsf{mac}(\mathsf{secret}(a,s), n_1)), \mathsf{outbox}(p,s)),\\
&\ (f_1(n_1, s), \mathsf{inbox}(s,p)), (f_1(n_1,s), \mathsf{outbox}(s,p)),\\
&\ \mathsf{inv}(p), (p, \mathsf{taken}) \mid p, n_1, a, s \in \mathcal{C},\\
&\ \Gamma(\{p\}) = \{\mathsf{Alias}\}, \Gamma(\{n_1\}) = \{\mathsf{Nonce}\},\\
&\ \Gamma(\{a,s\}) = \{\mathsf{Agent}\}\},
\end{aligned}$$

and $\mathit{GSMP}_{\mathsf{Ch}^\sharp}$ consists of the following set closed under subterms:

$$\begin{aligned}
&\{\mathsf{attack}_{\mathsf{Ch}}, (g, \mathsf{secCh}(a,b)), (k, \mathsf{sessKeys}(a,b)),\\
&\ (g, \mathsf{opened}), (g, \mathsf{closed}), \mathsf{scrypt}(k, f_{pseudo(a,b,g)}),\\
&\ (k, \mathsf{sessKeys}(b,a)), (k, \mathsf{sessKeys}(c,p)), g\\
&\ \mathsf{crypt}(\mathsf{pk}(b), \mathsf{sign}(\mathsf{inv}(p), f_{newSess}(p,c,k))),\\
&\ (k, \mathsf{sessKeys}(p,c)) \mid g, a, b, k, p \in \mathcal{C},\\
&\ \Gamma(\{g\}) = \{\mathfrak{a}\}, \Gamma(\{a,b\}) = \{\mathsf{Names}\},\\
&\ \Gamma(\{k\}) = \{\mathsf{Key}\}, \Gamma(\{p\}) = \{\mathsf{Alias}\},\\
&\ \Gamma(\{c\}) = \{\mathsf{Agent}\}\}
\end{aligned}$$

The second condition (2) and third condition (3) of vertical composability are verified. Besides, none of the keys in the protocol are labeled App, thus the fourth condition (4) is also verified.

We proved that the two protocols are vertical composable, and we proved in PSPSP [Hes+21] that $\mathsf{Ch}^\sharp$ and $\mathsf{Ch}^\star \parallel \mathsf{App}$ are secure, thus we can conclude with Corollary 2.17 that $\frac{\mathsf{App}}{\mathsf{Ch}}$ is secure.

2.7 Further examples

We include in this section further examples to illustrate the extend of our method. In particular, we want to show how to formalize different security goals for a channel protocol. We also want to highlight what aspects are mechanisms of a channel protocol and what aspects specify the guarantees that the channel exposes in its interface and thus what an application protocol is verified against when we verify $\mathsf{Ch}^\star \parallel \mathsf{App}$. We formalize the following examples:

- another variant of the channel from our running example that uses certificate to authenticate one endpoint in the key-exchange (Figure 2.7),
- a channel providing authentication without secrecy (Figures 2.8 to 2.10),
- two different channel mechanisms to guarantee replay protections that both expose the same interface, and (Figures 2.11 to 2.14)

We do not reintroduce notations when they have already been introduced in our main examples.

2.7.1 Key-exchange with certificate

$$\frac{\mathsf{Ch}_0 \colon \forall B \in \mathsf{Agent}, \mathrm{CA} \in \mathsf{CAuthority}.}{\mathsf{Ch}\colon \xrightarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(C)),\mathsf{certificate}(B,\mathsf{pk}(B)))}}$$

$$\frac{\mathsf{Ch}_1 \colon \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.}{\begin{array}{ll} \mathsf{Ch}\colon & \xleftarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(C)),\mathsf{certificate}(B,\mathsf{PKB}))}. \\ \mathsf{Ch}\colon & K \to \mathsf{sessKeys}(A,B). \\ \mathsf{Ch}\colon & \xrightarrow{\mathsf{crypt}(\mathsf{PKB},\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)),f_{newSess}(A,B,K)))} \end{array}}$$

Figure 2.7: Example for the key-exchange part of a channel with certificates

Consider first the variant of Ch in Figure 2.7 where an agent is authenticated with a certificate. Let $\mathsf{CAuthority}$ be a set of the public constants representing the honest certification authorities. Let $\mathsf{certificate}$ be a transparent function representing the certificate. In Ch_0, the certification authority CA signs a certificate for an agent B. In Ch_1, an honest agent with alias P generates a session key K for talking to an agent B, provided that she receives a valid certificate for

that agent from a certification authority, stores it in $\mathsf{sessKeys}(P, B)$ and signs it with the private key $\mathsf{inv}(P)$ of her alias, and encrypts it with the public key PKB of B included in the certificate. This protocol has the same rules $\mathsf{Ch}_2, \ldots, \mathsf{Ch}_7$ than the original channel. Even though the channel mechanisms are different, it offers the same guarantees, especially, we do not need to alter the formalization of goals. Therefore, the interface for this protocol is exactly the same than the one in Figure 2.4.

2.7.2 Authenticated channel without secrecy

We continue with an example of a channel that provides unilateral authentication but without secrecy in Figure 2.8. We consider a similar setting than the one in Figure 2.3, and therefore we consider the same set of principals. Additionally, let $\mathsf{mac}/2$, $f_{authentic}/4$ and $f_{mac}/3$ be public functions to respectively model a message authentication code and message formats. For the mac function, we have $\mathsf{Ana}(\mathsf{mac}(t_1, t_2)) = (\emptyset, \{t_2\})$ and $f_{authentic}$ and f_{mac} are transparent functions, i.e. $\mathsf{Ana}(f_{authentic}(t_1, t_2, t_3, t_4)) = (\emptyset, \{t_1, t_2, t_3, t_4\})$ and $\mathsf{Ana}(f_{mac}(t_1, t_2, t_3)) = (\emptyset, \{t_1, t_2, t_3\})$. Besides, we introduce a new family of set, $\mathsf{authCh}(A, B)$, that we describe shortly later.

The two first rules, Ch_1 and Ch_2, remain unchanged with respect to the running example. In Ch_3, an honest A can transmit a payload message X that an application protocol has inserted into an outbox set. For transmission, A generates a MAC of X with a key K that was established with B. In Ch_4, an honest B can receive the authenticated payload X from A, provided it is MAC-ed correctly with a key K that has been established with A. It is then inserted into $\mathsf{inbox}(A, B)$ to make it available on an application level.

We can now describe the security guarantees exposed in the interface, i.e., the $\star$-labeled steps. Comparing with the running example, we basically only replaced the set $\mathsf{secCh}(A, B)$ by the set $\mathsf{authCh}(A, B)$ that represents all messages ever sent by an honest A for an honest B—and we note that Ch_3 declassifies the payload X. Here, the interface warns that payloads are not guaranteed to be secret (but only authentic), i.e., the application must assume all messages handed to the channel will end up in the intruder knowledge.

Once again, the rules Ch_4 and Ch_7 bear similarities; they are applicable when a message that looks like a legitimate message from honest A to honest B with the right session key arrives at B. Ch_4 can fire if the corresponding X was indeed sent by A for B, i.e., $\mathsf{authCh}(A, B)$ holds. Otherwise, we have an authentication attack and Ch_7 fires. Thus again, the interface promises that the channel delivers only messages that indeed come from the claimed origin—and if this is not true,

$\mathsf{Ch}_1 : \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.$

$\mathsf{Ch}: \quad K \rightarrow \mathsf{sessKeys}(P, B).$

$\mathsf{Ch}: \quad \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}$

$\mathsf{Ch}_2 : \forall A \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\mathsf{Ch}: \quad \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P)),f_{newSess}(P,B,K))}.$

$\mathsf{Ch}: \quad K \rightarrow \mathsf{sessKeys}(B, P)$

$\mathsf{Ch}_3 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\star: \quad X \leftarrow \mathsf{outbox}(A, B).$

$\mathsf{Ch}: \quad K \dot{\in} \mathsf{sessKeys}(A, B).$

$\star: \quad X \rightarrow \mathsf{authCh}(A, B).$

$\mathsf{Ch}: \quad \xrightarrow{f_{authentic}(A,B,X,\mathsf{mac}(K,f_{mac}(A,B,X)))}.$

$\star: \quad \xrightarrow{X}$

$\mathsf{Ch}_4 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch}: \quad \xleftarrow{f_{authentic}(A,B,X,\mathsf{mac}(K,f_{mac}(A,B,X)))}.$

$\mathsf{Ch}: \quad K \dot{\in} \mathsf{sessKeys}(B, A).$

$\star: \quad X \dot{\in} \mathsf{authCh}(A, B).$

$\star: \quad X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_5 : \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.$

$\star: \quad X \leftarrow \mathsf{outbox}(A, B).$

$\star: \quad \xrightarrow{X}$

$\mathsf{Ch}_6 : \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.$

$\star: \quad \xleftarrow{X}.$

$\star: \quad X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_7 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch}: \quad \xleftarrow{f_{authentic}(A,B,X,\mathsf{mac}(K,f_{mac}(A,B,X)))}.$

$\mathsf{Ch}: \quad K \dot{\in} \mathsf{sessKeys}(A, B).$

$\star: \quad X \dot{\notin} \mathsf{authCh}(A, B).$

$\mathsf{Ch}: \quad \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

Figure 2.8: Example for an unilateraly authenticated channel without secrecy

then the channel has an attack according to Ch_7.

The intruder rules Ch_5 and Ch_6 are not modified. Now consider the idealization $\mathsf{Ch}^\star$ of the protocol. This still describes all changes that the channel can ever do to the sets outbox and inbox that it shares with the application (given that

$\underline{\mathsf{Ch}_3^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}$
$\star\colon X \leftarrow \mathsf{outbox}(A,B).$
$\star\colon X \rightarrow \mathsf{authCh}(A,B).$
$\star\colon \xrightarrow{X}$

$\underline{\mathsf{Ch}_4^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}$
$\star\colon X \mathrel{\dot\in} \mathsf{authCh}(A,B).$
$\star\colon X \rightarrow \mathsf{inbox}(A,B)$

$\underline{\mathsf{Ch}_5^\star\colon \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.}$
$\star\colon X \leftarrow \mathsf{outbox}(A,B).$
$\star\colon \xrightarrow{X}$

$\underline{\mathsf{Ch}_6^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.}$
$\star\colon \xleftarrow{X}.$
$\star\colon X \rightarrow \mathsf{inbox}(A,B)$

Figure 2.9: Idealization of the channel protocol from Figure 2.8

the channel protocol is safe). All messages sent by honest A to honest B move to a set $\mathsf{authCh}(A,B)$ and from there to the inbox of B. The main difference in this interface in Figure 2.9 compared to the one in Figure 2.4 is that messages transported on the channel are declassified.

For concision, we keep in the abstraction in Figure 2.10 only the rules that perform an action or that are not redundant, i.e., if after abstraction a rule contains only receiving and checking steps, then we drop it. We describe here the transformation in detail one more time. Ch_1 and Ch_2 remains unaffected by the transformations since they do not deal with any payload messages and only have Ch-labeled steps: these rules are "pure" channel rules. Thus, $\mathsf{Ch}_1^\sharp$ and $\mathsf{Ch}_2^\sharp$ are identical to the original rules.

A payload message X occurs in Ch_3, thus we need to divide this rule into two rules: $\mathsf{Ch}_{3a}^\sharp$ that contains the positive check ($\star\colon G \mathrel{\dot\in} \mathsf{opened}$), and $\mathsf{Ch}_{3b}^\sharp$ that contains the positive check ($\star\colon G \mathrel{\dot\in} \mathsf{closed}$). For $\mathsf{Ch}_{3a}^\sharp$, the step containing the set operation for outbox is dropped, and we replace the payload message that is inserted to secCh by the variable G of type $\mathfrak{a}$, as is the payload message in the transmitted message. For $\mathsf{Ch}_{3b}^\sharp$, the set operation for outbox is also dropped and the payload message inserted into the set secCh is replaced by the variable G of type $\mathfrak{a}$. It is also replaced in the transmitted message. However, since a variable of type $\mathfrak{a}$ that is in the closed is declassified, we need to replace the declassification step by the steps ($\star\colon G \leftarrow \mathsf{closed}.\star\colon G \rightarrow \mathsf{opened}.\star\colon \xrightarrow{G}$). The transformations for the rule Ch_4 leads to a rule performing no action so it is dropped.

The transformation for the rules Ch_5 and Ch_6 are the same as the ones of the main example. We only keep here the rule Ch_{5b} that is the only non redundant rule; it represents the abstraction of the declassification of a payload. We add still the rule $\mathsf{Ch}_{\mathsf{new}}^\sharp$ to allow for the creation of new variable of type $\mathfrak{a}$. Finally,

$\mathsf{Ch}_1^\sharp$: $\forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}$, new K.

Ch: $K \rightarrow \mathsf{sessKeys}(P, B)$.

Ch: $\xrightarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}$

$\mathsf{Ch}_2^\sharp$: $\forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}$.

Ch: $\xleftarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}$.

Ch: $K \rightarrow \mathsf{sessKeys}(B, P)$

$\mathsf{Ch}_{3a}^\sharp$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}$.

$\star$: $G \dot{\in} \mathsf{opened}$.

Ch: $K \dot{\in} \mathsf{sessKeys}(A, B)$.

$\star$: $G \rightarrow \mathsf{authCh}(A, B)$.

Ch: $\xrightarrow{f_{authentic}(A,B,G,\mathsf{mac}(K,f_{mac}(A,B,G)))}$.

$\star$: $\xrightarrow{G}$

$\mathsf{Ch}_{3b}^\sharp$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}$.

Ch: $K \dot{\in} \mathsf{sessKeys}(A, B)$.

$\star$: $G \rightarrow \mathsf{authCh}(A, B)$.

Ch: $\xrightarrow{f_{authentic}(A,B,G,\mathsf{mac}(K,f_{mac}(A,B,G)))}$.

$\star$: $G \leftarrow \mathsf{closed}(A, B)$.

$\star$: $G \rightarrow \mathsf{opened}(A, B)$.

$\star$: $\xrightarrow{G}$

$\mathsf{Ch}_{5b}^\sharp$:

$\star$: $G \leftarrow \mathsf{closed}$

$\star$: $G \rightarrow \mathsf{opened}$.

$\star$: $\xrightarrow{G}$.

$\mathsf{Ch}_{\mathsf{new}}^\sharp$: new G.

$\star$: $G \rightarrow \mathsf{closed}$

$\mathsf{Ch}_{7a,b}^\sharp$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}$.

$\star$: $G \dot{\in} \mathsf{opened}/G \dot{\in} \mathsf{closed}$.

Ch: $\xleftarrow{f_{authentic}(A,B,G,\mathsf{mac}(K,f_{mac}(A,B,G)))}$.

Ch: $K \in \mathsf{sessKeys}(A, B)$.

$\star$: $G \dot{\notin} \mathsf{authCh}(A, B)$.

Ch: $\xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

Figure 2.10: Abstraction for our example channel Ch from Figure 2.8

Ch_7 is also split into two rules. Further, in both rules, the payload X is replaced by the variable G of type $\mathfrak{a}$.

2.7.3 Channel with replay protection

$\mathsf{Ch}_1 : \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.$

$\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(P, B).$

$\mathsf{Ch} : \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}$

$\mathsf{Ch}_2 : \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f_{newSess}(P,B,K)))}.$

$\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(B, P)$

$\mathsf{Ch}_3 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}, \mathsf{new}\ N.$

$\star : X \leftarrow \mathsf{outbox}(A, B).$

$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$

$\star : X, N \rightarrow \mathsf{secCh}(A, B).$

$\mathsf{Ch} : \xrightarrow{\mathsf{scrypt}(K, f_{replay}(A,B,X,N))}$

$\mathsf{Ch}_4 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay}(A,B,X,N))}.$

$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(B, A).$

$\star : X, N \mathrel{\dot\in} \mathsf{secCh}(A, B).$

$\mathsf{Ch} : N \mathrel{\dot\notin} \mathsf{seen}(A, B).$

$\star : N \mathrel{\dot\notin} \mathsf{end}(A, B).$

$\mathsf{Ch} : N \rightarrow \mathsf{seen}(A, B).$

$\star : N \rightarrow \mathsf{end}(A, B).$

$\star : X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_5 : \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.$

$\star : X \leftarrow \mathsf{outbox}(A, B).$

$\star : \xrightarrow{X}$

$\mathsf{Ch}_6 : \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.$

$\star : \xleftarrow{X}.$

$\star : X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_7 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay}(A,B,X,N))}.$

$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$

$\star : X, N \mathrel{\dot\notin} \mathsf{secCh}(A, B).$

$\mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

$\mathsf{Ch}_8 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay}(A,B,X,N))}.$

$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$

$\star : X, N \mathrel{\dot\in} \mathsf{secCh}(A, B).$

$\mathsf{Ch} : N \mathrel{\dot\notin} \mathsf{seen}(A, B).$

$\star : N \mathrel{\dot\in} \mathsf{end}(A, B).$

$\mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

Figure 2.11: Example for an unilaterally authenticated pseudonymous channel with replay protection

$\mathsf{Ch}_3^\star$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}$, new N.

$\star{:}X \leftarrow \mathsf{outbox}(A,B)$.
$\star{:}X, N \rightarrow \mathsf{secCh}(A,B)$

$\mathsf{Ch}_4^\star$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}$.

$\star{:}X, N \mathrel{\dot\in} \mathsf{secCh}(A,B)$.
$\star{:}N \mathrel{\dot\notin} \mathsf{end}(A,B)$.
$\star{:}N \rightarrow \mathsf{end}(A,B)$.
$\star{:}X \rightarrow \mathsf{inbox}(A,B)$.

$\mathsf{Ch}_5^\star$: $\forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}$.

$\star{:}X \leftarrow \mathsf{outbox}(A,B)$.
$\star{:}\xrightarrow{X}$

$\mathsf{Ch}_6^\star$: $\forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}$.

$\star{:}\xleftarrow{X}$.
$\star{:}X \rightarrow \mathsf{inbox}(A,B)$

Figure 2.12: Idealization of the channel protocol from Figure 2.11

Let us now go back to our original example with a unilaterally authenticated pseudonymous channel, but let us add a replay protection mechanism in Figure 2.11. The first one we introduce is quite simple and not really practical, because it will basically require to remember nonces for messages received so far. The second one is a bit more involved, but will expose the same interface to the application and not demanding to remember much.

Let $f_{replay}/4$ be a public and transparent function. We also introduce two new sets: seen to keep track of identifiers that have already been received for the channel mechanism and end for specifying the replay protection goal (injective agreement).

The two first rules remain unchanged. In rule Ch_3, an honest A can transmit a payload message X that an application protocol has inserted into an outbox. In the transmission, it adds a fresh nonce N, and for encryption it uses a session key K that was established for that recipient. In Ch_4, an honest B can retrieve the encrypted payload X and the nonce N from A, provided it is encrypted correctly with a key K that has been established with A and that the nonce N has not been seen in an earlier exchange. The payload is then inserted into $\mathsf{inbox}(A,B)$ to make it available on the application level and the nonce is registered into the set $\mathsf{seen}(A,B)$.

We can now describe what has to do with the security guarantees in the interface. We keep the set $\mathsf{secCh}(A,B)$ that represents all messages and challenges ever sent by an honest A for an honest B. Note that here, we insert jointly X, N into the set; this allows for storing the same payload X several times, if A sends it several times to B. We introduce the set $\mathsf{end}(A,B)$ to formulate the injectivity aspect of the goal w.r.t. nonce N. Note once again the similarities between

$$\frac{\mathsf{Ch}_1^\sharp\colon \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \text{new } K.}{\begin{array}{l}\mathsf{Ch}\colon K \to \mathsf{sessKeys}(P,B).\\ \mathsf{Ch}\colon \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}\end{array}}$$

$$\frac{\mathsf{Ch}_2^\sharp\colon \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{Ch}\colon \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}.\\ \mathsf{Ch}\colon K \to \mathsf{sessKeys}(B,P)\end{array}}$$

$$\frac{\mathsf{Ch}_{3a,b}^\sharp\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}, \text{new } N.}{\begin{array}{l}\star\colon G \mathrel{\dot\in} \text{opened}/G \mathrel{\dot\in} \text{closed}.\\ \star\colon K \mathrel{\dot\in} \mathsf{sessKeys}(A,B).\\ \star\colon G,N \to \mathsf{secCh}(A,B).\\ \mathsf{Ch}\colon \xrightarrow{\mathsf{scrypt}(K,f_{replay}(A,B,G,N))}\end{array}}$$

$$\frac{\mathsf{Ch}_{4a,b}^\sharp\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star\colon G \mathrel{\dot\in} \text{opened}/G \mathrel{\dot\in} \text{closed}.\\ \mathsf{Ch}\colon \xleftarrow{\mathsf{scrypt}(K,f_{replay}(A,B,G,N))}.\\ \star\colon K \mathrel{\dot\in} \mathsf{sessKeys}(B,A).\\ \star\colon G,N \mathrel{\dot\in} \mathsf{secCh}(A,B).\\ \mathsf{Ch}\colon N \mathrel{\dot\notin} \mathsf{seen}(A,B).\\ \star\colon N \mathrel{\dot\notin} \mathsf{end}(A,B).\\ \mathsf{Ch}\colon N \to \mathsf{seen}(A,B).\\ \star\colon N \to \mathsf{end}(A,B)\end{array}}$$

$$\frac{\mathsf{Ch}_{5b}^\sharp\colon}{\begin{array}{l}\star\colon G \leftarrow \text{closed}\\ \star\colon G \to \text{opened}.\\ \star\colon \xrightarrow{G}\end{array}}$$

$$\frac{\mathsf{Ch}_{\mathsf{new}}^\sharp\colon \text{new } G.}{\star\colon G \to \text{closed}}$$

$$\frac{\mathsf{Ch}_{7a,b}^\sharp\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star\colon G \mathrel{\dot\in} \text{opened}/G \mathrel{\dot\in} \text{closed}.\\ \mathsf{Ch}\colon \xleftarrow{\mathsf{scrypt}(K,f_{replay}(A,B,G,N))}.\\ \mathsf{Ch}\colon K \mathrel{\dot\in} \mathsf{sessKeys}(A,B).\\ \star\colon G,N \mathrel{\dot\notin} \mathsf{secCh}(A,B).\\ \mathsf{Ch}\colon \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}\end{array}}$$

$$\frac{\mathsf{Ch}_{8a,b}^\sharp\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}{\begin{array}{l}\star\colon G \mathrel{\dot\in} \text{opened}/G \mathrel{\dot\in} \text{closed}.\\ \mathsf{Ch}\colon \xleftarrow{\mathsf{scrypt}(K,f_{replay}(A,B,G,N))}.\\ \mathsf{Ch}\colon K \mathrel{\dot\in} \mathsf{sessKeys}(A,B).\\ \star\colon G,N \mathrel{\dot\in} \mathsf{secCh}(A,B).\\ \star\colon N \mathrel{\dot\notin} \mathsf{seen}(A,B).\\ \star\colon N \mathrel{\dot\in} \mathsf{end}(A,B).\\ \mathsf{Ch}\colon \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}\end{array}}$$

Figure 2.13: Abstraction for our example channel Ch from Figure 2.11

rules Ch_4, Ch_7 and Ch_8; they are applicable when a message that looks like a legitimate message from honest A to honest B with the right session key arrives at B. Ch_4 can fire if the corresponding X was indeed sent for the first time

by A for B, i.e., $\mathsf{secCh}(A, B)$ holds and N is not in $\mathsf{end}(A, B)$ yet (and in this case we insert N into $\mathsf{end}(A, B)$). Otherwise, we either have an authentication attack and Ch_7 fires, or we have a replay attack and Ch_8 fires.

The rules Ch_5 and Ch_6 describe again the sending and the receiving operations for a dishonest principal and remain unchanged. Note that the idealization of this protocol in Figure 2.12 is slightly different than the one in our main example, since now the operations on $\mathsf{secCh}(A, B)$ must include a nonce N and the replay protection goal is stated with the set $\mathsf{end}(A, B)$.

We give the abstraction of the protocol in Figure 2.13. Note that we once again removed the redundant rules and the ones that do not perform an action anymore.

2.7.4 Second mechanism for replay protection

We now give an alternative protocol for replay protected channels that will exhibit the exact same interface, i.e., offers the same guarantees to an application protocol. This is sometimes useful to design a "canonical" and simple but inefficient solution (like all the nonces above have to be remembered) and then to replace it with a more efficient one that offers the same "functionality".

More generally, when we have two different channel protocols Ch_1 and Ch_2, but that offer the same guarantees for an application protocol ($\mathsf{Ch}_1^\star = \mathsf{Ch}_2^\star$), then for verifying the vertical compositions $\frac{\mathsf{App}}{\mathsf{Ch}_1}$ and $\frac{\mathsf{App}}{\mathsf{Ch}_2}$, it is enough to verify $\mathsf{Ch}_1^\sharp$, $\mathsf{Ch}_2^\sharp$ and $\mathsf{Ch}_1^\star \parallel \mathsf{App}$ since the two protocols have the exact same interface.

In this example, the mechanism to provide replay protection is based on a challenge-response mechanism. We still consider a similar setting as before and therefore the same set of principals. Additionally let $f_{replay2}/5$ and $f'_{newSess}/4$ be public and transparent functions. We need for this example to further consider two new families of set: $\mathsf{myChall}(A, B)$ that A uses to keep track of the challenges she issued to B and $\mathsf{theirChall}(A, B)$ that A uses to keep track of the challenges she received from B. However, these sets will at any time contain at most one value.

This time, the two rules Ch_1 and Ch_2 are modified for the key exchange to issue a challenge. In Ch_1, an honest agent with alias P generates a fresh session key for talking to an agent B, stores it in the set $\mathsf{sessKeys}(P, B)$, generates a fresh nonce to challenge the agent B, stores it in the set $\mathsf{myChall}(A, B)$ and signs them with the private key $\mathsf{inv}(P)$ of their alias, and encrypts it with the public

$\mathsf{Ch}_1 : \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \text{new } K, \text{new } N.$

$\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(P, B).$
$\mathsf{Ch} : N \rightarrow \mathsf{myChall}(P, B).$
$\mathsf{Ch} : \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f'_{newSess}(P,B,K,N)))}$

$\mathsf{Ch}_2 : \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B), \mathsf{sign}(\mathsf{inv}(P), f'_{newSess}(P,B,K,N)))}.$
$\mathsf{Ch} : N \rightarrow \mathsf{theirChall}(B, P).$
$\mathsf{Ch} : K \rightarrow \mathsf{sessKeys}(B, P)$

$\mathsf{Ch}_3 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}},$ new $M.$

$\star : X \leftarrow \mathsf{outbox}(A, B).$
$\mathsf{Ch} : N \leftarrow \mathsf{theirChall}(A, B).$
$\mathsf{Ch} : M \rightarrow \mathsf{myChall}(A, B).$
$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$
$\star : X, N \rightarrow \mathsf{secCh}(A, B).$
$\mathsf{Ch} : \xrightarrow{\mathsf{scrypt}(K, f_{replay2}(A,B,X,N,M))}$

$\mathsf{Ch}_4 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay2}(A,B,X,N,M))}.$
$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(B, A).$
$\mathsf{Ch} : N \leftarrow \mathsf{myChall}(B, A).$
$\star : X, N \mathrel{\dot\in} \mathsf{secCh}(A, B).$
$\mathsf{Ch} : M \rightarrow \mathsf{theirChall}(B, A).$
$\star : N \mathrel{\dot\notin} \mathsf{end}(A, B).$
$\star : N \rightarrow \mathsf{end}(A, B).$
$\star : X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_5 : \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.$

$\star : X \leftarrow \mathsf{outbox}(A, B).$
$\star : \xrightarrow{X}$

$\mathsf{Ch}_6 : \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.$

$\star : \xleftarrow{X}.$
$\star : X \rightarrow \mathsf{inbox}(A, B)$

$\mathsf{Ch}_7 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay2}(A,B,X,N,M))}.$
$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$
$\mathsf{Ch} : N \leftarrow \mathsf{myChall}(B, A).$
$\star : X, N \mathrel{\dot\notin} \mathsf{secCh}(A, B).$
$\mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

$\mathsf{Ch}_8 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

$\mathsf{Ch} : \xleftarrow{\mathsf{scrypt}(K, f_{replay2}(A,B,X,N,M))}.$
$\mathsf{Ch} : K \mathrel{\dot\in} \mathsf{sessKeys}(A, B).$
$\mathsf{Ch} : N \leftarrow \mathsf{myChall}(B, A).$
$\star : X, N \mathrel{\dot\in} \mathsf{secCh}(A, B).$
$\mathsf{Ch} : M \rightarrow \mathsf{theirChall}(B, A).$
$\star : N \mathrel{\dot\in} \mathsf{end}(A, B).$
$\mathsf{Ch} : \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}$

Figure 2.14: Second example for an unilaterally authenticated pseudonymous channel with replay protection

key $\mathsf{pk}(B)$ of B. In Ch_2, an honest agent B is receiving a session key K and a

nonce N encrypted with his public key and signed by an agent under an alias P. They insert K into their set $\mathsf{sessKeys}(B, P)$ and N in $\mathsf{theirChall}(B, P)$.

In Ch_3, an honest A can transmit a payload X and a response to the recipient's challenge that an application protocol has inserted into an outbox set using for encryption any session K that was established for communicating with B. They retrieve the challenge N that this recipient had emitted and generate a fresh nonce M for their response that they insert into their set $\mathsf{myChall}(A, B)$. In Ch_4, an honest B can retrieve the encrypted payload X and the response M to their challenge N from A, provided that it is encrypted correctly with a key K that has been established with A. The response is inserted into $\mathsf{theirChall}(B, A)$ and the payload is then inserted into $\mathsf{inbox}(A, B)$ to make it available on an application level.

This protocol and the previous one offer an additional guarantee compared to the protocol in our main example. The example of an application protocol in Figure 2.2 implements a challenge response itself and thus it is not relying on a replay protection by the channel. In fact, the App from the running example is perfectly fine with these two replay-protected channels.

$$\frac{\mathsf{App}_1 \colon \forall C \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.}{\begin{array}{l}\mathsf{App} \colon P \mathrel{\dot{\notin}} \mathsf{taken}.\\ \mathsf{App} \colon P \to \mathsf{taken}.\\ \mathsf{App} \colon P \to \mathsf{alias}(C)\end{array}} \qquad \frac{\mathsf{App}_2 \colon \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.}{\star \colon \xrightarrow{\mathsf{inv}(P)}}$$

$$\frac{\mathsf{App}_3 \colon \forall S \in \mathsf{Agent}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}, \mathsf{new}N.}{\begin{array}{l}\mathsf{App} \colon P \mathrel{\dot{\in}} \mathsf{alias}(C).\\ \mathsf{App} \colon N \to \mathsf{loginCounter}(P, S).\\ \quad \star \colon f_2(\mathsf{secret}(C, S)) \to \mathsf{outbox}(P, S)\end{array}}$$

$$\frac{\mathsf{App}_4 \colon \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.}{\begin{array}{l}\quad \star \colon f_2(\mathsf{secret}(C, S)) \leftarrow \mathsf{inbox}(P, S).\\ \mathsf{App} \colon P \mathrel{\dot{\in}} \mathsf{alias}(C).\\ \mathsf{App} \colon N \leftarrow \mathsf{loginCounter}(P, S)\end{array}}$$

$$\frac{\mathsf{App}_4 \colon \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.}{\begin{array}{l}\quad \star \colon f_2(\mathsf{secret}(C, S)) \leftarrow \mathsf{inbox}(P, S).\\ \mathsf{App} \colon P \mathrel{\dot{\notin}} \mathsf{alias}(C).\\ \mathsf{App} \colon \xleftarrow{\mathsf{attack}_{\mathsf{App}}}\end{array}}$$

Figure 2.15: Example of a login protocol without replay protection

Now with the additional guarantee for replay protection from the channel, we consider in Figure 2.15 a weaker login protocol as an application that relies on the replay protection mechanism provided by the channel. Rules App_1 and App_2 remain unchanged. The original rule App_3 has been removed since the server does not need to issue a challenge (this is now taken care of by the channel). This means that the server S does not need to create a fresh nonce anymore and to keep track of the ones that have not been answered with a set $\mathsf{sent}(S, P)$. The new rule App_3 describes how the client C updates its login counter on the server S with a fresh nonce N and sends its pre-shared secret, i.e., C inserts their response into her $\mathsf{outbox}(P, S)$, provided that P is an alias owned by C. The server removes this login attempt from the set $\mathsf{loginCounter}$ provided the login message has been sent by an alias that the client owns. If not, there is an attack. Remains then also the underlying secrecy goal that the intruder cannot learn any shared secret, e.g., $\mathsf{secret}(C, S)$ here.

2.8 Channel Bindings

There is a number of flaws in application protocols that arise from using secure channels where one party is not authenticated (like the channel in our running example) and using this channel to transmit a credential to authenticate that party (like our login application). The problem is that a dishonest server may forward these credentials in a man-in-the-middle attack to another server, pretending to be the owner of the credentials. This is for instance the case in the SAML SSO attack from [Arm+08] or the re-negotiation attack on TLS (cf. for instance the study about channel bindings in [BDP15]). In fact, one could say that this is all just the old attack on the Needham-Schroeder Public-Key Protocol in new disguises: if the messages fail to bind to context, a dishonest participant can abuse it, and designers often disregard the dishonest server in their intuitive analysis.

We show here an example akin to the gist of the TLS re-negotiation attack (borrowed from the formalization from [GM11]): the protocol in Figure 2.16 is meant to be run over the unilaterally authenticated channel from Figure 2.3. The first two rules App_1 and App_2 are again just honest agents choosing any number of aliases that have not been taken yet, and the intruder owning all dishonest aliases. Note that the unilaterally authenticated channel from Figure 2.3 only guarantees the authentication of the server side while the client side is authenticated only with respect to an alias. App_3 now describes that an honest agent A, who owns alias P, sends a critical command to the server. Critical here means that this requires the authentication of A. The function $\mathsf{critCmd}$ is a message format, and the nonce N represents some relevant arguments to

$\mathsf{App}_1 : \forall A \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.$

$\mathsf{App}: \quad P \mathrel{\dot{\notin}} \mathsf{taken}.$
$\mathsf{App}: \quad P \to \mathsf{taken}.$
$\mathsf{App}: \quad P \to \mathsf{alias}(A).$

$\mathsf{App}_2 : \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.$

$\star: \quad \xrightarrow{\mathsf{inv}(P)}.$

$\mathsf{App}_3 : \forall A \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, S \in \mathsf{Agent}, \mathsf{new}\ N.$

$\mathsf{App}: \quad P \mathrel{\dot{\in}} \mathsf{alias}(A).$
$\mathsf{App}: \quad \mathsf{critCmd}(N) \to \mathsf{begin}(A, S).$
$\star: \quad \mathsf{critCmd}(N) \to \mathsf{outbox}(P, S).$

$\mathsf{App}_4 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, \mathsf{new}\ M.$

$\star: \quad \mathsf{critCmd}(N) \leftarrow \mathsf{inbox}(P, S).$
$\mathsf{App}: \quad (\mathsf{critCmd}(N), M) \to \mathsf{pending}(P, S).$
$\star: \quad \mathsf{authPlease}(M) \to \mathsf{outbox}(S, P).$

$\mathsf{App}_5 : \forall A \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, S \in \mathit{Agent}.$

$\mathsf{App}: \quad P \mathrel{\dot{\in}} \mathsf{alias}(A).$
$\star: \quad \mathsf{authPlease}(M) \leftarrow \mathsf{inbox}(S, P).$
$\mathsf{App}: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{sigKey}(A)), \mathsf{ack}(M)) \to \mathsf{outbox}(P, S).$

$\mathsf{App}_6 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, A \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\star: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{sigKey}(A)), \mathsf{ack}(M)) \leftarrow \mathsf{inbox}(P, S).$
$\mathsf{App}: \quad (\mathsf{critCmd}(N), M) \leftarrow \mathsf{pending}(P, S).$
$\mathsf{App}: \quad \mathsf{critCmd}(N) \mathrel{\dot{\notin}} \mathsf{begin}(A, S).$
$\mathsf{App}: \quad \xrightarrow{\mathsf{attack}_{\mathsf{App}}}.$

Figure 2.16: Example of an application (to deploy over a unilaterally authenticated secure channel) that has a renegotiation-style flaw.

the command. For the security goal, it is noted in the set $\mathsf{begin}(A, S)$ that A really meant to issue this command to S. An honest server in App_4 receives this command and, since it comes from the unauthenticated source P, marks it as pending and sends an authentication request with a fresh challenge M (also the function $\mathsf{authPlease}$ is a message format). App_5 models how an honest A answers this challenge using a signature with a dedicated signing key pair, and we assume the server knows the public key $\mathsf{sigKey}(A)$ and that it belongs to A; again, ack is a format. Finally, when the server receives a response with a signature by an agent A that fits a pending request of a critical command N, then the server infers that this command was indeed issued by A. We formalize here with App_6 only that this would be an authentication problem, if A were

an honest agent and $\mathsf{critCmd}(N) \dot{\notin} \mathsf{begin}(A, S)$, i.e., the server is believing a command to come from A while this is actually not the case.

An attack with App_6 is actually possible: suppose an honest agent a under alias p_1 starts a session with a dishonest server i, and i starts a session under alias p_2 with the honest server s, issuing some critical command. The intruder forwards authentication request $\mathsf{authPlease}(m)$ from s to a/p_1 who responds with a corresponding signature according to App_5. With this, App_6 is applicable, since i can now authenticate its command as coming from a, while the critical command is actually not in $\mathsf{begin}(a, s)$.

More generally, the problem is the following. The fact that we have a secure channel with an agent (under some alias) does not mean that all messages we receive from that agent are necessarily authored by that agent. In the case of the login application of Figure 2.2, this is not a problem, since a dishonest server only learns the password that a user has with this server and cannot re-use this in other sessions (unless, of course, a user uses the same password with multiple servers). In contrast, the application of Figure 2.16 has a flaw since the signature that is meant to authenticate the endpoint could in fact come from a different run.

This demonstrates that the compositionality does not magically give us secure protocols, but it guarantees that when there is a flaw, like in the example, we can identify one or two culprits:

- the channel does not live up to what its interface promises, and in this case we find an attack against $\mathsf{Ch}^\sharp$, and
- or (as in this example) even if the channel behaves as advertised, the application could fail to achieve its goals, and in this case we find an attack against $\mathsf{App} \parallel \mathsf{Ch}^\star$.

A fix for the App protocol in Figure 2.16 could be to require that the command directly be authenticated, i.e., replacing the signature in rules App_5 and App_6 with $\mathsf{sign}(\mathsf{inv}(\mathsf{sigKey}(A)), f_{\mathsf{context}}(\mathsf{critCmd}(N), \mathsf{ack}(M), S))$. For good measure, we have here even included the name of the server S. A similar solution is also advised by [Arm+08] to fix the attack on SAML SSO: here the original protocol contains a credential from the identity provider that basically just acknowledges that the holder is a particular person, but this is of course not secure when the recipient (the relying party) is dishonest and authenticates itself with it. The solution is here to include the name of the relying party to prevent forwarding.

Finally, let us look at an example where the party has a key certificate, say

$\mathsf{sign}(\mathsf{inv}(\mathsf{pkca}), f_{\mathsf{cert}}(A, \mathsf{pk}(A)))$ (where pkca is the key of a certificate authority; we omit other typical fields like expiration for simplicity). If such a credential is used over a unilaterally authenticate channel to authenticate A, we have the same problems as before that a dishonest server could abuse this certificate to pose as A. Also in this case, we thus have to require that A signs something with the private key $\mathsf{inv}(\mathsf{pk}(A))$ to prove ownership, and the signed text must contain at least the name of B so that B cannot take this to another agent. This is shown in Figure 2.17 where we actually bind also the alias P to it. We have formulated here only the simple (non-injective) authentication goal that P indeed belongs to A. The binding to an alias has differences to the bindings usually considered in TLS and similar protocols [BDP15], as it has an asymmetric form, and we want to investigate further this relationship in future work.

$\mathsf{App}_1 : \forall A \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.$

$\mathsf{App}: \quad P \mathrel{\dot{\notin}} \mathsf{taken}.$
$\mathsf{App}: \quad P \to \mathsf{taken}.$
$\mathsf{App}: \quad P \to \mathsf{alias}(A).$

$\mathsf{App}_2 : \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.$

$\star: \quad \xrightarrow{\mathsf{inv}(P)} .$

$\mathsf{App}_3 : \forall A \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, S \in \mathsf{Agent}, \mathsf{new}\ N.$

$\mathsf{App}: \quad P \mathrel{\dot{\in}} \mathsf{alias}(A).$
$\star: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{pkca}), f_{\mathsf{cert}}(A, \mathsf{pk}(A))) \to \mathsf{outbox}(P, S).$
$\star: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), f_{\mathsf{pwn}}(A, P, S)) \to \mathsf{outbox}(P, S)$

$\mathsf{App}_4 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, A \in \mathsf{Agent}|_{\mathsf{Hon}}.$

$\star: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{pkca}), f_{\mathsf{cert}}(A, \mathsf{pk}(A))) \leftarrow \mathsf{inbox}(P, S).$
$\star: \quad \mathsf{sign}(\mathsf{inv}(\mathsf{pk}(A)), f_{\mathsf{pwn}}(A, P, S)) \leftarrow \mathsf{inbox}(P, S).$
$\star: \quad P \mathrel{\dot{\notin}} \mathsf{alias}(A).$
$\mathsf{App}: \quad \xrightarrow{\mathsf{attack}_{\mathsf{App}}} .$

Figure 2.17: Binding the alias P to a real name with an asymmetric credential.

We conclude with the remark that the application in Figure 2.17, composed with a suitable channel protocol, could be itself serve as a channel protocol, as it lifts the underlying unilaterally authenticated secure channel to a bilaterally authenticated one. This can be done by rules that take messages from a higher-level $\mathsf{outbox}_H(A, B)$, transfer them into the low-level $\mathsf{outbox}(A, B)$, and vice-versa on the receiving side with inboxes. Thus we can describe this as several layers of composition to achieve a channel with better guarantees.

2.9 Related Work and Conclusion

There exists a sequence of works on protocol composability that has pushed the boundaries of the class of protocols that can be composed, for instance [GT00; Gut09; CD09]. These works are concerned with protocols that do not *interact* with each other but just run independently on the same network, maybe sharing an infrastructure of fixed long-term keys. A limited form of interaction is allowed in [GM11] for vertical composition: a handshake protocol can generate secure keys that are then used to encrypt traffic of an application protocol; similarly, [CCW17] allows for sequential composition between a handshake establishing keys that can then be used by a subsequent protocol.

There are several refinement approaches that are close to vertical composition, such as [SB18], where a particular application that assumes abstract channels for communication gets refined by a particular implementation of a channel. The drawback of a refinement proof is that it has to be entirely re-done after changing the application. Indeed, the work [CCM15] bears the word *refinement* in its title, while it is actually a vertical *composition* (i.e., not specializing to a particular application) and is thus closest to our work. Our work generalizes this result in several regards: while [CCM15] considers only authentic, confidential and secure channels, we can specify any channel property that can be expressed by our formalism; this is of course also limited to trace-based properties but we can formulate all goals from the geometric fragment [Gut14]. Second, [CCM15] formulates the result only for secrecy goals of the application, while our result holds for all properties expressible in our formalism. Moreover, note that our formalism is stateful, i.e., both channel and application may use information that goes beyond single isolated sessions. This also includes a general notion of declassification that has not been present in any vertical composition approach so far. Moreover, [CCM15] requires a particular tagging scheme on protocols, while we have a more general non-unifiability requirement (that can be implemented by tagging but also instead by other forms of message structuring like XML or ASN.1). Last but not least, we want to point out the succinctness of our result. We see a contribution of this chapter in decomposing the problem into two smaller problems: a parallel composition of stateful protocols and a sound abstraction of payloads messages in the channel. For the first, we had to make a non-trivial extension to an existing compositionality result, namely handling abstract payload types and declassification, but this allows to reduce a large part of the problem to existing results, and can handle everything in greater generality. This is both mathematically economical and easy to understand and use.

Our work significantly generalizes [MV09; MV14], which were a first step in solving vertical composition without fixing a particular form of interaction, but

had to fix the number of transmissions that the channel can be used for, and the constructions are very complicated. We see as future work the application of our results in cases where the low-level protocol can hardly be called a channel but some general way to handle a form of payload, e.g., a distributed ledger, generalizing further the class of compositions that we support.

We emphasize that our results can be used with standard automated verification tools. Our compositionality result reduces the verification of $\frac{\mathsf{App}}{\mathsf{Ch}}$ to a number of syntactic conditions and the verification tasks of $\mathsf{Ch}^\star \parallel \mathsf{App}$ and $\mathsf{Ch}^\sharp$. In most cases, these are well suited for automated verification tools: while one can of course consider protocols that are not suitable for automated verification, our running example for instance requires only features expressible (with slight over approximation) in the standard tools like ProVerif [Bla01], AVISPA [Arm+05], Maude-NPA [EMM07], CPSA[Gut11] or Tamarin [Mei+13]. We have verified for instance our running example in Isabelle with PSPSP [Hes+21].

We see however three main limitations to our results. First, the behavior and goals must of course be expressible with transaction and sets, where the interface between low-level and high-level is just sets that one can only read and the other can only write—and the low-level is agnostic of the high-level data. Second, the results we are building on do not support algebraic properties, limiting the class of primitives that can be used, e.g., it is not possible yet to consider Diffie-Hellmann-based protocols. We consider the extension of this compositionality result to support the term algebra as future work. Third, we require that messages from channel and payload are discernable. This forbids multiple vertical compositions with several instances of the same channel protocol.

Finally, while this work is based on a black-box model of cryptography, there is a great similarity of the ideas in this chapter with the Universal Composability framework [Can01; KT11]. UC is typically used in a refinement style: one defines an ideal functionality and shows that (under appropriate cryptographic hardness assumptions) a particular real system implements the ideal one in the sense that real and ideal system cannot be distinguished. The real system can be for instance a channel protocol Ch and the ideal system would be similar to our abstraction $\mathsf{Ch}^\star$, i.e., abstractly describing properties of the channel without containing concrete cryptography. We can then verify an application being correct using $\mathsf{Ch}^\star$ instead of Ch. The differences to our work are that we do not consider one particular implementation Ch, but give a general methodology to verify an arbitrary implementation Ch, in particular, reducing the problem to one with abstract constants $\mathsf{Ch}^\sharp$ that is compatible with existing protocol verification tools. This allows notably also for payloads that can be declassified, even after occurring in a transmission. However, our model is Dolev-Yao style abstracting from cryptography and we consider it an interesting future challenge

to extend our ideas in UC style to a full cryptographic result.

Chapter 3

Preliminaries for Alpha-Beta Privacy

As we have mentioned earlier, it is common in protocol verification to consider an algebraic model of messages, namely interpreting functions in the quotient algebra modulo a set of algebraic equations (i.e., two terms are different *unless* the algebraic equations deem them equal). Many approaches usually reason about only logical implications, i.e., derivations that follow in *every* interpretation. In contrast, we reason from now on in this thesis about the different interpretations of formulae and hence have to make the interpretation of functions explicit. For this reason, we use *Herbrand logic* [HG06], a variant of first-order logic, that allows us just that. We adapt some useful notions from [MV19].

3.1 Herbrand Logic

(α, β)-privacy is based on specifying two formulae α and β in *First-Order Logic with Herbrand Universes*, or *Herbrand Logic* for short [HG06]. For brevity, we only list the differences with respect to standard first-order logic (*FOL*).

Herbrand Logic fixes the universe in which to interpret all symbols. We introduce a signature $\Sigma = \Sigma_f \uplus \Sigma_i \uplus \Sigma_r$ with Σ_f the set of *uninterpreted function symbols*, Σ_i the set of *interpreted function symbols* and Σ_r the set of *relation symbols*. Let $\mathcal{T}_{\Sigma_f}$ be the set of ground terms that can be built using symbols in Σ_f and let $\approx$ be a congruence relation on $\mathcal{T}_{\Sigma_f}$; then we define the *Herbrand Universe* as the quotient algebra $\mathcal{A} = \mathcal{T}_{\Sigma_f}/\approx = \{[\![t]\!]_\approx \mid t \in \mathcal{T}_{\Sigma_f}\}$, where $[\![t]\!]_\approx = \{t' \mid t \in \mathcal{T}_{\Sigma_f} \wedge t \approx t'\}$. The algebra fixes the "interpretation" of all uninterpreted function symbols: $f^{\mathcal{A}}([\![t_1]\!]_\approx, \ldots, [\![t_n]\!]_\approx) = [\![f(t_1, \ldots, t_n)]\!]_\approx$.

The interpreted function symbols Σ_i and the relation symbols Σ_r behave as in FOL, i.e., as function and relation symbols on the universe. To highlight the distinction between uninterpreted and interpreted function symbols, we write $f(t_1, \ldots, t_n)$ if $f \in \Sigma_f$ and $f[t_1, \ldots, t_n]$ if $f \in \Sigma_i$. Given a signature Σ, a set

$\mathcal{V}$ of variables distinct from Σ, and a congruence relation $\approx$, and thus fixing a universe A, we define an *interpretation* $\mathcal{I}$ (with respect to Σ, $\mathcal{V}$, and $\approx$) as a function such that: $\mathcal{I}(x) \in A$ for every $x \in \mathcal{V}$; $\mathcal{I}(f)\colon A^n \mapsto A$ for every $f/n \in \Sigma_i$ of arity n; and $\mathcal{I}(r) \subseteq A^n$ for every $r/n \in \Sigma_r$ of arity n. Note that the functions of Σ_f are determined by the quotient algebra. We define a *model relation* $\mathcal{I} \models \phi$ (in words: ϕ holds under $\mathcal{I}$) as is standard and use notation like $\phi \models \psi$.

Let Σ_f contain the constant 0 and the unary function s, and let Σ_i contain the binary function $+$, i.e., the universe contains the natural numbers $0, s(0), s(s(0)), \ldots$, which we also write as $0, 1, 2, \ldots$ We characterize $+$ by the axiom[1]:

$$\alpha_{ax} \equiv \forall x, y.\ x + 0 = x \ \wedge\ x + s(y) = s(x + y)\,.$$

We employ standard syntactic sugar and write, e.g., $\forall x.\ \phi$ for $\neg\exists x.\ \neg\phi$, and $x \in \{t_1, \ldots, t_n\}$ for $x = t_1 \vee \ldots \vee x = t_n$. Slightly abusing notation, we will also consider a substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ as a formula $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$.

3.2 Encoding of Frames

We use, as it is customary in security protocol analysis, a black-box algebraic model. We choose a subset $\Sigma_{op} \subseteq \Sigma_f$ of uninterpreted functions to be the *operators* available to the intruder. For instance, we generally require $0, s \in \Sigma_{op}$, so the intruder can "generate" any natural number. In order to represent the intruder's knowledge, we use frames.

Definition 3.1 (Frame)**.** *A* frame *is written as* $F = \{\!|m_1 \mapsto t_1, \ldots, m_l \mapsto t_l|\!\}$*, where the* m_i *are distinguished constants called* labels *and the* t_i *are terms that do not contain any* m_i*. We call* $m_1, \ldots, m_l$ *the* domain *and* $t_1, \ldots, t_l$ *the* image *of the frame. We write* $F\{\!|t|\!\}$ *for replacing in the term* t *every occurrence of* m_i *with* t_i*, i.e.,* F *works like a substitution.*

The labels m_i can be regarded as *memory locations* of the intruder, representing that the intruder knows the messages t_i. The set of *recipes* is the least set that contains $m_1, \ldots, m_l$ and that is closed under all the cryptographic operators in Σ_{op}.

We use two frames *concr* and *struct* that always have the same domain D in any formula. Let *concr* and *struct* be unary function symbols, and *gen* a unary

[1] This characterization is only possible due to the expressive power of Herbrand logic (in FOL one cannot characterize the universe appropriately).

relation symbol defined by the following axioms:

$$\begin{aligned}\phi_{gen} \equiv \forall r.gen(r) \Leftrightarrow \big(r \in D \vee \textstyle\bigvee_{f^n \in \Sigma_{op}} \exists r_1, \ldots, r_n. \\ r = f(r_1, \ldots, r_n) \wedge gen(r_1) \wedge \ldots \wedge gen(r_n)\big)\end{aligned}$$

$$\begin{aligned}\phi_{hom} \equiv \textstyle\bigwedge_{f^n \in \Sigma_{op}} \forall r_1, \ldots, r_n.\ gen(r_1) \wedge \ldots \wedge gen(r_n) \implies \\ concr[f(r_1, \ldots, r_n)] = f(concr[r_1], \ldots, concr[r_n]) \wedge \\ struct[f(r_1, \ldots, r_n)] = f(struct[r_1], \ldots, struct[r_n])\end{aligned}$$

$$\begin{aligned}\phi_{\sim} \equiv \forall r, s.\ gen(r) \wedge gen(s) \implies \\ concr[r] = concr[s] \Leftrightarrow struct[r] = struct[s]\end{aligned}$$

Then, the formula

$$\begin{aligned}\phi \equiv struct[l_1] = t_1 \wedge \ldots \wedge struct[l_n] = t_n \wedge \\ concr[l_1] = t'_1 \wedge \ldots \wedge concr[l_n] = t'_n\end{aligned}$$

specifies two frames *concr* and *struct* with domain $D = \{l_1, \ldots, l_n\}$ and the augmentation with the axioms above means that *concr* and *struct* are *statically equivalent*: for any pair of recipes r and s that the intruder can generate, *concr* agrees on r and s iff *struct* does. We define $\phi_{frame} \equiv \phi_{home} \wedge \phi$.

3.3 Alpha-Beta-Privacy

Privacy-type properties of security and voting protocols are often specified as *trace equivalence of two processes* in some process calculus, such as the Applied-π calculus [ABF18; BAF08; CRZ07; DRS08]. While such approaches have uncovered vulnerabilities in a number of protocols, they rely on asking whether the intruder can distinguish two variants of a process; e.g., the intruder should not be able to detect a difference between two processes differing only by the swap of the votes of two honest voters. It is quite hard to intuitively understand what such a trace equivalence goal actually entails and what not, and one may wonder if there are other trace equivalences that should be checked for. It is a rather technical way to encode the privacy goals of a protocol, and although one can get insights from a failed proof when the goal is too strong, one cannot easily see when it is too weak.

To fill the gap between intuitive ideas of the privacy goals and the mathematical notions used to formalize and reason about them, (α, β)-*privacy* has been proposed in [MV19]. It is a declarative approach based on specifying two formulae α and β in first-order logic with Herbrand universes. α formalizes the

payload, i.e., the "non-technical" information, that we intentionally release to the intruder, and β describes the "technical" information that he has, i.e., his "actual knowledge": what (names, keys, etc.) he initially knows, which actual cryptographic messages he observed and what he infers from them. He may be unable to decrypt a message, but know anyway that it has a certain format and contains certain (protected) information, e.g., a vote.

The distinction between payload and technical information is at the core of (α, β)-privacy. We formalize it by a distinguished subset $\Sigma_0 \subset \Sigma$ of the alphabet, where Σ_0 contains only the non-technical information, such as votes and addition, while $\Sigma \setminus \Sigma_0$ includes cryptographic operators. The formula α is always over just Σ_0, whereas β is over the full Σ.

Definition 3.2 (Model-theoretical (α, β)-privacy [MV19])**.** *Consider a countable signature Σ and a payload alphabet $\Sigma_0 \subset \Sigma$, a formula α over Σ_0 and a formula β over Σ s.t. $fv(\alpha) \subseteq fv(\beta)$, both α and β are consistent and $\beta \models \alpha$. We say that* (α, β)-privacy holds (model-theoretically) *iff every Σ_0-model of α can be extended to a Σ-model of β, where a Σ-interpretation $\mathcal{I}'$ is an extension of a Σ_0-interpretation $\mathcal{I}$ if they agree on all variables and all the interpreted function and relation symbols of Σ_0.*

In this thesis, we call model-theoretical (α, β)-privacy also *static possibilistic* (α, β)*-privacy* and, in contrast to [MV19], we allow β to have more free variables than α. All α formulae we consider in this thesis are *combinatoric*, which means that Σ_0 is finite and contains only uninterpreted constants. Then α has only finitely many models.

The common equivalence-based approaches to privacy are about the distinguishability between two alternatives. In contrast, (α, β)-privacy represents only one single situation that can occur, and it is the question what the intruder can deduce about this situation. To model this, we formalize that the intruder not only knows some concrete messages, but also that the intruder may know something about the structure of these messages, e.g., that a particular encrypted message contains a vote v_1, where v_1 is a free variable of α. Hence, we define the intruder knowledge by two frames *concr* and *struct*, where *struct* formalizes the *structural knowledge* of the intruder and thus may contain free variables of α, and the frame for the *concrete knowledge concr* is the same except that all variables are instantiated with what really happened, e.g., $v_1 = 1$. The main idea is that we require as part of β that *struct* and *concr* are statically equivalent, which means that if the intruder knows that two concrete constructible messages are equal, then also their structure has to be equal, and vice versa. For example, let $h \in \Sigma \setminus \Sigma_{op}$ and

$$struct = \{m_1 \mapsto h(v_1), m_2 \mapsto h(v_2)\} \text{ and } concr = \{m_1 \mapsto h(0), m_2 \mapsto h(1)\}\,.$$

Every model of β has the property $v_1 \neq v_2$. Suppose $\alpha \equiv v_1, v_2 \in \{0, 1\}$, then (α, β)-privacy is violated, since, for instance, $v_1 = 0, v_2 = 0$ is a model of α, but cannot be extended to a model of β. However, if $\alpha \equiv v_1, v_2 \in \{0, 1\} \wedge v_1 + v_2 = 1$, then all models of α are compatible with β and privacy is preserved.

Definition 3.3 (Message-analysis problem (adapted from [MV19]))**.** *Let α be combinatoric, and struct and concr be two frames with domain D. We say that β is a* message-analysis *problem if $\beta \equiv MsgAna(D, \alpha, struct, concr)$ with:*

$$MsgAna(D, \alpha, struct, concr) \equiv \alpha \wedge \phi_{gen} \wedge \phi_{frame} \wedge \phi_{\sim}$$

In the following, we assume β in every state to be implicitly augmented by the respective α and by the axioms ϕ_{gen}, ϕ_{hom} and $\phi_{\sim}$ where D is the set of labels occurring in β. In other words, we assume β in every state to be implicitly a message-analysis problem.

Chapter 4

Formalizing and Proving Privacy Properties of Voting Protocols using Alpha-Beta Privacy

Le suffrage par le sort est de la nature de la démocratie; le suffrage par choix est de celle de l'aristocratie.

Charles de Secondat, baron de Montesquieu
De l'esprit des lois (1748), II, 2.

"The suffrage by lot is natural to democracy, as that by choice is to aristocracy.", Charles de Secondat, baron de Montesquieu, The Spirit of the Laws (1748), II, 2.

Privacy is essential for freedom: to make a choice like a vote in a completely free way, i.e., determined only by one's own convictions, context, interests and expectations (rather than those of others), it is crucial that this choice cannot be observed by others. However, it is not sufficient to give people the possibility to make the choice in a private way: we also have to actually prevent them from *proving* what they have chosen. While one has the right to say what one has chosen (by the freedom of speech), we must also guarantee the possibility to *lie* about it, too. The reason is that otherwise we limit the privacy through a backdoor, as there can arise pressure to prove what one has chosen, especially when bribery or abuse of power comes into play. This chapter investigates the tension between privacy and coercion with the focus on voting privacy,

however, this is also relevant in other areas like electronic medical prescriptions (preventing pressure from the pharmaceutical industry onto doctors).

Related to the understanding of privacy goals is the problem that a demagogue can easily raise doubts about the legitimacy of an election. Our best chance to defeat this are open source systems that scientists, and ideally also ordinary people, can understand and convince themselves to be correct. The less obscure, the harder it is to defame, and the easier to recognize criticisms as unfounded. One of the challenges for describing systems in both a formally precise and intuitive way is privacy goals and their subtle relation to coercion.

This chapter gives a model-theoretic way to formalize and reason about privacy and receipt-freeness. We build on the framework of (α, β)-privacy [MV19], that defines privacy as a state-based safety property, where each state consists of two formulae α, the deliberately released high-level information like the result of an election, and β, the observations that the intruder could make. During transitions, the information in α and β can only increase (by adding new conjuncts to the formulae). The question is (for every reachable state), whether every model of α is compatible with the observations in β: if not, the intruder is able to rule out some models of α and thus has obtained more information about the system than we have deliberately released. In particular, we use this to come as close as possible to the following intuitive definition of privacy goals:

(a) **Voting privacy:** the number of voters and the result of the election are published at the end of the election. The intruder should not find out more than that about voters and votes.

(b) **Receipt-freeness:** no voter has a way to prove how they voted. This can be indirectly expressed by saying: for everything that could have happened according to (a), the voter can make up a consistent "story".

Our contribution is a general methodology to modeling voting privacy and receipt-freeness with (α, β)-privacy that can be applied for a variety of protocols. This in particular includes a proof methodology that allows for simple model-theoretic arguments, suitable for manual proofs and proof assistants like Isabelle and Coq. We illustrated this practically at hand of the FOO'92 protocol as an example, showing in particular how the use of permutations in the reasoning can lead to elegant proofs. In the model we propose, α is the same for both voting secrecy and receipt-freeness; the difference lies in β, namely in the challenge for a coerced voter to make a consistent story. From this construction, it immediately follows that receipt-freeness implies vote secrecy.

We do regard our models in (α, β)-privacy as a complementary view to existing

approaches like [DKR09]. We believe that their formalization may be equivalent in some sense to ours (at least we found no examples where the notions would differ), and regardless, our models aim to provide a fresh perspective. This holds in particular as it allows for a different style of proofs that are, in some sense, easier to conduct. While we consider only a static approach in this chapter, we extend this approach to a dynamic one in the next chapter.

4.1 Verifying Voting Privacy

An (α, β)-pair characterizes a single state of a transition system. To illustrate voting privacy and receipt-freeness we pick a few reachable states of the voting protocol FOO'92 and prove (or disprove) fulfillment of some properties. In fact, a manual proof for the entire infinite state-space is possible by appropriate generalization, but for the purpose of illustration, this would be a laborious task.

First, let us look at α and for simplicity consider a choice between 0 or 1 (all definitions can be easily extended to more complex voting choices). We use a sequence of variables $v_1, \ldots, v_N$ to model the votes. During each transition where an honest voter i sends a message that contains their vote v_i, we augment α by $v_i \in \{0, 1\}$, since the intruder does not know more than they will cast a valid vote. For dishonest voters, it is more complicated and actually depends on the protocol, since dishonest voters (or the intruder) may not follow the protocol and, e.g., replay a message of some honest voter (thus not necessarily knowing what vote they have cast). Anyway, in this case one should augment α with $v_i = b$ for the concrete value of the vote that they have cast, since the intruder is allowed to know all dishonest votes. Finally, when the result is about to be published and suppose the sum of the votes that equal 1 is R, then we finally augment α with the information $\sum_{i=1}^{N} v_i = R$. Therefore, from this point on, the intruder is allowed to know the result, but before this point, it is already a violation if he can obtain a partial result (beyond the votes of the dishonest agents). For all examples in this chapter, we have

$$\alpha \equiv v_1 \in \{0, 1\} \wedge \ldots \wedge v_N \in \{0, 1\} \wedge \sum_{i=1}^{N} v_i = R\,, \tag{4.1}$$

i.e., N honest voters have cast their votes, and R of these votes are 1. In fact, we also use the same formula in examples for receipt-freeness since we want that the same amount of information is kept secret, just some honest voters are "under more pressure" from the intruder.

4.1.1 The FOO'92 Voting Protocol in Alpha-Beta Privacy

The FOO'92 protocol [FOO92] has been formally studied with the Applied π-calculus [KR05]. We give here a brief description of the protocol and introduce relevant aspects on the fly. The FOO'92 protocol is divided in six distinct phases. In the preparation phase, each voter decides their vote and compute their ballot using a bit-commitment scheme. The ballot is then blinded and signed, before being sent to the administrator through an anonymous channel. In the administration phase, the administrator verifies that each received ballot comes from a legitimate voter that has not voted yet. The administrator checks the validity of the signature on the ballot, and sends a certified ballot to the voter. In the voting phase, each voter checks the validity of the certificate for their ballot and sends it to the counter through an anonymous channel. In the collecting phase, the counter checks the signature on the ballots, and then prints them in a random order on the bulletin board. In the opening phase, each voter checks that their ballot is printed on the bulletin board, and then sends their commitment value to the counter through an anonymous channel. In the counting phase, the counter opens all the commitments and publishes the votes and the result on the bulletin board.

The final result of FOO'92 is the publication of a bulletin board of cryptographic messages containing all the votes. More precisely, each entry contains $\mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v_i, r_i))$ and r_i for some $i \in \{1, \ldots, N\}$. This is the signature with the private key of an administrator A and contains a cryptographic commitment of the vote with some (initially secret) random value r_i. Here, we assume as part of Σ_{op} the following operators: sign for signature, verify for signature verification, and $\mathsf{retrieve}$ for obtaining the message under the signature; we also assume the following properties of the congruence relation on terms: $\mathsf{retrieve}(\mathsf{sign}(\mathsf{priv}(A), m)) \approx m$ and $\mathsf{verify}(\mathsf{pub}(A), \mathsf{sign}(\mathsf{priv}(A), m)) \approx \mathsf{true}$. Moreover, we have commit, $\mathsf{vcommit}$ and open for commitments with the properties $\mathsf{open}(\mathsf{commit}(m, r), r) \approx m$ and $\mathsf{vcommit}(r, \mathsf{commit}(m, r)) \approx \mathsf{true}$.

Let us consider an intruder who just obtains this bulletin board. This is not unrealistic since the exchanges in the other phases are best protected by anonymous channels, anyway. It is crucial that the bulletin board lists its entries in some unpredictable order. To model that, we introduce an interpreted function $\pi[\cdot]$ that is a permutation on $\{1, \ldots, N\}$.[1] To conveniently make use of this function, we like to also access the votes v_i and the random values r_i (these are uninterpreted constants of $\Sigma \setminus \Sigma_0$) through a function, and thus introduce two further interpreted functions $v[\cdot]$ and $r[\cdot]$ with the properties $v_i = v[i]$ and $r_i = r[i]$ for each $1 \leq i \leq N$.

[1] i.e., $\forall i.\, 1 \leq i \leq N \implies 1 \leq \pi[i] \leq N \wedge \forall j.\, 1 \leq j \leq N \wedge \pi[i] = \pi[j] \implies i = j$.

We can then describe the structural knowledge of the intruder who initially knows all public keys and has seen the bulletin board by the following frame:

$$\begin{aligned} \mathit{struct} = \{\!|\, & m_0 \mapsto \mathsf{pub}(A), m_1 \mapsto \mathsf{pub}(V_1), \ldots, m_n \mapsto \mathsf{pub}(V_N), \\ & m_{N+1} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[1]], r[\pi[1]])), \ldots, \\ & m_{2N} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[N]], r[\pi[N]])), \\ & m_{2N+1} \mapsto r[\pi[1]], \ldots, m_{3N} \mapsto r[\pi[N]] \,|\!\} \end{aligned}$$

To obtain the concrete knowledge frame, we need to replace the interpreted terms by their actual values. For the function π, this means the actual permutation used on the bulletin board; let us call it π_0. Mind π_0 is not a symbol of Σ but an actual permutation. Further, let $\theta_0 \models \alpha$ be an interpretation of each v_i by mapping them to $0, 1$ and that is a model of α, i.e., the true vote of every voter. Note that both π_0 and θ_0 are arbitrary, so the proofs we give hold for every such choice. Then, the concrete knowledge is obtained by replacing $\pi[x]$ by $\pi_0(x)$, $v[x]$ by $\theta_0(v_x)$ and $r[x]$ by r_x. Now we can specify the frame *concr* as follows:

$$\begin{aligned} \mathit{concr} = \{\!|\, & m_0 \mapsto \mathsf{pub}(A), m_1 \mapsto \mathsf{pub}(V_1), \ldots, m_n \mapsto \mathsf{pub}(V_N), \\ & m_{N+1} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(\theta_0(v_{\pi_0(1)}), r_{\pi_0(1)})), \ldots, \\ & m_{2N} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(\theta_0(v_{\pi_0(N)}), r_{\pi_0(N)})), \\ & m_{2N+1} \mapsto r_{\pi_0(1)}, \ldots, m_{3N} \mapsto r_{\pi_0(N)} \,|\!\} \end{aligned}$$

Thus, the frame *concr* replaces every occurrence of $v[\pi[i]]$ by $\theta_0(v_{\pi_0(i)})$ and every $r[\pi[i]]$ by $r_{\pi_0(i)}$. The point is that in the concrete messages the intruder observes, everything is instantiated with respect to π_0 and θ_0, while the structural knowledge about these messages is with respect to $\pi[\cdot]$ and $v[\cdot]$, i.e., reflecting what the intruder knows about the content of a message. For example, $v[\pi[j]]$ reflects that the intruder knows that this is the vote of the voter who has entry number j on the bulletin board, but he may be unable to find out the true permutation π_0 and neither the votes directly as a function of the voters. We know the technical information according to Chapter 3 and the protocol description:

$$\beta \equiv \bigwedge_{i=1}^{N} \Big(v[i] = v_i \wedge r[i] = r_i \Big) \wedge \mathit{MsgAna}(D, \alpha, \mathit{struct}, \mathit{concr}) \tag{4.2}$$

Let us call S the state with this β (4.2) and the α (4.1) previously defined.

4.1.2 Voting Privacy Holds in S

To show that (α, β)-privacy holds in S, we need to show how an arbitrary model of α can be extended to a model of β. To that end, we consider an arbitrary model $\theta_I \models \alpha$, called an *intruder hypothesis*, i.e., that maps each v_i to $\{0, 1\}$ so that their sum is R. We show how θ_I can be extended to a model $\mathcal{I} \models \beta$. In other words, we show that β does not allow the intruder to logically rule out any hypothesis about the votes v_i. We do this construction for an arbitrary θ_I, thus, *every* model of α can be extended to a model of β.

Since $\mathcal{I}$ must be an extension of θ_I, we have $\mathcal{I}(v_i) = \theta_I(v_i)$ for all votes v_i. Further, we need to give an interpretation for all other symbols of Σ, in our example $gen(\cdot)$, $struct[\cdot]$, $concr[\cdot]$, $\pi[\cdot]$, $r[\cdot]$ and of course $v[\cdot]$. For the symbols gen, $struct$, and $concr$, there is not much choice so that they satisfy the formulae ϕ_{gen} and ϕ_{frame}, and we give a canonical construction for them (i.e., the same construction applies for any message analysis problem). More interesting is to find an interpretation of the protocol specific functions $\pi[\cdot]$, $r[\cdot]$ and $v[\cdot]$, so that $\mathcal{I} \models \phi_{\sim}$, i.e., satisfying the static equivalence of $struct$ and $concr$ modulo $\approx$. While this is generally difficult, we are sometimes in luck: in some cases (α, β)-privacy allows for a relatively easy proof by reasoning about permutations—i.e., how "human provers" would like to do it. Indeed, for the state S, we can find an interpretation for $\pi[\cdot]$ (and the other functions) such that $\mathcal{I}(struct) = \mathcal{I}(concr)$. In this case $\mathcal{I} \models \phi_{\sim}$ follows trivially. Note that here, we do not even need to reason about algebraic properties of the operators (i.e., the congruence relation $\approx$) to conduct the proof.

The proof idea for this is actually straightforward in this case. Remember that S entails "what really happened", i.e., a particular model $\theta_0 \models \alpha$ and a particular permutation π_0 that reflect the true outcome of the vote, and the true permutation under which the votes have been published. The idea is that any discrepancy between θ_I and θ_0 can be "balanced" by an appropriate interpretation of π. More precisely, the voting function is interpreted following the intruder hypothesis, i.e., $v[i]$ is $\theta_I(v_i)$ for all voters. Since both $\theta_0 \models \alpha$ and $\theta_I \models \alpha$, we have $\sum_{i=1}^{N} \theta_0(v_i) = \sum_{i=1}^{N} \theta_I(v_i) = R$. Since $v_1, \ldots, v_N \in \{0, 1\}$, the list $[\theta_I(v_1), \ldots, \theta_I(v_N)]$ is a permutation of $[\theta_0(v_1), \ldots, \theta_0(v_N)]$. Thus, we can find a permutation $\rho\colon \{1, \ldots, N\} \to \{1, \ldots, N\}$ such that $\theta_I(v_i) = \theta_0(v_{\rho(i)})$ for all $i \in \{1, \ldots, N\}$. Intuitively, ρ is the discrepancy between θ_I and θ_0. Then, let us define π_I as the intruder's hypothesis of π: $\pi_I = \rho^{-1} \circ \pi_0$. Finally, r is interpreted accordingly, as the commitment secrets are permuted the same way that the votes, i.e., a value $r[i]$ is $r_{\rho(i)}$. Let us thus define (recall that the Herbrand universe is $A = \{[t]_{\approx} \mid t \in \mathcal{T}_{\Sigma_f}\}$):

Definition 4.1 (A model of the functions)**.** *Let $\mathcal{I}$ map v to the function $\mathcal{I}(v)\colon A \to A$, r to the function $\mathcal{I}(r)\colon A \to A$ and π to the function $\mathcal{I}(\pi)\colon A \to$*

A:

$$\mathcal{I}(v)([t]_\approx) = [\theta_I(v_t)]_\approx \text{ if } t \in [\{1,\ldots,N\}]_\approx \text{ and } \mathcal{I}(v)([t]_\approx) = [t]_\approx \text{ otherwise}$$
$$\mathcal{I}(r)([t]_\approx) = [r_{\rho(t)}]_\approx \quad \text{if } t \in [\{1,\ldots,N\}]_\approx \text{ and } \mathcal{I}(r)([t]_\approx) = [t]_\approx \text{ otherwise}$$
$$\mathcal{I}(\pi)([t]_\approx) = [\pi_I(t)]_\approx \quad \text{if } t \in [\{1,\ldots,N\}]_\approx \text{ and } \mathcal{I}(\pi)([t]_\approx) = [t]_\approx \text{ otherwise}$$

Example 4.1. *Given three voters, i.e., $N = 3$, and the result of the vote is $R = 2$, the true result of the vote is $\theta_0 = \{v_1 \mapsto 1, v_2 \mapsto 1, v_3 \mapsto 0\}$ and the actual permutation is $\pi_0 = \left(\begin{smallmatrix}1&2&3\\1&3&2\end{smallmatrix}\right)$, the bulletin board is then:*

Bulletin board	j	1	2	3
	$v_{\pi_0(j)}$	1	0	1

Let us consider one possible intruder hypothesis, i.e., one model of α, $\theta_I = \{v_1 \mapsto 0, v_2 \mapsto 1, v_3 \mapsto 1\}$. It is then possible to isolate one corresponding permutation: $\rho = \left(\begin{smallmatrix}1&2&3\\3&1&2\end{smallmatrix}\right)$. We can then build $\pi_I = \left(\begin{smallmatrix}1&2&3\\2&1&3\end{smallmatrix}\right)$.

The construction of the remaining items is generic for all message analysis problems, namely *struct* and *concr* behave like substitutions, and that *gen* is true exactly for the recipes:

Definition 4.2 (A model of *gen*, *struct* and *concr*)**.** *Let D be the domain of the frames struct and concr. Then we define*

$$\begin{aligned}\mathcal{I}(gen) &= \{[t]_\approx \mid t \in \mathcal{T}_{\Sigma_{op}\cup D}\}\\ \mathcal{I}(struct)([t]_\approx) &= \mathcal{I}(struct\{\!|t|\!\}) \text{ for all } t \in \mathcal{T}_{\Sigma_f}\\ \mathcal{I}(concr)([t]_\approx) &= \mathcal{I}(concr\{\!|t|\!\}) \text{ for all } t \in \mathcal{T}_{\Sigma_f}\end{aligned}$$

This interpretation expresses that *gen* is exactly the set of recipes. For *struct* and *concr*, we define the meaning by first applying the actual frames $struct\{\!|\cdot|\!\}$ and $concr\{\!|\cdot|\!\}$ as substitutions to a given term t, i.e., replacing the labels $m_i \in D$ in t; afterwards, we apply $\mathcal{I}$ to the resulting term since $struct\{\!|t|\!\}$ in general contains variables and interpreted function symbols that need to be interpreted.

This interpretation is well-defined because it does not depend on the choice of the representative of the equivalence classes, e.g., if $s \approx t$ then $struct\{\!|s|\!\} \approx struct\{\!|t|\!\}$. It is immediate that $\mathcal{I}$ is a model of ϕ_{frame}:

Lemma 4.3. *Let $\mathcal{I}$ be defined such as in Definition 4.2. Then $\mathcal{I} \models \phi_{frame}$.*

Proof. Following Definition 4.2, $\mathcal{I}$ models the second conjunct of ϕ_{frame}.

It remains to show that $\mathcal{I}$ models the first conjunct of ϕ_{frame}, namely ϕ_{hom}. Let $f^n \in \Sigma_{op}$. Let $r_1,\ldots,r_n$ be n recipes in $\mathcal{T}_{\Sigma_{op}\cup D}$. Note that $\mathcal{I}(r_i) = [r_i]_\approx$. It is

sufficient to show that $\mathcal{I} \models struct[f(r_1, \ldots, r_n)] = f(struct[r_1], \ldots, struct[r_n])$.

$$\begin{aligned}
\mathcal{I}(struct[f(r_1, \ldots, r_n)]) &= \mathcal{I}(struct)(\mathcal{I}(f(r_1, \ldots, r_n))) \\
&= \mathcal{I}(struct)([f(r_1, \ldots, r_n)]_{\approx}) \\
&= [struct\{\!|f(r_1, \ldots, r_n)|\!\}]_{\approx} \text{ by 4.2,} \\
&= [f(struct\{\!|r_1|\!\}, \ldots, struct\{\!|r_n|\!\})]_{\approx} \\
&= f([struct\{\!|r_1|\!\}]_{\approx}, \ldots, [struct\{\!|r_n|\!\}]_{\approx}) \\
&= f(\mathcal{I}(struct)([r_1]_{\approx}), \ldots, \mathcal{I}(struct)([r_n]_{\approx})) \text{ by 4.2,} \\
&= f(\mathcal{I}(struct)(\mathcal{I}(r_1)), \ldots, \mathcal{I}(struct)(\mathcal{I}(r_n))) \\
&= f(\mathcal{I}(struct[r_1]), \ldots, \mathcal{I}(struct[r_n])) \\
&= \mathcal{I}(f(struct[r_1], \ldots, struct[r_n])).
\end{aligned}$$

The development is similar for proving that $\mathcal{I} \models concr[f(r_1, \ldots, r_n)] = f(concr[r_1], \ldots, concr[r_n])$. Therefore, we proved that $\mathcal{I} \models \phi_{hom}$. Thus, $\mathcal{I} \models \phi_{frame}$. □

It remains to show that $\mathcal{I} \models \phi_{\sim}$. In general, such a proof of static equivalence of two frames can be difficult (especially by hand). However, in our case we have $\mathcal{I}(struct) = \mathcal{I}(concr)$ by construction—we have designed the interpretation of π so that this holds—and then static equivalence is immediate:

Lemma 4.4. *Let $\mathcal{I}$ be defined such as in Definition 4.2. If $\mathcal{I}(struct) = \mathcal{I}(concr)$ then $\mathcal{I} \models \phi_{\sim}$.*

Proof. Suppose $\mathcal{I}(struct) = \mathcal{I}(concr)$. Recall that *struct* and *concr* have the same domain D so $gen_{struct} = gen_{concr} = gen$. Let r and s be two recipes in $\mathcal{T}_{\Sigma_{op} \cup D}$. Suppose now that $\mathcal{I} \models struct[r] = struct[s]$.

$$\begin{aligned}
\mathcal{I} \models struct[r] = struct[s] \text{ iff } & \mathcal{I}(struct)(\mathcal{I}(r)) = \mathcal{I}(struct)(\mathcal{I}(s)) \\
\text{iff } & \mathcal{I}(concr)(\mathcal{I}(r)) = \mathcal{I}(concr)(\mathcal{I}(s)) \\
\text{iff } & \mathcal{I} \models concr[r] = concr[s].
\end{aligned}$$

Thus, $\mathcal{I} \models \phi_{\sim}$. □

We can now conclude on the voting privacy in the state S that we described.

Theorem 4.5. *Voting privacy holds in the state S.*

Proof. First, let us prove that $\mathcal{I}(struct) = \mathcal{I}(concr)$. For the *struct*, we just have to look at the interpretation of $v[\pi[i]]$ and $r[\pi[i]]$ because all the other

terms are uninterpreted symbols. For $i \in \{1, \ldots, N\}$,

$$\begin{aligned}
\mathcal{I}(v[\pi[i]]) &= \mathcal{I}(v)(\mathcal{I}(\pi)([i]_{\approx})) = \mathcal{I}(v)([\pi_I(i)]_{\approx}) = \mathcal{I}(v)([(\rho^{-1} \circ \pi_0)(i)]_{\approx}) \\
&= [\theta_I(v_{\rho^{-1}(\pi_0(i))})]_{\approx} = [\theta_0(v_{\pi_0(i)})]_{\approx} \\
\mathcal{I}(r[\pi[i]]) &= \mathcal{I}(r)(\mathcal{I}(\pi)([i]_{\approx})) = \mathcal{I}(r)([\pi_I(i)]_{\approx}) = \mathcal{I}(r)([(\rho^{-1} \circ \pi_0)(i)]_{\approx}) \\
&= [r_{(\rho \circ \rho^{-1} \circ \pi_0)(i)}]_{\approx} = [r_{\pi_0(i)}]_{\approx}.
\end{aligned}$$

Since for the *concr*, the messages are of the form $m_{N+i} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(\theta_0(v_{\pi_o(i)}), r_{\pi_0(i)}))$, we have $\mathcal{I}(struct) = \mathcal{I}(concr)$. Then, we have shown that for every model $\theta_I \models \alpha$, i.e., any possible intruder's hypothesis, we can find a model $\mathcal{I}$ of β that agrees with θ_I, i.e., $\mathcal{I}(v[i]) = \theta_I(v_i)$ for all votes v_i. □

This FOO'92 example demonstrates the declarativity of the (α, β)-privacy approach, in particular that we are able to reason about permutations allows for a rather simple proof how "human provers" would like it: after a small insight (the discrepancy between θ_I and θ_0 can be balanced in the interpretation of π), then the rest all falls into place.

4.1.3 Voting Privacy Holds in S'

In many cases, it is not as easy as before. For instance, in the FOO'92 protocol, we have a first phase where voters send a blind signature of their vote-commitment to an administrator and receive a signature from that administrator. Let us now consider a state S' where the intruder has seen also all theses blinded signatures (the formula α is again (4.1)):

$$\begin{aligned}
struct = \{\!|\, & m_0 \mapsto \mathsf{pub}(A), m_1 \mapsto \mathsf{pub}(V_1), \ldots, m_N \mapsto \mathsf{pub}(V_N), \\
& m_{N+1} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[1]], r[\pi[1]])), \ldots, \\
& m_{2N} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[N]], r[\pi[N]])), \\
& m_{2N+1} \mapsto r[\pi[1]], \ldots, m_{3N} \mapsto r[\pi[N]], \\
& m_{3N+1} \mapsto \mathsf{sign}(\mathsf{priv}(V_1), \mathsf{blind}(\mathsf{commit}(v[1], r[1]), b_1)), \ldots, \\
& m_{4N} \mapsto \mathsf{sign}(\mathsf{priv}(V_N), \mathsf{blind}(\mathsf{commit}(v[N], r[N]), b_N)), \\
& m_{4N+1} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{blind}(\mathsf{commit}(v[1], r[1]), b_1)), \ldots, \\
& m_{5N} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{blind}(\mathsf{commit}(v[N], r[N]), b_N)) |\!\}
\end{aligned}$$

Here, we have augmented the frame from S by the messages from the voters stored at $m_{3N+1}, \ldots, m_{4N}$, and the messages from the administrator stored at $m_{4N+1}, \ldots, m_{5N}$, where b_i is the corresponding blinding secret of voter V_i.

We assume the following property about blind and sign: $\mathsf{unblind}(\mathsf{sign}(\mathsf{priv}(A), \mathsf{blind}(m, b), b)) \approx \mathsf{sign}(\mathsf{priv}(A), m)$, so that each voter can unblind the reply message from the administrator. The concrete frame *concr* is again obtained by replacing $\pi[x]$ by $\pi_0(x)$, $v[x]$ by $\theta_0(v_x)$ and $r[x]$ by r_x.

Note that the messages between voters and administrators are actually shown in the order of the voters rather than under a permutation. The reason is that such a permutation would not make the problem harder for the intruder, since the signatures of the voters already identify which message belongs to whom and the replies from the administrator could probably be linked due to timing.

Now, the difficulty is that we cannot find an interpretation $\mathcal{I}$ such that $\mathcal{I}(\mathit{struct}) = \mathcal{I}(\mathit{concr})$, because the messages stored at $m_{3N+1}, \ldots, m_{4N}$ are signed by the individual voters and are thus linked to the voters.[2]

Instead, the point is here that, due to the blinding, the intruder cannot derive anything useful from these messages. Formally, we show for the same $\mathcal{I}$ as constructed for S (for given $\theta_I \models \alpha$), that the weaker property $\mathcal{I} \models \phi_\sim$ still holds in S'. This therefore requires a full static equivalence proof modulo the properties of $\approx$, which is quite involved (cf., for instance [Cha+16]).

Claim 4.1. *Voting privacy holds in the state S'.*

4.2 Receipt-freeness

We now assume that the intruder tries to influence one particular voter, let us call him Dan[3] and identify him with the first voter V_1. We will later briefly discuss the case when the intruder tries to influence several voters. The question is whether Dan can *prove* to the intruder how he voted by a kind of "receipt". The protocol does not explicitly produce any such receipt, but revealing all messages that Dan knows could allow the intruder to verify how Dan voted, i.e., that Dan is unable to lie about his vote. For instance, for FOO'92, we will now show, that if the intruder has observed all the messages between voters and administrators (state S'), and if Dan reveals his blinding factor, then the intruder can indeed identify Dan's vote with certainty. If we consider however FOO'92 without the blinded signature messages (as in state S), and that the intruder sees only the final bulletin board, Dan can claim any vote to be his—and the intruder has no

[2] In fact, due to the messages stored at $m_{3N+1}, \ldots, m_{4N}$, in any model $\mathcal{I}$ where $\mathcal{I}(\mathit{struct}) = \mathcal{I}(\mathit{concr})$ we necessarily also have $\theta_0 = \theta_I$, and thus there cannot be such a simple construction for every $\theta_I \models \alpha$.

[3] According to Saxo Grammaticus in his *Gesta Danorum, Book I*, Dan was the first legendary king of the Danes and Denmark.

chance to falsify that claim. Mind that does not hold in the state before the commitments are opened as we also discuss below. Note that the contribution of this section is not the further formalization of the FOO protocol, but the formalization of a privacy goal for receipt-freeness in (α, β)-privacy.

4.2.1 Formalizing Receipt-freeness

Consider a given state where we want to check whether the protocol is receipt-free with respect to the voter Dan. The intruder can ask Dan to reveal his *entire knowledge*, i.e., all the secrets Dan knows (his private key, his commitment value and his blinding factor) as well as messages that Dan has received from other parties, like the administrator. If Dan has any "receipt" (in the broadest sense of the word), then it is something that can be derived from his knowledge. The point is that Dan does not necessarily tell the truth, but can present any collection of messages that can be constructed from his knowledge. We call this *Dan's story*. Dan's story has to be consistent with whatever the intruder can check, e.g., Dan cannot lie about his private key, since the intruder knows his corresponding public key. We thus want to express that a state is receipt-free, if for every model $\theta_I \models \alpha$, Dan can come up with a consistent story (in particular consistent with θ_I). We do not even change the formula α, but only add an additional challenge in β: that the intruder obtains a story from Dan, i.e., what he claims to be his knowledge. We see receipt-freeness as preserving voting privacy even under this additional challenge. From that actually follows a relation between the goals: receipt freeness implies voting privacy.

We reason about Dan's knowledge similarly to the intruder's knowledge: we introduce the frames $concr_{Dan}$ and $struct_{Dan}$ whose domain $D_{Dan} = \{d_1, \ldots, d_l\}$ is disjoint from the domain D of the intruder knowledge: $D_{Dan} \cap D = \emptyset$. We consider similar axioms that we introduced in Section 3.2 for the frames *struct* and *concr*:

$$\begin{aligned}\phi_{hom,Dan} \equiv \bigwedge\nolimits_{f^n \in \Sigma_{op}} \forall r_1, \ldots, r_n.\, gen(r_1) \wedge \ldots \wedge gen(r_n) \implies \\ concr_{Dan}[f(r_1, \ldots, r_n)] = f(concr_{Dan}[r_1], \ldots, concr_{Dan}[r_n]) \\ \wedge\, struct_{Dan}[f(r_1, \ldots, r_n)] = f(struct_{Dan}[r_1], \ldots, struct_{Dan}[r_n])\end{aligned}$$

$$\begin{aligned}\phi_{Dan} \equiv struct_{Dan}[d_1] = t_1 \wedge \ldots \wedge struct_{Dan}[d_l] = t_l \\ \wedge\, concr_{Dan}[d_1] = t'_1 \wedge \ldots \wedge concr_{Dan}[d_l] = t'_l\end{aligned}$$

If we consider that the protocol itself is not a secret, the intruder "knows" $struct_{Dan}$, i.e., what the messages are supposed to be according to protocol, and Dan's story has to be consistent with this. The idea is that what Dan can lie about is $concr_{Dan}$. We let Dan choose any recipes $s_1, \ldots, s_l$ (with respect to

D_{Dan}), one for each item in his knowledge and send $concr[s_1], \ldots, concr[s_l]$ as his story to the intruder. The augmented intruder knowledge has then domain $D \cup D_{Dan}$ where the *concr*'s memory are filled with Dan's story and the *struct* is identical with $struct_{Dan}$, i.e., what it is supposed to be. This is captured by the formula ϕ_{lie} that we call the *Lying Axiom.*

Definition 4.6 (The Lying Axiom ϕ_{lie})**.** *Let Dan be the coerced voter, struct and concr the intruder's knowledge,* $struct_{Dan}$ *and* $concr_{Dan}$ *Dan's story with domain* $D_{Dan} = \{d_1, \ldots, d_l\}$*. Then, we define the lying axiom* ϕ_{lie}*:*

$$\begin{aligned}\phi_{lie}(Dan) \quad \equiv \quad & struct[d_1] = struct_{Dan}[d_1] \wedge \ldots \wedge struct[d_l] = struct_{Dan}[d_l] \\ & \wedge \exists\, s_1, \ldots, s_l.\, gen_{D_{Dan}}(s_1) \wedge \ldots \wedge gen_{D_{Dan}}(s_l). \\ & \big(concr[d_1] = concr_{Dan}[s_1] \wedge \ldots \wedge concr[d_l] = concr_{Dan}[s_l]\big)\end{aligned}$$

In fact, $\mathcal{I} \models \phi_{lie}(Dan)$ (w.r.t. the whole domain $D \cup D_{Dan}$) means that Dan's story is consistent with the protocol (i.e., the *struct* values) and $\mathcal{I}$'s interpretation of the free variables of α. Thus, we define:

Definition 4.7 (Receipt-freeness problem)**.** *We say that* β *is a* receipt-freeness *problem (with respect to a combinatoric* α*, the frames struct and concr with domain* $D \cup D_{Dan}$*, a coerced voter Dan and his story* $struct_{Dan}$ *and* $concr_{Dan}$ *with domain* D_{Dan}*) if* $\beta \equiv RcpFree(D, \alpha, struct, concr, Dan)$ *where:*

$$\begin{aligned}RcpFree(D, \alpha, struct, concr, Dan) \equiv\ & MsgAna(D \cup D_{Dan}, \alpha, struct, concr) \\ & \wedge \phi_{lie}(Dan) \wedge \phi_{hom,Dan} \wedge \phi_{Dan}\end{aligned}$$

We say receipt-freeness holds *if the* (α, β)*-pair is consistent. We call* $\beta' \equiv MsgAna(D, \alpha, struct, concr)$ *the message-analysis problem underlying* β*.*

β is always consistent since there is at least one way to satisfy β: the truth (i.e., Dan selects $s_i = d_i$ for each $1 \leq i \leq l$). Note that the story of Dan may be the truth when this is compatible with the intruder hypothesis (e.g., when $\theta_I = \theta_0$) without breaking receipt-freeness. What matters is only that the intruder cannot rule out any model of α, including the truth when θ_I coincides with θ_0.

The consistent "story" is here represented by the axiom ϕ_{lie}. For every receipt-freeness problem, we also defined an underlying message-analysis problem that is just a restriction of the original receipt-freeness problem. Indeed, the message-analysis problem is part of the receipt-freeness problem and can be restricted over the domain D. In that sense, the next proposition relates the two privacy properties.

Proposition 4.8. *Let α be combinatoric, struct and concr two frames with domain D, V an arbitrary coerced voter. Then, if the receipt-freeness is holding for voter V, then the voting privacy is holding:*

$$RcpFree(D, \alpha, struct, concr, V) \models MsgAna(D, \alpha, struct, concr)\,.$$

It is then sufficient to prove receipt-freeness to prove plain voting privacy.

4.2.2 Receipt-freeness in the current state

FOO'92 does not satisfy receipt-freeness as shown in [DKR09], and even though our notion of receipt-freeness is defined differently, it agrees with their results. FOO'92 serves well anyway for illustration: in the final state S that we have considered before (where the intruder has seen only the final bulletin board), receipt-freeness *does* hold as we now show.

Example 4.2. *Let us first continue with Example 4.1. In the intruder's hypothesis θ_I that we considered, the intruder assumes Dan (i.e., V_1) has voted* 0*, but he actually voted* 1 *(see θ_0). Dan can however point to a vote that is consistent with θ_I, namely the second entry on the bulletin board, and claim it to be his vote. While the intruder may have doubts about Dan's story, he just* cannot rule out *that Dan speaks the truth.*

Let us first consider S, augment it to a receipt-freeness problem with respect to a voter Dan, and show that receipt-freeness actually holds in this state. We first need to define what the knowledge of Dan is. The structural information is very similar to the intruder's knowledge that consists of the published information (the bulletin board and the public keys); additionally Dan also knows his private key, his own vote, his own commitment value and his blinding factor. We did not include the blinded message as it can be reconstructed using the blinding factor.

$$\begin{aligned} struct_{Dan} = \{\!|\, & d_0 \mapsto \mathsf{pub}(A), d_1 \mapsto \mathsf{pub}(V_1), \ldots, d_n \mapsto \mathsf{pub}(V_N), \\ & d_{N+1} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[1]], r[\pi[1]])), \ldots, \\ & d_{2N} \mapsto \mathsf{sign}(\mathsf{priv}(A), \mathsf{commit}(v[\pi[N]], r[\pi[N]])), \\ & d_{2N+1} \mapsto r[\pi[1]], \ldots, d_{3N} \mapsto r[\pi[N]], \\ & d_{3N+1} \mapsto \mathsf{priv}(Dan), d_{3N+2} \mapsto v[1], \\ & d_{3N+3} \mapsto r[1], d_{3N+4} \mapsto b_1 \,|\!\} \end{aligned}$$

The concrete frame is again obtained by replacing $\pi[x]$ by $\pi_0(x)$, $v[x]$ by $\theta_0(v_x)$ and $r[x]$ by r_x. The formula α is the same for receipt-freeness than for voting

privacy, i.e., the intruder still is not supposed to find out anything more than the published result of the election (in particular not what Dan has voted). However, he has more information in β due to the story that Dan gives to the intruder as part of the receipt-freeness definition:

$$\beta_{RF} \equiv \bigwedge_{i=1}^{N} v[i] = v_i \wedge r[i] = r_i \wedge \mathit{RcpFree}(D, \alpha, \mathit{struct}, \mathit{concr}, \mathit{Dan})$$

When it comes to crafting his story for the public values, Dan has no choice but to tell the truth. As the intruder knows Dan's public key, Dan also has to tell the truth for his private key. For his blinding factor, he may also use the truth as the intruder has not witnessed the exchange with the administrator. For d_{3N+2} (the actual vote) and d_{3N+3} (the commitment value), Dan needs to adapt his story to what the intruder "wants to hear", i.e., to a given θ_I (and π_I). Observe at this point the order of quantification here: we want to show that every model $\theta_I \models \alpha$ can be extended to a model $\mathcal{I} \models \beta_{RF}$ where β_{RF} entails an existential quantifier for Dan's story. So, we have to show how, given θ_I, we can construct $\mathcal{I}$ and a value for the recipes of the story $s_1, \ldots, s_l$ that satisfies all conditions. We take exactly the same construction for $\mathcal{I}$ (depending on θ_I) that we used for state S, i.e., using the discrepancy ρ between the intruder hypothesis θ_I and the reality θ_0 (i.e., such that $\theta_0(v_{\rho(i)}) = \theta_I(v_i)$) for interpreting $\pi[\cdot]$, namely as the permutation $\pi_I = \rho^{-1} \cdot \pi_0$. It is sufficient to show that Dan can make his story consistent with this interpretation, namely by pointing to the vote $\rho(1)$ as being his own vote. Let $d_{N+\rho(1)}$ and $d_{2N+\rho(1)}$ be the indices in Dan's knowledge for the signed commitment and commitment values on the bulletin board at position $\rho(1)$. He can claim this entry by choosing:

$$s_{3N+2} = \mathsf{open}(\mathsf{retrieve}(d_{N+\rho(1)}), d_{2N+\rho(1)}) \text{ and } s_{3N+3} = d_{2N+\rho(1)}$$

For all other values s_i, Dan tells the truth, i.e. $s_i = d_i$. With this, we can conclude:

Theorem 4.9. *(α, β_{RF})-privacy holds, i.e., receipt-freeness holds in S.*

Proof. The idea once again here is to prove that for all $\theta_I \models \alpha$, $\mathcal{I}(\mathit{struct}) = \mathcal{I}(\mathit{concr})$. We extend the proof of Theorem 4.5. *gen* is extended to the domain $D \cup D_{Dan}$. The frames *struct* and *concr* are also extended to the new domain as explained with the knowledge of Dan. We already described Dan's strategy for lying. By definition, $\mathcal{I}(v[1]) = [\theta_I(v_1)]_{\approx}$ and $\mathcal{I}(r[1]) = [r_{\rho(1)}]_{\approx}$

Since $\mathcal{I}(\mathit{concr}[d_{3N+2}]) = \mathcal{I}(\mathit{concr}_{Dan}[s_{3N+2}]) = [\theta_0(v_{\rho(1)})]_{\approx} = [\theta_I(v_1)]_{\approx}$ and $\mathcal{I}(\mathit{concr}[d_{3N+3}]) = \mathcal{I}(\mathit{concr}_{Dan}[s_{3N+3}]) = [r_{\rho(1)}]_{\approx}$ by construction, we still have

$\mathcal{I}(struct) = \mathcal{I}(concr)$. Thus, in the augmented state S, receipt-freeness holds. □

One may argue that the choice of s_{3N+2} and s_{3N+3} is hardly a *strategy* for Dan, since the choice is based on the permutation ρ (that neither Dan nor the intruder would know), but formally that is fine since the existential quantifier over the s_i only requires that there *is* a recipe that works, and thus our construction is just the simplest way to conduct the proof of receipt-freeness. Dan can choose any vote on the bulletin board that matches the intruder's expectation for Dan $\theta_I(v_1)$.

The aspect of strategy becomes more relevant if we consider the case that more than one voter is bribed by the intruder, because the intruder knows that some agent is lying if more than one points to the same vote. This becomes an issue when the intruder has bribed a significant part of the voters, which may be possible when a vote is held amongst a small consortium. If the bribed voters have no way to "coordinate" their story, the risk of a collision (that reveals the lie) comes into play. For instance, suppose there are 100 voters, and 40 voted yes. If the intruder has bribed 20 of them, there is a substantial chance that two or more of them point to the same vote if they cannot coordinate their story.

We observe that our definition of receipt-freeness is independent of what the intruder actually wants: we actually have formalized that agents vote however they want and we prove that they can get away with lying—however only with respect to models of α. If the intruder has bribed more voters than actually want to vote for the intruder's preferred choice, then the expected outcome is not a model of α (since the result is not compatible with all bribed people having voted the way the intruder wants). Both this and the previous issue (of coordination) are problems that arise when a significant part of the votership is bribed: they may be coerced into voting what the intruder wants out of fear not to get away with lying after all. These are the boundaries where a *possibilistic* approach like classical (α, β)-privacy makes sense and where probabilities and behavior models would be needed. We see it as a strong point for the declarativity of (α, β)-privacy that such subtle points become clearly visible from the formalization and discussion of examples. We extend in the next chapter (α, β)-privacy with probabilities.

4.2.3 Violation of Receipt-Freeness in FOO'92

To see the problems of FOO'92 with receipt-freeness, let us consider just the state after the third phase of the protocol. In this case, the bulletin board con-

tains all the ballots (the signed commitments) but the commitment secrets have not yet been revealed. In this state receipt-freeness does not hold: Unopened commitments violate receipt freeness, since the creator of the commitment is in a unique position to prove authorship to the intruder (by revealing the commitment secret). Effectively, this allows the intruder to bribe agents for obtaining the commitment secrets, and this is captured by our notion of receipt freeness. This is in particular relevant since voters could refuse to make the last step (the protocol cannot force them, since, by construction, one cannot see who the missing voters are).

While it is intuitively clear that receipt-freeness is violated in this intermediate step, let us prove that it is violated according to our formal definition. Consider the bulletin board without the commitments open, i.e., the same frame as in state S but removing the elements $m_{2N+1}, \ldots, m_{3N}$ (the commitment values). Since in this case, the result has not been published yet, we have here $\alpha \equiv v_1, \ldots, v_N \in \{0, 1\}$, i.e., the intruder knows nothing more than there are N binary votes in the game.

The knowledge of the coerced voter Dan is the same as in the previous subsection, except for removing the entries $d_{2N+1}, \ldots, d_{3N}$ which contain the $r[\pi[i]]$ that have not yet been published at this point, of course. The intruder again asks Dan to reveal his knowledge as before, which entails that Dan must claim some vote on the bulletin board to be his own and present a fitting commitment secret, namely $struct_{Dan}[d_{3N+3}] = r[1]$. Thus the only consistent story that Dan can give for this value is the truth: $s_{3N+3} = d_{3N+3}$. That in turn is only consistent with a given intruder hypothesis $\theta_I \models \alpha$ if $\theta_I(v_1) = \theta_0(v_1)$, i.e., it rules out any model that does not state Dan's vote correctly. Thus, at this point, Dan has *proved* to the intruder what he voted.

In fact, this demonstrates how our notion of receipt-freeness is connected to voting secrecy, namely whether the information given by Dan proves anything to the intruder, i.e., whether it allows him to rule out any model θ_I of α. Note that this is a very fine notion: receipt-freeness would be violated even in a state where Dan cannot precisely prove what he voted, but only giving the intruder enough information to rule out *some* model of α.

4.3 Related work

This work is based on the framework of (α, β)-privacy [MV19], which is in turn based on Herbrand logic [HG06]. As a variant of First-Order Logic, using the ground terms of uninterpreted function symbols as a universe, Herbrand

logic is very expressive, e.g., it can axiomatize natural number arithmetic. The main idea of (α, β)-privacy is to depart from the most popular approach of specifying privacy as bisimilarity of pairs of processes as in [ABF18; BAF08; CRZ07; DRS08]. Instead, we define privacy as a reachability problem of states, where each state is characterized by (at least) two formulae, namely α giving the public high-level information (like a voting result), and β containing all observations that the intruder could make.

While [MV19] has already defined voting secrecy, this work gives the first adaption of (α, β)-privacy to a real-world voting protocol, namely FOO'92 [FOO92]. Another core contribution of this chapter is the formalization of receipt-freeness, namely as a refinement of standard voting secrecy. Here, the high-level information α remains the same (i.e., the same information must be kept private), but the intruder gets extra observations as part of β through the interaction with a voter Dan. The most similar work is [DKR09] where voting privacy, receipt-freeness and coercion-resistance have been expressed with observational equivalence (see also [Ara+17]). The formalization of these properties rely on labeled bisimilarity of two processes, also proving a hierarchy between these goals. We believe that our formalization in (α, β)-privacy is more declarative and intuitive, due to its model-theoretic formulation. An interesting question for future work is how the two approaches compare, i.e., whether one can captures anything as an attack that the other does not. If they turned out to be equivalent in some sense instead, then this would indicate that the "right" concept has been hit.

Another question is automation. There are several fragments of bisimilarity for which automation is being developed. However some protocols, even the relatively simple FOO'92, are hard to analyze fully automatically: for instance, [DKR09] is at the high-level a manual proof, reducing the problem to a static equivalence of two frames (which is then automated). Only in the recent paper [BS18] a fully automatic analysis of FOO'92 is given. Our focus on a declarative formalization rather than automation concerns allows often for very simple proofs, e.g., in FOO'92 in S, which basically amounts to finding a fitting interpretation for a permutation. This is exactly how one may want to prove such a property manually or in a proof assistant like Isabelle or Coq.

4.4 Conclusion

(α, β)-privacy was introduced as a simple and declarative way to specify privacy goals and reason about them. We present here the first major use-case using this framework. This use-case illustrates the refined voting privacy goal that

we have defined in this chapter. Indeed, we showed how for any model θ of α, we could step by step construct a model $\mathcal{I}$ of β. On top of this voting privacy property, we defined a new property: receipt-freeness. We showed that receipt-freeness implies voting privacy. We illustrated these properties for a voting system, but both privacy and receipt-freeness are actually relevant to a variety of areas, for instance healthcare privacy [DJP12]. Indeed, prescriptions by a medial doctor have similar requirements regarding privacy and even receipt-freeness: for instance, we want to prevent that a doctor could be coerced by a pharmaceutical company to prescribe specific medication, which is actually a receipt-freeness problem.

We are currently investigating coercion resistance as a stronger variant of receipt-freeness, where the intruder can initially determine values for the coerced voter to use. To counter such attacks, one needs protocols with a different setup than FOO'92, allowing re-voting. This also requires to formalize more details about the underlying transition system than we did in this chapter, including how the intruder can take a more active part in the protocol. In fact, it is part of ongoing work to provide languages, proof strategies and potentially automated tools for specifying and verifying transition systems with (α, β)-privacy. The idea is here that the formula β can be automatically derived from what happens (like message exchanges) and that only α needs to be specified by the modeler, namely indicating at which point which information is deliberately released. We extend (α, β)-privacy with transition system in the next chapter.

Chapter 5

Privacy As Reachability

There are two further open problems with (α, β)-privacy, which we tackle in this chapter.

Problem 1. The main difficulty in reasoning about privacy with trace equivalence is that one needs to consider two possible worlds: for every step the first system can make, one has to show that the other system can make a similar step so that they are still indistinguishable (and so are the executed steps). Many works tame this difficulty by making the processes just differ in some message-subterms, so everything except these subterms is equal. One can obtain a verification question that is close to a reachability problem, which drastically reduces the range of protocols that can be considered.

What distinguishes (α, β)-privacy from trace equivalence is that it considers *one* possible world rather than two. (α, β)-privacy is until now only a static approach that does not reason about the development of a system, like the influence that the actions of an intruder can have on a system, and thus does not solve Problem 1 . . . yet.

The *first main contribution* of this chapter is to lift (α, β)-privacy from a static approach to a dynamic one. We define a transaction-process formalism for distributed systems that can exchange cryptographic messages (in a black-box cryptography model). Our formalism

- includes privacy variables that can be non-deterministically chosen from finite domains (e.g., the candidates in a voting protocol),
- can work also with long-term mutable states (e.g., modeling a hash-key chain), and
- allows one to specify the consciously released information (e.g., the number of cast votes and the result).

We define *dynamic* (α, β)*-privacy* that holds if (α, β)-privacy holds in every

state of the transition system. Hence, every state is an (α, β)-privacy problem, i.e., a pure reachability problem that supports a wide variety of privacy goals.

This does not solve all the challenges of automation: (i) (α, β)-privacy is in general undecidable, but for most reasonable protocols (including the algebraic model of cryptography) it is decidable, as it boils down to a static equivalence of frames; (ii) the set of reachable states is infinite. Symbolic and abstract interpretation methods still need to be developed.

We argue though that this approach is very helpful for manual analysis, because it is a novel view of privacy that allows us to characterize the reachable states in a declarative logical way, and analyze the dynamic (α, β)-privacy question for them. As a topical case study we consider the core of the privacy-preserving proximity tracing system DP-3T [Vau20]. We discover counter-examples for dynamic (α, β)-privacy, i.e., the intruder can make more deductions about the honest agents than released in α. Step by step, including more details in α, we obtain a characterization of all information that the system actually discloses, and then prove dynamic (α, β)-privacy. This can be helpful to understand the actual privacy impact of a system, and is also an answer to Problem 1.

Problem 2. Many approaches (e.g., quantitative information flow, differential privacy, etc.) reason about privacy by considering quantitative aspects and probabilities. Trace equivalence approaches are instead purely qualitative and possibilistic, and so is (dynamic) (α, β)-privacy; this is appropriate for many scenarios, but we give examples where probability distributions play a crucial role (i.e., in a purely possibilistic setting, there is no attack, but with probabilities there is).

As *second main contribution* of this chapter, we give a *conservative* extension of (dynamic) (α, β)-privacy by probabilistic variables. As for non-deterministic variables (used when probabilities are irrelevant or when the intruder does not know the distribution), probabilistic variables can be sampled from a finite domain with a probability distribution, which may depend on probabilistic variables that were chosen earlier.

As proof-of-concept, we consider some simple examples, and then we show that a well-known problem of vote copying (e.g., in Helios, where a dishonest voter can copy an honest voter's vote) can be analyzed with probabilistic (α, β)-privacy in a new light: one can observe the influence on the distribution by the dishonest votes, where possibilistic models would not allow the deduction. The intruder almost becomes an empirical scientist who needs to decide when the distortion of the probabilities is significant to deduce how a particular voter voted. Hence, our approach successfully tackles Problem 2.

We prove two theorems for (dynamic) probabilistic (α, β)-privacy. We define a notion of *extensibility*, which says that β does not exclude choices of probabilistic variables. Theorem 5.10 says that if we prove *possibilistic* (α, β)-privacy for an extensible pair (α, β), then *probabilistic* (α, β)-privacy holds as well. Theorem 5.11 proves stability under background knowledge: if the intruder has additional background information α_0 (e.g., knowledge about the distribution of votes or particular voters), then in any state with an extensible pair (α, β), probabilistic $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy still holds; the intruder does not learn more than what he already knew and what we deliberately release.

As a *third contribution* of this chapter, we formalize the relationship between our approach and trace equivalence (Theorems 5.15 and 5.16).

In §5.1, we lift (α, β)-privacy from static to dynamic. In §5.2, we give a conservative extension of (dynamic) (α, β)-privacy with probabilities. We formalize the DP-3T protocol with dynamic possibilistic (α, β)-privacy in §5.3, we show how to formalize voting privacy goals with dynamic (α, β)-privacy in §5.4, and we prove the relationship with trace equivalence in §5.5, and discuss that with information flow in §5.6. Finally, in §5.7, we discuss future work.

5.1 Transition Systems for Alpha-Beta-privacy

We lift the definition of static (α, β)-privacy to a dynamic one with transition systems. In §5.1.1, we describe the syntax of a protocol specification, notably the syntax of processes. We give the operational semantics for transition systems in §5.1.2 and define the state with, amongst other things, the following formulae: the *payload formula* α, the *technical information formula* β and the *truth formula* γ. We also define δ, which is a sequence of *conditional updates* on the cells, and η, which is a *probability decision tree* for the random variables. In §5.1.3, we show how to derive a legitimate linkability attack on the OSK protocol.

5.1.1 Syntax

We consider a number a *transaction processes* and a number of families of *memory cells*, which allow us to model the stateful nature of some protocols. These cells can be used, for instance, to store the status of a key (e.g., valid or revoked).

In the processes, we talk about privacy variables of two sorts: *non-deterministic*

or *random*. Each of them has a domain $D = \{c_1, \ldots, c_n\}$, where $c_1, \ldots, c_n$ are constants, i.e., a variable will be instantiated to one of these values. A random variable can also use $D_{prob} = \{c_1 : p_1, \ldots, c_n : p_n\}$, where the p_i are probabilities s.t. they form a distribution, i.e., $p_i \in [0, 1]$ and $\sum_{i=1}^{n} p_i = 1$. We might omit the probabilities in D_{prob} when the distribution is uniform, and if only some of the p_i are made precise, then the rest of them are uniformly distributed amongst the remaining probability weight. We consider only finite domains. This is not a restriction, since it is possible to leave the size of the model as a parameter in all definitions.

Definition 5.1 (Syntax)**.** *A* protocol specification *consists of:*

- *a number of families of* memory cells*, e.g.,* $\mathsf{cell}(\cdot)$*, together with an initial value which is a ground context* $k([\cdot])$*, so that initially* $\mathsf{cell}(t) = k([t])$*,*
- *a number of* transaction processes *of the form* $\mathcal{P}_l$*, where* $\mathcal{P}_l$ *is a left process according to the syntax below, and*
- *an initial state (see Definition 5.3), containing, e.g., domain specific axioms in the formulae* α *and* β *(see preliminaries).*

We define left processes *and* right processes *as follows:*

$$
\begin{array}{lll}
\mathcal{P}_l ::= & \mathsf{mode}\ x \in D.\mathcal{P}_l & \mathcal{P}_r ::= \mathsf{snd}(t).\mathcal{P}_r \\
\mid & \mathsf{mode}\ x \leftarrow D_{prob}.\mathcal{P}_l & \mid \mathsf{cell}(s) := t.P_r \\
\mid & \mathsf{rcv}(x).\mathcal{P}_l & \mid \mathsf{mode}\ \phi.\mathcal{P}_r \\
\mid & x := \mathsf{cell}(s).\mathcal{P}_l & \mid 0 \\
\mid & \mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l & \\
\mid & \nu\overline{N}.\mathcal{P}_r &
\end{array}
$$

where x *ranges over variables;* mode *is either* $\star$ *or* $\diamond$*,* D *is the finite domain of a non-deterministic variable;* D_{prob} *is the finite domain of a random variable;* s *and* t *range over terms,* cell *over families of memory cells, and* ϕ *over Herbrand formulae; and* $\overline{N}$ *is a set of fresh variables, i.e., that do not occur elsewhere. In* ϕ *formulae, we also allow the symbol* $\mathcal{I}$ *around terms, e.g.,* $x \doteq \mathcal{I}(x)$*. This will refer during execution to the true interpretation of the variables as we define below.*

"$\mathsf{mode}\ x \in D$" is the picking of a non-deterministic variable, "$\mathsf{mode}\ x \leftarrow D_{prob}$" is the sampling of a random variable, and both are only released in β if mode is $\diamond$, and additionally in α if mode is $\star$. "$\mathsf{rcv}(x)$" is a standard message input, where the variable x is replaced with an actual received message. "$x := \mathsf{cell}(s)$" is a cell read where x is replaced by a value stored in the memory cell. The conditional "$\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l$" is standard. "$\nu\overline{N}.\mathcal{P}_r$" creates a sequence of

fresh variables; it does not recur on $\mathcal{P}_l$ but on $\mathcal{P}_r$, meaning that one can have new variables only once, directly before entering the right process. "$\mathsf{snd}(t)$" is a standard message output. "$\mathsf{cell}(s) := t$" is a cell write that stores a term in the cell. "$\mathsf{mode}\ \phi$" releases a formula in α of the current state if mode is $\star$ and in γ if mode is $\diamond$. Finally, "0" is the null process.

We may write "$\mathsf{let}\ x = t$" for the substitution of all following occurrences of x by t. Another syntactic sugar concerns parsing of messages. For many (cryptographic) operators we may have a corresponding *destructor* and *verifier*, e.g., we often use the public functions $\mathsf{pair}/2$, $\mathsf{proj}_i/1$ and $\mathsf{vpair}/1$ with the properties $\mathsf{proj}_i(\mathsf{pair}(t_1, t_2)) \approx t_i$, and $\mathsf{vpair}(\mathsf{pair}(t_1, t_2)) \approx \mathsf{true}$. More generally, let f/n be a destructor (like the proj_i) and v/n a corresponding verifier (like vpair); then we may write "$\mathsf{try}\ t = f(t_1, \ldots, t_n)\ \mathsf{in}\ \mathcal{P}_1\ \mathsf{catch}\ \mathcal{P}_2$" in lieu of "$\mathsf{if}\ v(t_1, \ldots, t_n) \doteq \mathsf{true}\ \mathsf{then}\ \mathsf{let}\ t = f(t_1, \ldots, t_n).\mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2$". In the try construct, t is substituted in $\mathcal{P}_1$ and, as for the else branch in the conditional construct, we may omit the catch branch when $\mathcal{P}_2$ is the null process. Let us now look at a first example.

Example 5.1 (Basic Hash)**.** *As a first example, we consider the* Basic Hash protocol *[BCH10]: a reader can access a database of authorized tags that carry a mutable state. We consider n tags in the domain $\mathsf{Tags} = \{t_1, \ldots, t_n\}$. Let $\mathsf{sk}/1$ and $h/2$ be private functions. Each tag T has an immutable secret key $\mathsf{sk}(T)$. Let $\mathsf{pair}/2$, $\mathsf{vpair}/1$ and $\mathsf{proj}_i/1$ be public functions as before. The tag sends messages of the form of a pair of a fresh nonce and the hash of the same nonce and its secret key.*

Tag
$\star\ T \in \mathsf{Tags}.$
$\nu N.\mathsf{snd}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))).0$

When the reader receives a message from a tag T, it has first to figure out who T is by trying all known keys $\mathsf{sk}(T)$ of any token T, almost like a guessing attack. In order not to have to describe this procedure as transactions (it is included in the intruder model if he knows any keys), we simply define two special private functions for the reader ($\mathsf{extract}/1$ and $\mathsf{vextract}/1$) that check if a message is valid and extract T from it such that $\mathsf{extract}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))) \approx \mathsf{sk}(T)$ and $\mathsf{vextract}(\mathsf{pair}(N, h(\mathsf{sk}(T), N))) \approx \mathsf{true}$.

Definition 5.2 (Requirements on Processes)**.** *We require that α formulae are over Σ_0 and contain only variables that were released in α. In "$\mathsf{mode}\ x \in D.\mathcal{P}_l$", "$\mathsf{mode}\ x \leftarrow D_{prob}.\mathcal{P}_l$", "$\mathsf{rcv}(x).\mathcal{P}_l$" and "$x := \mathsf{cell}(s).\mathcal{P}_l$", we require that x cannot be instantiated twice, i.e., $\mathcal{P}_l$ contains neither "$\mathsf{mode}\ x \in D'$", nor "$\mathsf{mode}\ x \leftarrow D'_{prob}$", nor "$\mathsf{rcv}(x)$", nor "$x := \mathsf{cell}(s')$". We also require that*

Reader

$\mathsf{rcv}(t).$
$\mathsf{try}\ R = \mathsf{extract}(t)\ \mathsf{in}$
$\quad \mathsf{snd}(\mathsf{ok}).0$

in different branches of conditionals, the same random and non-deterministic variables are chosen in the same order and from the same set of values, and the ordering with receive steps is also the same. This is formalized by the following function that is only defined when the requirements are met:

$$\begin{aligned}
varseq(\mathsf{mode}\ x \in D.\mathcal{P}_l) &= \mathsf{mode}\ x \in D.varseq(\mathcal{P}_l)\\
varseq(\mathsf{mode}\ x \leftarrow D_{prob}.\mathcal{P}_l) &= \mathsf{mode}\ x \leftarrow D.varseq(\mathcal{P}_l)\\
varseq(\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_1 \mathsf{else}\ \mathcal{P}_2) &= varseq(\mathcal{P}_1)\\
\mathsf{if}\ varseq(\mathcal{P}_1) = varseq(\mathcal{P}_2)\ &\textit{and undefined otherwise}\\
varseq(\mathsf{rcv}(t).\mathcal{P}_l) &= \mathsf{rcv}(t).varseq(\mathcal{P}_l)\\
varseq(_.\mathcal{P}_r) &= varseq(\mathcal{P}_r)\\
varseq(0) &= 0
\end{aligned}$$

Note that in the case for D_{prob}, the right-hand side uses D, which is supposed to mean drop the probabilities from the domain: two branches are allowed to assign different probabilities to the elements of the domain, but it has to be the same domain. We also require that when a random variable y is sampled inside a conditional "$\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_l\ \mathsf{else}\ \mathcal{P}_l$*", ϕ can only contain conjunctions of the form $x \doteq c_i$, where x is a random variable that has previously been sampled, and if y is an α variable, then also x must be.*

Finally, we require that every transaction in a protocol specification is a closed process*, i.e., it has no free variables and the* binding occurrence *of a variable is the first occurrence where in the context it is not free (so further occurrences do not open a new scope):*

$$\begin{aligned}
fv(\mathsf{mode}\ x \in D.\mathcal{P}_l) &= fv(\mathcal{P}_l) \setminus \{x\}\\
fv(\mathsf{mode}\ x \leftarrow D_{prob}.\mathcal{P}_l) &= fv(\mathcal{P}_l) \setminus \{x\}\\
fv(\mathsf{rcv}(x).\mathcal{P}_l) &= fv(\mathcal{P}_l) \setminus \{x\}\\
fv(x \coloneqq \mathsf{cell}(s).\mathcal{P}_l) &= (fv(s) \cup fv(\mathcal{P}_l)) \setminus \{x\}\\
fv(\mathsf{if}\ \phi\ \mathsf{then}\ \mathcal{P}_1 \mathsf{else}\ \mathcal{P}_2) &= fv(\phi) \cup fv(\mathcal{P}_1) \cup fv(\mathcal{P}_2)\\
fv(\nu\overline{N}.\mathcal{P}_r) &= fv(\mathcal{P}_r) \setminus \overline{N}
\end{aligned}$$

, and the free variables of a right process are all variables occurring in it.

5.1.2 Operational Semantics

We describe the operational semantics that lifts the definition of static (α, β)-privacy to a dynamic one with transition systems. We define *possibilistic dynamic* (α, β)*-privacy*, which intuitively holds if (α, β)-privacy holds in every state of the transition system. Let us start by defining the states of our transition system, where, for simplicity, we already include the tree η that we will need in Section 5.2, in which we discuss probabilities explicitly.

Definition 5.3 (State)**.** *A* state *is a tuple* $(\alpha, \beta, \gamma, \delta, \eta)$*, where:*

- *formula* α *over* Σ_0 *is the* released information*,*
- *formula* β *over* Σ *is the* technical information *available to the intruder, such that* β *is consistent and entails* α *(thus also* α *is consistent and* $fv(\alpha) \subseteq fv(\beta)$[1]*),*
- *formula* γ *over* Σ_0 *is the* truth*, which is true for exactly one model with respect to the free variables of* α *and* Σ_0*, and* $\gamma \wedge \beta$ *is consistent,*
- δ *is a sequence of* conditional updates *of the form* $\mathsf{cell}(s) \coloneqq t$ *if* ϕ*, where* s *and* t *are terms and* ϕ *is a formula over* Σ*, and its free variables are a subset of the free variables of* α*, and*
- η *is a* probability decision tree*, which, for a sequence of random variables* $x_1, \ldots, x_n$*, has* $n + 1$ *levels where every inner node on the* i*-th level is labeled* x_i*, and the leaves on level* $n + 1$ *are unlabeled. To each variable* x_i *we associate a domain* $D_i = \{c_{l_1}, \ldots, c_{l_i}\}$*. Every* x_i *node in* η *has* l_i *children, and every branch from a node to its children is labeled with one of the* c_{l_i} *and a probability, so that the probabilities under a variable sum up to* 1*. In the following, we will use* η *in such a way that the first* k *levels are the random variables of* α*, and the remaining levels the random variables of* β*.*[2] *If there are no probabilistic variables, we may omit the tree* η*.*

The formulae α and β play the same roles than in the previous chapter. To define our transition system, we introduce the formula γ that represents the "truth", i.e., the real execution of a protocol. For instance, for a voting protocol, α may contain $v_i \in \{0, 1\}$ (i.e., that vote v_i is one of these values), β may contain cryptographic messages that contain v_i, and γ may contain $v_i = 1$, i.e., what

[1] [MV19] only allowed that $fv(\alpha) = fv(\beta)$, but our constructions do not require it.

[2] Note that the tree has exponential size in the number of variables, so for implementation in automated tools, a more efficient representation should be chosen, but for the conceptual level, this is irrelevant.

the vote actually is (and this is not visible to the intruder). The consequences of γ is what really happened, e.g., the result that one can derive from the votes in γ is the true result of the election. The sequence δ represents in a symbolic way all updates that a protocol may have performed on the memory cells: when updates are under a condition, the intruder does not know whether they where executed, so each update operation in δ come with a condition ϕ; these entries in general contain variables when the intruder does not know the concrete values. We describe η and the probabilistic mechanisms in Section 5.2.

During the execution of a transaction, the intruder will in general not know what exactly is happening, in particular in a conditional, it will not be generally clear which branch has been taken. Therefore, we define now the notion of possibilities that represent all possible choices, what that means for the condition and the structure of messages. One of the possibilities is marked to indicate which one is actually true.

Definition 5.4 (Possibility, configuration)**.** *A* possibility $(P, \phi, struct)$ *consists of a process P, a formula ϕ and a frame struct representing the structural knowledge attached to this process P. A* configuration *is a pair $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ is a state and $\mathcal{P}$ is a non-empty finite set of possibilities s.t.:*

- *$fv(\mathcal{P})$ is a subset of the free variables of $\mathcal{S}$,*
- *exactly one element of $\mathcal{P}$ is marked as the* actual possibility*, which we depict by underlining that element,*
- *the formulae $\phi_1, \ldots, \phi_n$ of $\mathcal{P}$ are mutually exclusive (i.e., $\models \neg\phi_i \vee \neg\phi_j$ for $i \neq j$) and β implies their disjunction (i.e., $\beta \models \phi_1 \vee \ldots \vee \phi_n$), and*
- *$\beta \wedge \gamma \models \phi$ for the condition ϕ of the marked possibility.*

In a state, we can start the execution of any transaction from the protocol description as follows:

Definition 5.5 (Initial configuration)**.** *Consider a configuration $(\mathcal{S}, \mathcal{P})$, a transaction process $\mathcal{P}_l$, a substitution θ that substitutes the fresh variables $\overline{N}$ (from a $\nu\overline{N}.\mathcal{P}_r$ specification) with fresh constants from $\Sigma \backslash \Sigma_0$ that do not occur elsewhere in the description or in $(\mathcal{S}, \mathcal{P})$, and that replaces all other variables with fresh variables that do not occur elsewhere in the description or in $(\mathcal{S}, \mathcal{P})$. The* initial configuration *of $\mathcal{P}_l$ w.r.t. $(\mathcal{S}, \mathcal{P})$ and θ is $(\mathcal{S}', \{(\theta(\mathcal{P}_l), \phi, struct) \mid (0, \phi, struct) \in \mathcal{P}\})$.*

From this initial configuration, we can get to a new state (or several states) by the following normalization and evaluation rules, basically working off the steps

of the process $\mathcal{P}_l$. We first define these rules and then give a larger example in Section 5.1.3.

5.1.2.1 Normalization Rules

We have six normalization rules for a configuration: redundancy, cell reads, conditional, cell write, redundant entries in δ, release in α. Recall that in a configuration, we have always one possibility being marked, which we denote by underlining it; in the following rules however, if no possibility is underlined, then the rule applies for all possibilities, no matter if marked or not.

Redundancy We can always remove redundant possibilities when the intruder knows that a condition is inconsistent with β (this can never happen to the marked possibility, as the truth is always consistent with β):

$$\{(P, \phi, \mathit{struct})\} \cup \mathcal{P} \implies \mathcal{P} \text{ if } \beta \models \neg\phi$$

Cell Reads Let $C[\cdot]$ be the initial state of cell, and let the cell operations in the current state $\mathcal{S}$ be $\mathsf{cell}(s_1) \coloneqq t_1$ if ϕ_1, …, $\mathsf{cell}(s_n) \coloneqq t_n$ if ϕ_n. Then, every configuration that starts with the reading of a memory cell is normalized via:

$$\begin{aligned}
&\{(x \coloneqq \mathsf{cell}(s).P_l, \phi, \mathit{struct})\} \cup \mathcal{P} \implies \\
&\{(\mathsf{if}\ s = s_n \wedge \phi_n\ \mathsf{then\ let}\ x \coloneqq t_n.P_l\ \mathsf{else} \\
&\quad \mathsf{if}\ s = s_{n-1} \wedge \phi_{n-1}\ \mathsf{then\ let}\ x \coloneqq t_{n-1}.P_l\ \mathsf{else} \\
&\quad \ldots \\
&\quad \mathsf{if}\ s = s_1 \wedge \phi_1\ \mathsf{then\ let}\ x \coloneqq t_1.P_l\ \mathsf{else} \\
&\quad \mathsf{let}\ x \coloneqq C[s].P_l, \phi, \mathit{struct})\} \cup \mathcal{P}
\end{aligned}$$

The same rule holds if the possibility is marked (and then the transformed possibility is the marked one).

Conditional A conditional process is normalized via:

$$\begin{aligned}
&\{(\mathsf{if}\ \psi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2, \phi, \mathit{struct})\} \cup \mathcal{P} \implies \\
&\{(\mathcal{P}_1, \phi \wedge \psi, \mathit{struct}), (\mathcal{P}_2, \phi \wedge \neg\psi, \mathit{struct})\} \cup \mathcal{P}
\end{aligned}$$

If the process "$\mathsf{if}\ \psi\ \mathsf{then}\ \mathcal{P}_1\ \mathsf{else}\ \mathcal{P}_2$" is marked, then, by construction, $\beta \wedge \gamma \models \phi$, thus either $\beta \wedge \gamma \models \phi \wedge \psi$ or $\beta \wedge \gamma \models \phi \wedge \neg\psi$. Accordingly, exactly one of the alternatives is marked.

Cell write A cell write process is normalized via:

$$\{(\mathsf{cell}(s) \coloneqq t.P_r, \phi, \mathit{struct})\} \cup \mathcal{P} \implies \{(P_r, \phi, \mathit{struct})\} \cup \mathcal{P}$$

where δ is augmented with the entry $\mathsf{cell}(s) \coloneqq t$ if ϕ. The order of these entries in δ depends on which normalizations are performed first, e.g., if we have $\{(\mathsf{cell}(s_1) \coloneqq t_1.0, \phi_1, \mathit{struct}_1), (\mathsf{cell}(s_2) \coloneqq t_2.0, \phi_2, \mathit{struct}_2)\}$, then normalization yields $\{(0, \phi_1, \mathit{struct}_1), (0, \phi_2, \mathit{struct}_2)\}$. Depending on whether we have started normalizing the first or the second possibility, the resulting δ is either $\delta \equiv \mathsf{cell}(s_1) \coloneqq t_1$ if ϕ_1, $\mathsf{cell}(s_2) \coloneqq t_2$ if ϕ_2 or $\delta \equiv \mathsf{cell}(s_2) \coloneqq t_2$ if ϕ_2, $\mathsf{cell}(s_1) \coloneqq t_1$ if ϕ_1.

However, both orderings are in some sense equivalent, because ϕ_1 and ϕ_2 are mutually exclusive, so at most one of them can happen in any given model $\mathcal{I}$ of β. A similar argument holds for any critical pair of applicable normalization rules, and thus an arbitrary application strategy of the normalization rules may be fixed for the uniqueness of the definition.

Redundant entries in δ An entry $\mathsf{cell}(s) \coloneqq t$ if ϕ can be removed from δ if $\beta \models \neg\phi$.

Release Given a process that wants to release some information ϕ_0, if the possibility is marked then we add it to α if mode is $\star$ or to γ if mode is $\diamond$, otherwise we ignore it:

$$\{\underline{(\mathsf{mode}\ \alpha_0.P_r, \phi, \mathit{struct})}\} \cup \mathcal{P} \implies \{(P_r, \phi, \mathit{struct})\} \cup \mathcal{P}$$

Recall that in process specifications, the formula ϕ_0 may contain subterms of the form $\mathcal{I}(t)$, e.g., $x = \mathcal{I}(x)$. When adding to α or to γ, this subterm must be replaced by the actual value $\mathcal{I}(t)$ where $\mathcal{I}$ is the unique model of γ, i.e., the truth.

5.1.2.2 Evaluation Rules

We call a set of configurations *normalized* if normalization rules have been applied as far as possible. The first step of a normalized set of configurations is either a random sampling or non-deterministic choice, a send or a receive step, or they finished—since all other constructs are acted upon by the normalization rules. The following evaluation rules can produce multiple successor configurations (due to non-deterministic choice or random sampling), and they can produce non-normalized configurations. In this case, before another of the evaluation rules can be taken, the configurations have to be normalized again.

Non-deterministic choice If the first step in the marked process is a non-deterministic choice, then in fact, all processes must start with a non-deterministic choice of the same variable x and from the same domain D. This is because we required that *varseq* is defined and the set of configurations is normalized. In this case, the evaluation is defined as a non-deterministic configuration transition for every $c \in D$ as follows:

$$\begin{aligned}((\alpha,\beta,\gamma,\delta,\eta),\{((\mathsf{mode}\ x \in D.\mathcal{P}_1,\phi_1,\mathit{struct}_1),\ldots,\\ (\mathsf{mode}\ x \in D.\mathcal{P}_n,\phi_n,\mathit{struct}_n))\}) \implies\\ ((\alpha',\beta',\gamma',\delta,\eta),\{(\mathcal{P}_1,\phi_1,\mathit{struct}_1),\ldots,(\mathcal{P}_n,\phi_n,\mathit{struct}_n)\})\end{aligned}$$

where $\alpha' = \alpha \wedge x \in D$ if mode is $\star$ and $\alpha' = \alpha$ if mode is $\diamond$, $\beta' = \beta \wedge x \in D$ and $\gamma' = \gamma \wedge x \doteq c$ for whatever mode.

Random Sampling For the same reason as before, if the first step in the marked process is a random sampling, then all processes must start with a random choice of the same variable x. They may be sampled at different probabilities $D_{prob,1},\ldots,D_{prob,n}$, but the underlying set of elements D is identical. Then, for every $c \in D$ we have a configuration transition as follows:

$$\begin{aligned}((\alpha,\beta,\gamma,\delta,\eta),\{((\mathsf{mode}\ x \leftarrow D_{prob,1}.\mathcal{P}_1,\phi_1,\mathit{struct}_1),\ldots,\\ (\mathsf{mode}\ x \leftarrow D_{prob,n}.\mathcal{P}_n,\phi_n,\mathit{struct}_n))\}) \implies\\ ((\alpha',\beta',\gamma',\delta,\eta'),\{(\mathcal{P}_1,\phi_1,\mathit{struct}_1),\ldots,(\mathcal{P}_n,\phi_n,\mathit{struct}_n)\})\end{aligned}$$

where $\alpha' = \alpha \wedge x \in D$ if $\star$ is specified and $\alpha' = \alpha$ otherwise, $\beta' = \beta \wedge x \in D$ and $\gamma' = \gamma \wedge x \doteq c$ for whatever mode.

The tree η' is obtained from η as follows. First, let us describe the case that x has been sampled when mode was set to $\diamond$. Then, we replace the leaves of η with a new level of nodes labeled x, each of which have $|D|$ leaves as children. The probabilities on the new branches are determined as follows: for every x-labeled node, the path from the root determines an interpretation of all the random variables. We check which of the conditions of $\phi_1,\ldots,\phi_n$ agree with this interpretation. If there is more than one, say ϕ_i and ϕ_j, then $D_{prob,i} = D_{prob,j}$ (due to our requirement that the conditions for a probability distribution for a random variable can only depend on the value of earlier random variables). Thus we can label the children of a given node accordingly.Note that it may happen that no ϕ_i agrees with the node; this is when the respective interpretation has already been excluded by α or β; in this case, the probability distribution is immaterial in the following, we just set it to uniform distribution.

If x has been sampled when mode was set to $\star$, i.e., is an α variable, the construction of η' is slightly different: we introduce the new level below the last α-variable in η, where the children of an x-node in η' are n copies of the subtree

of the corresponding node in η. This is possible since the choice of an α variable by construction can only depend on other α variables.

Example 5.2 (Probability tree)**.** *Consider the simple process that samples in that order a variable x with* mode *set to $\star$, a variable y with* mode *set to $\diamond$ and a variable z with* mode *set to $\star$:*

Example of a probability tree

$$\begin{array}{l}\star\ x \leftarrow \{1\colon \frac{1}{3}, 2\colon \frac{1}{3}, 3\colon \frac{1}{3}\}.\\ \diamond\ y \leftarrow \{1\colon \frac{1}{4}, 2\colon \frac{3}{4}\}.\\ \star\ z \leftarrow \{1\colon \frac{1}{2}, 2\colon \frac{1}{2}\}\end{array}$$

With our evaluation rules, this produces the probability decision tree in Figure 5.1 (note that the nodes with variables z appears before the nodes with variable y because z has been sampled with mode *set to $\star$):*

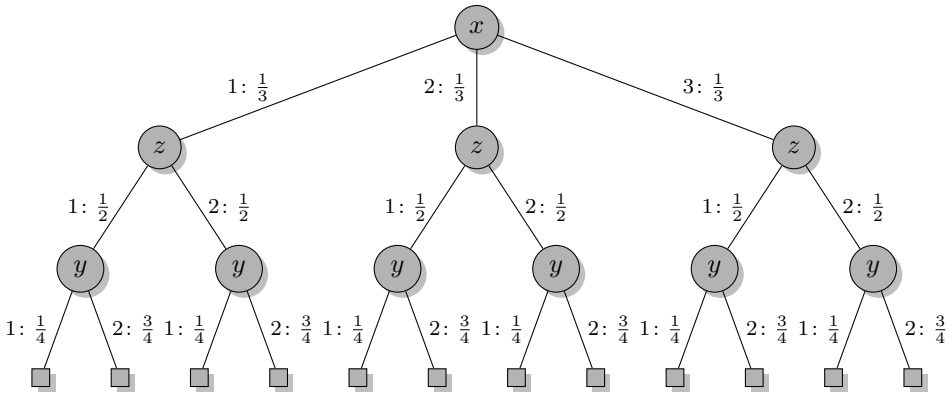

Figure 5.1: Example of a probability tree

To see an another example of an η tree, see Figure 5.4.

Marked process receives Also in this case, if one process starts with a receive, all the others start with a receive as well. Also here, we have several possible state transitions, since the intruder can freely choose a message to send to the processes. Let r be any recipe that the intruder can generate according to β, i.e., $\beta \models gen(r)$. For every such r, we have a configuration transition:

$$\{\underline{(\mathsf{rcv}(x).P_1, \phi_1, struct_1)}, \ldots, (\mathsf{rcv}(x).P_k, \phi_k, struct_k)\} \rightarrow$$
$$\{\underline{(P_1[x \mapsto struct_1[r]], \phi_1, struct_1)}, \ldots, (P_k[x \mapsto struct_k[r]], \phi_k, struct_k)\}$$

Note that our construction requires that in any $\mathsf{rcv}(x).P_k$, x is a variable that did not occur previously in the same process, i.e., we forbid $\mathsf{rcv}(x).\mathsf{rcv}(x).P_k$, as explained in Definition 5.2.

Marked process sends If the marked process sends a message next, this is observable, and all processes that do not send are ruled out. Thus, we have the rule

$$\{\underline{(\mathsf{snd}(m_1).P_1, \phi_1, \mathit{struct}_1)}, \ldots, (\mathsf{snd}(m_k).P_k, \phi_k, \mathit{struct}_k)\} \cup \mathcal{P} \rightarrow$$
$$\{\underline{(P_1, \phi_1, \mathit{struct}_1 \cup \{\!|l \mapsto m_1|\!\})}, \ldots, (P_k, \phi_k, \mathit{struct}_k \cup \{\!|l \mapsto m_k|\!\})\}$$

where l is a fresh label and $\mathcal{P}$ is a set of configurations that are finished, and we augment β by:

$$\phi_1 \vee \ldots \vee \phi_k \wedge \mathit{concr}[l] = \gamma(m_1) \wedge$$
$$\exists i \in \{1, \ldots, k\}.\ \bigvee_{j=1}^{k} i = j \wedge \mathit{struct}[l] = m_j \wedge \phi_j$$

This is because the intruder can now rule out all possibilities in $\mathcal{P}$ and their conditions (so one of the ϕ_i in the remaining processes must be true). Moreover, the intruder knows a priori only that the message they receive, concretely $\gamma(m_1)$, is one the m_i and this is the case iff ϕ_i holds.

Marked process has terminated If the marked process has terminated, then the others have either also terminated or start with a send step (since other cases are already done). If all processes are terminated, we are done, otherwise the intruder can rule out the processes that are not yet done:

$$\{\underline{(0, \phi_1, \mathit{struct}_1)}, \ldots, (0, \phi_k, \mathit{struct}_k)\} \cup \mathcal{P} \rightarrow$$
$$\{\underline{(0, \phi_1, \mathit{struct}_1)}, \ldots, (0, \phi_k, \mathit{struct}_k)\}$$

where $\mathcal{P}$ is a set of configurations that start with a send, and we augment β by $\phi_1 \vee \ldots \vee \phi_k$. In any case, we have thus finished the normalization and evaluation rules, and thus have reached a state.

After defining transition systems, let us define *dynamic* (α, β)*-privacy.* Note that this definition is possibilistic, so we refer to states without regards to η:

Definition 5.6 (Dynamic (α, β)-privacy)**.** *Given a configuration $(\mathcal{S}, \mathcal{P})$, a transaction process $\mathcal{P}_l$, and a substitution θ as in Definition 5.5, the* successor states *are defined as all states reachable from the initial configuration of $\mathcal{P}_l$ using the*

normalization and evaluation rules. The set of reachable states *of a protocol description is the least reflexive transitive closure of this successor relation w.r.t. a given initial state of the specification (the possibilities being initialized with* $(0, \mathsf{true}, \emptyset)$*).*

We say that a transition system satisfies dynamic (α, β)-privacy *iff static* (α, β)*-privacy holds for every reachable state.*

5.1.3 Linkability attack on OSK Protocol

As a second example, we consider the OSK protocol [OSK03]. Consider a finite set of tags $\mathsf{Tags} = \{t_1, \ldots, t_n\}$, and let $h/1$ and $g/1$ be two public functions (modeling one-way functions). Consider also two families of memory cells: one for the tags, $r(\cdot)$, one for the reader, $\mathsf{state}(\cdot)$, whose initial values are both $\mathsf{init}(\cdot)$. Each tag T owns $r(T)$ and the reader owns the entire family $\mathsf{state}(T)$, i.e., T's "database". The tag updates its state $r(T)$ by applying a hash to it at each session and sending out the current key under g. The privacy goal is thus that the intruder cannot find out anything besides the fact that this action is performed by *some* tag $T \in \mathsf{Tags}$, i.e., that he cannot link two or more sessions to the same tag.

Tag
$\star\ T \in \mathsf{Tags}.$
$\mathsf{Key} := r(T).$
$r(T) := h(\mathsf{Key}).$
$\mathsf{snd}(g(\mathsf{Key})).0$

The reader should receive a message of the form $g(h^j(\mathsf{init}(T)))$, and would accept this message, if its own database contains the value $h^i(\mathsf{init}(T))$ for some $i \leq j$ (to prevent replay). Like in Example 5.1, the server has to perform a kind of guessing attack to figure out T and $j - i$. To model this simply we introduce private functions $\mathsf{getT}/1$, $\mathsf{vgetT}/1$, $\mathsf{extract}/2$, $\mathsf{vextract}/2$ with the algebraic properties in Figure 5.2. The getT function extracts the name (if it is a valid message, as checked with vgetT) and $\mathsf{extract}$ extracts the current key (if it is a higher hash than the given key, as checked with $\mathsf{vextract}$). For applying the verifiers, we use the syntactic sugar try again to formulate the reader, who when

$$\begin{aligned}
\mathsf{getT}(g(\mathsf{init}(T))) &\approx \mathsf{init}(T)\\
\mathsf{getT}(g(h(X))) &\approx \mathsf{getT}(g(X))\\
\mathsf{vgetT}(g(\mathsf{init}(T))) &\approx \mathsf{true}\\
\mathsf{vgetT}(g(h(X))) &\approx \mathsf{vgetT}(g(X))\\
\mathsf{extract}(g(\mathsf{init}(T)), \mathsf{init}(T)) &\approx \mathsf{init}(T)\\
\mathsf{extract}(g(h(X)), \mathsf{init}(T)) &\approx h(\mathsf{extract}(g(X), \mathsf{init}(T)))\\
\mathsf{extract}(g(h(X)), h(X')) &\approx h(\mathsf{extract}(g(X), X'))\\
\mathsf{vextract}(g(\mathsf{init}(T)), \mathsf{init}(T)) &\approx \mathsf{true}\\
\mathsf{vextract}(g(h(X)), \mathsf{init}(T)) &\approx \mathsf{vextract}(g(X), \mathsf{init}(T))\\
\mathsf{vextract}(g(h(X)), h(X')) &\approx \mathsf{vextract}(g(X), X')
\end{aligned}$$

Figure 5.2: Algebraic properties for the OSK example.

successful, updates its own state and sends an ok message:

Reader

$\mathsf{rcv}(x).$
$\mathsf{try}\ T = \mathsf{getT}(x)\ \mathsf{in}$
  $s := \mathsf{state}(T).$
  $\mathsf{try}\ s' = \mathsf{extract}(x, s)\ \mathsf{in}$
    $\mathsf{state}(T) := h(s').$
    $\mathsf{snd}(\mathsf{ok}).0$

We illustrate the semantics by showing how to reach a state of the OSK protocol that violates (α, β)-privacy. The initial state is $\mathcal{S}_0 = \{\alpha_0 \equiv \mathsf{true}, \beta_0 \equiv \mathsf{true}, \gamma_0 \equiv \mathsf{true}, \delta_0 \equiv \mathsf{true}\}$; we omit η_0 since this example is purely possibilistic. We start with a transition of process *Tag*, and we thus get to the following possibilities (with a variable-renamed copy of *Tag*): $\{(\star\ T_1 \in \mathsf{Tags}.\ \mathsf{Key}_1 := r(T_1).\ r(T_1) := h(\mathsf{Key}_1).\ \mathsf{snd}(g(\mathsf{Key}_1)).\ 0, \mathsf{true}, \{\})\}$. The first step is choosing a value from Tags for T_1, i.e., we have $|\mathsf{Tags}|$ successor states. Let us focus on the choice t_1, and thus γ_0 is augmented by $T_1 \doteq t_1$, and α and β are augmented by $T_1 \in \mathsf{Tags}$. Then we apply the rule for cell reads. Since δ_0 is still empty, we just replace Key_1 by $\mathsf{init}(T_1)$ in the rest of the process. We can now apply the rule for cell write. This means δ_0 is augmented by $r(T_1) := h(\mathsf{init}(T_1))$ if true. is sending a message and we thus augment β by $concr[l_1] = g(\mathsf{init}(t_1)) \wedge struct[l_1] = g(\mathsf{init}(T_1))$. There is just one possibility in our configuration and it has terminated, so the transaction is completed, getting to the state shown in the first line of Figure 5.3 (we refer to the α in that line as α_1 and so on).

For the second transition, we use *Tag* once more, the new possibilities being

	α	β	γ	δ
1	$T_1 \in \mathsf{Tags}$	$concr[l_1] = g(\mathsf{init}(t_1)) \wedge struct[l_1] = g(\mathsf{init}(T_1))$	$T_1 \doteq t_1$	$r(T_1) \coloneqq h(\mathsf{init}(T_1))$ if true
2	$T_2 \in \mathsf{Tags}$	$concr[l_2] = g(h(\mathsf{init}(t_1))) \wedge \exists i \in \{1,2\}.$ $i = 1 \wedge struct[l_2] = g(h(\mathsf{init}(T_1))) \wedge T_1 \doteq T_2$ $\vee\ i = 2 \wedge struct[l_2] = g(\mathsf{init}(T_2)) \wedge T_1 \neq T_2$	$T_2 \doteq t_1$	$r(T_2) \coloneqq h(h(\mathsf{init}(T_1)))$ if $T_1 \doteq T_2$ $r(T_2) \coloneqq h(\mathsf{init}(T_2))$ if $T_1 \neq T_2$
3		$concr[l_3] = \mathsf{ok} \wedge \exists i \in \{1,2\}.$ $i = 1 \wedge struct[l_3] = \mathsf{ok} \wedge T_1 \doteq T_2$ $\vee\ i = 2 \wedge struct[l_3] = \mathsf{ok} \wedge T_1 \neq T_2$		$\mathsf{state}(T_1) \coloneqq h(\mathsf{init}(T_1))$ if $T_1 \doteq T_2$ $\mathsf{state}(T_2) \coloneqq \mathsf{init}(T_2)$ if $T_1 \neq T_2$
4		$T_1 \doteq T_2$		$\mathsf{state}(T_1) \coloneqq h(\mathsf{init}(T_1))$ if $T_1 \neq T_2$

Figure 5.3: Execution of the OSK Protocol

$\{(\star\ T_2 \in \mathsf{Tags}.\ \mathsf{Key}_2 := r(T_2).\ r(T_2) := h(\mathsf{Key}_2).\ \mathsf{snd}(g(\mathsf{Key}_2)).\ 0, \mathsf{true}, \mathit{struct})\}$. Let us consider the transition where we pick for the choice of T_2 the same tag t_1. This time, the cell read introduces a case split:

$$\mathsf{if}\ T_2 \doteq T_1\ \mathsf{then\ let}\ \mathsf{Key}_2 = h(\mathsf{init}(T_1)) \ldots\ \mathsf{else}\ \ \mathsf{let}\ \mathsf{Key}_2 = \mathsf{init}(T_2) \ldots .$$

The normalization of if splits this into two possibilities: $\{\underline{(P_a, T_1 \doteq T_2, \mathit{struct}_1)}, (P_b, T_1 \not\doteq T_2, \mathit{struct}_1)\}$ where P_a and P_b are instantiations of the process $r(T_2) := \mathsf{Key}_2.\ \mathsf{snd}(g(\mathsf{Key}_2))$ by $\mathsf{Key}_2 = h(\mathsf{init}(T_1))$ and $\mathsf{Key}_2 = \mathsf{init}(T_2)$, respectively, and where struct_1 is the frame from the first transaction. The case where $T_2 \doteq T_1$ is marked since this is the reality. The normalization of the cell writes augments δ_1 by two lines (in either order): $r(T_2) := h(h(\mathsf{init}(T_1)))$ if $T_2 \doteq T_1$ and $r(T_2) := h(\mathsf{init}(T_2))$ if $T_2 \neq T_1$. It remains to send the outgoing, message and the structural information is now different, leading to the β in line 2 of Figure 5.3. The corresponding structural knowledge of each possibility is updated with the respective version, let us call them struct_a and struct_b in the following. Since they both have terminated, we have reached the end of the second transaction.

The new possibilities are $\{\underline{(\mathit{Reader}(3), T_1 \doteq T_2, \mathit{struct}_a)}, (\mathit{Reader}(3), T_1 \not\doteq T_2, \mathit{struct}_b)\}$ after a *Reader* transition, where $\mathit{Reader}(3)$ is a renaming of the reader process variables with index 3. We evaluate the receive step and here we have a choice of every recipe that the intruder can generate: we use l_2, i.e., the message from the second token transaction. Note that $\mathit{struct}_a[l_2] = g(h(\mathsf{init}(T_1)))$ and $\mathit{struct}_b[l_2] = g(\mathsf{init}(T_2))$, which is what we insert for the received message x_3 in the respective processes. When the processes (successfully) try $\mathsf{getT}(x_3)$, we get thus $\mathsf{let}\ T_3 = T_1$ and $\mathsf{let}\ T_3 = T_2$, respectively. The state lookup gives the respective initial value, since we have not yet written anything to the state cells. Thus also trying $\mathsf{extract}(T, s)$ will succeed and either produce $s_3 := h(\mathsf{init}(T_1))$ or $s_3 := \mathsf{init}(T_2)$. We thus amend δ by the two lines (in either order) $\mathsf{state}(T_1) := h(h(\mathsf{init}(T_1)))$ if $T_1 \doteq T_2$ and $\mathsf{state}(T_2) := h(\mathsf{init}(T_2))$ if $T_1 \not\doteq T_2$. In both processes, we are now at a sending step. Even if the message is the same in both processes, we still have to consider a case distinction since the conditions differ, as shown in line 3 of Figure 5.3 (this formula can be simplified, of course). Again, both processes are empty, so we have finished the third transaction.

Finally, we have $\{\underline{(\mathit{Reader}(4), T_2 \doteq T_1, \mathit{struct}'_a)}, (\mathit{Reader}(4), T_2 \neq T_1, \mathit{struct}'_b)\}$ after performing another *Reader* process, with $\mathit{Reader}(4)$ again being a renaming of variables with index 4 and struct'_a and struct'_b are the augmented *struct*s frames with the last ok-message. We consider the intruder choosing l_1 as a recipe for the received message, i.e., $\mathit{struct}'_a[l_1] = \mathit{struct}'_b[l_1] = g(h(\mathsf{init}(T_1)))$ for variable x_4. The next operation tries $\mathsf{getT}(x_4)$, which gives T_1 in any case. Looking up the $\mathsf{state}(T_1)$ gives $s_4 := h(h(\mathsf{init}(T_1)))$ in the first possibility (due to $T_1 \doteq T_2$), and the initial value $s_4 := \mathsf{init}(T_1)$ in the second. Thus, the next try succeeds only for the second possibility, and we have: $\{\underline{(0, T_2 \doteq T_1, \mathit{struct}'_a)}$,

$(\mathsf{snd}(\mathsf{ok}).0, T_2 \doteq\!\!\!\neq T_1, struct'_b)\}$. Now, the evaluation rule for the marked possibility being finished tells us: the second possibility cannot be the case because it would send a message, and the intruder can see that this does not happen, so we can augment β by the condition of the only remaining possibility, i.e., $T_1 \doteq T_2$. That is indeed a violation of privacy since we can now exclude all those models of α where $T_1 \neq T_2$.

5.2 Probabilistic privacy

In the previous section, we introduced random privacy variables, but we did not fully explain their purpose and functioning yet. For some problems that satisfy dynamic possibilistic (α, β)-privacy, we might want to refine the analysis. Extending all the models of α to a model of β does not mean that we do not leak information on the likelihood of a specific model of α through the technical information in β.

5.2.1 Probabilistic Alpha-Beta-Privacy

We thus propose a conservative extension first of static and then of dynamic (α, β)-privacy in order to add probabilities. We still consider a protocol specification, but, while the problems that we considered before had only nondeterministic privacy variables, we now consider problems that also have some privacy variables that can be sampled according to a probability distribution. These random variables are organized in a probability decision tree as defined via η in the previous section. We define the probability of a model of α, and intuitively, if the models of β yield a different probability for the model of α that they extend, then β violates the privacy of α in a probabilistic sense. However, the difference in probabilities might not be *significant*, so it is left to the modeler's appreciation to define an acceptable threshold.

Before giving formal definitions, we illustrate the interest of a probabilistic definition of (α, β)-privacy by means of a simple example, the well-known *Monty Hall problem* that originates from the game show "Let's Make a Deal" [Sel75].

Example 5.3 (Monty Hall problem)**.** *There are three doors, and only one of these doors hides a valuable prize; the two other doors hide joke prizes*[3]*. We model this situation with a random variable x. The prize has an equal probability to be found behind each door. We write $x \leftarrow \{1, 2, 3\}$ to say that x is sampled*

[3]The joke prizes were called *zonk* in the game

from these values with a uniform distribution. The trader (this is "Let's Make a Deal" lingo) has to choose a door and communicate (send) their choice to the host, Monty Hall. Monty then opens one of the two doors that were not chosen by the trader and that does not hide the prize. However, from the point of view of the trader, Monty chooses a door randomly between the two remaining doors. Monty then gives the trader the choice whether they want to switch door or not, before ultimately the two remaining doors are opened and the location of the prize is revealed. Let us formalize this with a dynamic probabilistic (α, β)-privacy and define the Monty process.

Monty

$\star\ x \leftarrow \{1,2,3\}.$
$\mathsf{rcv}(\mathsf{choice}).$
$\star\ \mathsf{open} \leftarrow \{1,2,3\} \setminus \{x, \mathsf{open}\}.$
$\star\ x \neq \mathsf{open}.$
$\mathsf{snd}(\mathsf{open}).0$

$\mathsf{open} \leftarrow \{1\colon \frac{1}{3}, 2\colon \frac{1}{3}, 3\colon \frac{1}{3}\} \setminus \{x, \mathsf{open}\}$ *is syntactic sugar for:*

$\mathsf{if}\ x \doteq 1\ \mathsf{then}$
  $\mathsf{if}\ \mathsf{choice} \doteq 2\ \mathsf{then}\ \mathsf{open} \leftarrow \{1\colon 0, 2\colon 0, 3\colon 1\}.$
  $\mathsf{else\ if}\ \mathsf{choice} \doteq 3\ \mathsf{then}\ \mathsf{open} \leftarrow \{1\colon 0, 2\colon 1, 3\colon 0\}.$
  $\mathsf{else}\ \mathsf{open} \leftarrow \{1\colon 0, 2\colon \frac{1}{5}, 3\colon \frac{1}{2}\}.$
$\mathsf{else\ if}\ x \doteq 2 \ldots$

Before the execution of the process, $\alpha_0 \equiv \mathsf{true}$. After, it is revealed as part of α that the prize was put behind a door chosen with a uniform probability. It is also revealed which door has been opened, thus which door does not hide the prize. Let us now consider one concrete reachable state after the execution of the Monty process, namely one where the intruder in the role of trader chose, or sent, 1 for the choice, the prize is behind the third door and Monty opened the second door: $\gamma \equiv x \doteq 3\ \wedge\ \mathsf{choice} \doteq 1\ \wedge\ \mathsf{open} \doteq 2$. We then have:

$$\alpha \equiv x \in \{1,2,3\}\ \wedge\ x \neq 2,\ \textit{i.e.,}\ \alpha \equiv x \in \{1,3\}$$

Intuitively, the trader thinks they have an equal chance to find the prize either behind the door that they initially choose or behind the other door. However, since they know the way Monty chooses the door to open, β and η are as follows, where l is a fresh label generated during the evaluation of the transition:

$$\beta \equiv \alpha \wedge \mathit{concr}[l] = 2 \wedge \mathit{struct}[l] = \mathsf{open}$$

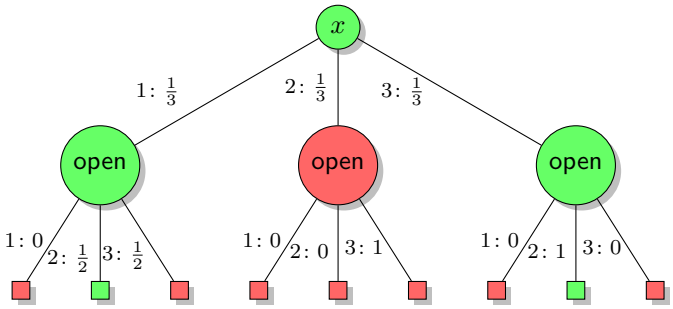

Figure 5.4: Probability tree for the Monty Hall problem

We colored in red the nodes for which no model exists in the considered state, e.g., the branch for which $x \doteq 2$ *is not supported by any valid model. The table below shows the models of* α *and their extension to* β *for this concrete reachable state where AP stands for the* absolute probability*, i.e., the multiplication of the probability of events for that model, and NP for the* normalized probability*, i.e., so that the column sums up to* 1*. Remember that this does not yet follow from our transition systems definition, but rather motivates the following definition of the probability of the models of* α *and* β*. We can see now the well-known glitch in the Monty Hall problem: according to* α*, the probability of* $x \doteq 1$ *and that of* $x \doteq 3$ *are both* $\frac{1}{2}$*, but according to* β*,* $x \doteq 1$ *has probability* $\frac{1}{3}$ *whereas* $x \doteq 3$ *has* $\frac{2}{3}$*, i.e.,* β *has been leaking information about* x *that was not contained in* α*. This reflects the advice that in this situation the trader should switch to door* 3*.*

α-Model	α-AP	α-NP	β-Model	β-AP	β-NP
$x = 1$	$\frac{1}{3}$	$\frac{1}{2}$	open $= 1$ open $= 2$ open $= 3$	impossible $\frac{1}{6}$ impossible	$\frac{1}{3}$
$x = 3$	$\frac{1}{3}$	$\frac{1}{2}$	open $= 1$ open $= 2$ open $= 3$	impossible $\frac{1}{3}$ impossible	$\frac{2}{3}$

Table 5.1: Models of α and their extensions to β for the Monty Problem

Since we propose below a conservative extension, we want to be able to talk about models that differ only by the choice of non-deterministic variables. To this aim, we define an equivalence relation for the interpretations over the free random variables of α (respectively, of β): given two models of α (of β) $\mathcal{I}_1$ and

$\mathcal{I}_2$, $\mathcal{I}_1 =_r \mathcal{I}_2$ iff $\mathcal{I}_1(x) = \mathcal{I}_2(x)$ for all $x \in fv_r(\alpha)$ (for all $x \in fv_r(\beta)$) where $fv_r(\cdot)$ refers to free random variables of a formula. We define an equivalence class $[\mathcal{I}]$ ($[\mathcal{I}']$) induced by this relation for all Σ_0-interpretations $\mathcal{I}$ of α (for all Σ-interpretation $\mathcal{I}'$ of β).

Since there are finitely many free random variables of α (of β), there are finitely many equivalence classes of probabilistic models of α, i.e., there exist $\mathcal{I}_1, \ldots, \mathcal{I}_k$, s.t. $k \in \mathbb{N}^+$, such that $[\mathcal{I}_1] \cup \ldots \cup [\mathcal{I}_k]$ is a partition of the models of α (respectively, there exists $\mathcal{I}_1, \ldots, \mathcal{I}_{k'}$, s.t. $k' \in \mathbb{N}^+$, such that $[\mathcal{I}_1] \cup \ldots \cup [\mathcal{I}_{k'}]$ is a partition to their extensions to models of β).

Definition 5.7 (Absolute and Normalized Probabilities)**.** *Given two formulae α and β, and a probability decision tree η, an* interpretation class $[\mathcal{I}]$ *corresponds to a unique path in η starting at the root node, and we define its* absolute probability $\mathcal{P}_{abs,\eta}([\mathcal{I}])$ *as the product of the probabilities along the path. Note that if $[\mathcal{I}]$ is an α interpretation class, the path traverses just the upper part of η that corresponds to the free variables of α while if $[\mathcal{I}]$ is a β interpretation the path reaches a leaf.*

Let $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ be the model classes of α (of β); we define the normalized probability *of each interpretation class as:*

$$\mathcal{P}_\eta([\mathcal{I}_i]) = \frac{\mathcal{P}_{abs,\eta}([\mathcal{I}_i])}{\sum_{j=1}^{k} \mathcal{P}_{abs,\eta}([\mathcal{I}_j])} .$$

Note that $\mathcal{P}_\eta([\mathcal{I}])$ is defined so it does not depend on the choice of the representative $\mathcal{I}$ of the equivalence class $[\mathcal{I}]$.

We allow β to have more free variables than α. In particular, it allows β to have more free random variables than α. The intuitive idea to formulate probabilistic (α, β)-privacy as a conservative extension is to require that the sum of the probabilities of the equivalence classes of β, when restricted to the free variables of α and the payload alphabet Σ_0, is equal to the probabilities of the equivalence classes of α:

Definition 5.8 (Probabilistic (α, β)-privacy)**.** *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$, and both α and β are consistent, and η is the probability decision tree. We say that* (α, β)-privacy holds probabilistically *iff (α, β)-privacy holds and*

$$\mathcal{P}_\eta([\mathcal{I}_0]) = \sum_{i=1}^{k} \mathcal{P}_\eta([\mathcal{I}_i])$$

for every model $\mathcal{I}_0$ of α, and for the models $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ of β (partitioned by equivalence class) s.t. $\mathcal{I}_i|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$ for every $i \in \{1, \ldots, k\}$. We say that

(α, β)-privacy holds probabilistically and dynamically *iff* (α, β)*-privacy holds probabilistically in every reachable state.*

Intuitively, this means that the probability of every model of β that agrees on the payload part, when considered together, equals the probability of the original model of α. With this definition, we can see that the Monty Hall problem from Example 5.3 satisfies possibilistic (α, β)-privacy in the state we considered but breaks probabilistic (α, β)-privacy. Now, let us see how we could correct the protocol:

Example 5.4 (Alternative Monty Hall)**.** *Suppose the door that Monty opens after the choice of the trader is taken randomly between the doors that were not chosen by the trader, thus including the one hiding the prize (even though this might shorten the game). We propose the process:*

Alternative Monty

$\star\ x \leftarrow \{1, 2, 3\}.$
$\mathsf{rcv}(\mathsf{choice}).$
$\star\ \mathsf{open} \leftarrow \{1, 2, 3\} \setminus \{\mathsf{choice}\}.$
$\mathsf{if}\ \mathsf{open} \doteq x\ \mathsf{then}$
  $\star\ x \doteq \mathsf{open}.$
else
  $\star\ x \not\doteq \mathsf{open}.$
$\mathsf{snd}(\mathsf{open}).0$

We consider the same reachable state as in Example 5.3. α*,* β *and* γ *are similar. The probability tree has a similar form but we update the probability on the branches accordingly. Let us look again at the models of* α *and their extension to* β*. This time again, the probability of* $x \doteq 1$ *and* $x \doteq 3$ *is both* $\frac{1}{2}$ *according to* α *and so it is according to* β*.* β *is not leaking information about* x *that was not contained in* α *anymore. This reflects that the trader cannot adopt a better strategy.*

The difference between the original Monty Hall problem in Example 5.3 and the alternative Monty Hall problem in Example 5.4 is that the choice of the door that Monty opens is independent of where the prize is located in the second case. In other words, the free random variable of β, namely the choice of the opened door, could have been 2 or 3, whatever the free random variable of α, namely the location of the prize; that is, the choice of the location of the prize does not influence the probability distribution of the choice of the opening of the door. We can actually identify a condition under which if (α, β)-privacy holds possibilistically, then (α, β)-privacy also holds probabilistically:

α-Model	α-AP	α-NP	β-Model	β-AP	β-NP
$x = 1$	$\frac{1}{3}$	$\frac{1}{2}$	$\mathsf{open} = 1$ $\mathsf{open} = 2$ $\mathsf{open} = 3$	impossible $\frac{1}{6}$ impossible	$\frac{1}{2}$
$x = 3$	$\frac{1}{3}$	$\frac{1}{2}$	$\mathsf{open} = 1$ $\mathsf{open} = 2$ $\mathsf{open} = 3$	impossible $\frac{1}{6}$ impossible	$\frac{1}{2}$

Table 5.2: Models of α and their extension to β for the Alternative Monty Problem

Definition 5.9 (Extensibility)**.** *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$ and both α and β are consistent. We say that a pair (α, β) is* extensible *if it is possible to extend every model of α by a number of models of β that cover the whole domain of the free random variables occurring exclusively in β, i.e., for all $\mathcal{I}_0 \models \alpha$. for all $\sigma\colon y_1, \ldots, y_k \mapsto dom(y_1) \times \cdots \times dom(y_k)$. there exists $\mathcal{I} \models \beta \wedge \sigma$ such that $\mathcal{I}|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$.*

The definition of extensibility does not refer to probabilities or to a probability tree η. When we have extensibility, it means that the $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ are exactly all the leaves under $\mathcal{I}_0$, so the probability of that subtree is 1, as can be seen by induction. As a consequence, the absolute probability of $[\mathcal{I}_0]$ is the same as the sum of the absolute probabilities of the $[\mathcal{I}_i]$.

We can prove that if (α, β)-privacy holds possibilistically for (α, β) that is extensible, then (α, β)-privacy also holds probabilistically. This is only a sufficient condition: probabilistic (α, β)-privacy may hold, even if (α, β) is not extensible.

Theorem 5.10. *Let $\Sigma_0 \subsetneq \Sigma$ and consider a formula α over Σ_0 and a formula β over Σ, s.t. $\beta \models \alpha$, $fv(\alpha) \subseteq fv(\beta)$ and both α and β are consistent. Let η be the probability decision tree. If (α, β)-privacy holds possibilistically, i.e., for every $\mathcal{I}_0 \models \alpha$, there exists $\mathcal{I} \models \beta$ such that $\mathcal{I}|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$, and (α, β) is extensible, then (α, β)-privacy holds probabilistically. We extend this theorem to dynamic (α, β)-privacy as expected.*

Proof. Let Σ_0, Σ, α, β, and η be as in the assumptions of the theorem and let (α, β)-privacy hold possibilistically and be extensible. Let $\mathcal{I}_0 \models \alpha$ and $[\mathcal{I}_1], \ldots, [\mathcal{I}_k]$ be the models of β (partitioned by equivalence class) such that $\mathcal{I}_i|_{\Sigma_0, fv(\alpha)} = \mathcal{I}_0$ for every $i \in \{1, \ldots, k\}$. Let p_α be the sum of the absolute

probabilities of all α models, so,

$$\mathcal{P}_\eta([\mathcal{I}_0]) = \frac{1}{p_\alpha}\mathcal{P}_{abs,\eta}([\mathcal{I}_0])\,.$$

Since (α,β)-privacy is extensible, the probability of the β subtree for each α model is 1 (as every leaf of the subtree is a model and the children of each node in η sum up to 1). Thus,

$$\mathcal{P}_{\eta,abs}([\mathcal{I}_0]) = \sum_{i=1}^{k}\mathcal{P}_{\eta,abs}([\mathcal{I}_i])$$

The sum of the absolute probabilities of all β model classes is thus p_α (since this is the sum of the α-model classes). Thus,

$$\begin{aligned}\sum_{i=1}^{k}\mathcal{P}_\eta([\mathcal{I}_i]) &= \sum_{i=1}^{k}\frac{1}{p_\alpha}\mathcal{P}_{\eta,abs}([\mathcal{I}_i])\\ &= \frac{1}{p_\alpha}\sum_{i=1}^{k}\mathcal{P}_{\eta,abs}([\mathcal{I}_i])\\ &= \frac{1}{p_\alpha}\mathcal{P}_{\eta,abs}([\mathcal{I}_0])\\ &= \mathcal{P}_\eta([\mathcal{I}_0])\end{aligned}$$

, and therefore, (α,β)-privacy holds probabilistically. □

5.2.2 The intruder as an empirical scientist

Probabilistic (α,β)-privacy is a "sharp sword": a relatively small shift in probabilities due to the information in β is already a violation of (α,β)-privacy, while it may be too insignificant to be beneficial for the intruder. One wants to avoid even minimal shifts, because in many areas of security we have seen how an intruder can actually "amplify" tiny imperfections in a system, so that they become significant. Nonetheless, it can be insightful to analyze also a system that does not satisfy probabilistic (α,β)-privacy for a weaker goal. In particular, we want to regard now the intruder as an empirical scientist who conducts an experiment, where they manipulate some "independent variables", and try to observe a significant effect.

The Helios voting protocol originally allowed an intruder, as a dishonest voter, to copy votes of others [CS11]. For a small number of voters, this can lead to a violation of possibilistic (α,β)-privacy, e.g., a and b are honest, c dishonest, and

we consider a binary vote between 0 and 1; then the intruder can copy a's vote; the result is greater than two iff a voted 1. Thus, possibilistic (α, β)-privacy does not hold. It would, however, if the cardinality would not allow for such a deduction, but if we look at the problem probabilistically, i.e., we know a *distribution* of the votes, then this yields a shift in probabilities.

We could see this as an experiment, where the intruder is not really interested in breaking the privacy of the entire vote, but rather targeting the privacy of a particular voter a—even sacrificing their chance to make their preferred option win by copying a's vote. If this is the actual goal, then actually it makes sense to focus on this single vote also in the privacy goal, i.e., to say $\alpha \equiv x_1 \in \{0, 1\}$, and β containing all the other votes and the result. This means that all the other voters and the result are for the intruder right now not interesting but just how a has voted. The copying of α's vote is thus the experiment the intruder takes if we regard a's vote as an *independent variable* of a scientific experiment.

Let us make such an experiment for 6 honest and 2 dishonest voters, meaning the intruder can copy the vote of a two times. The result is 5 votes for candidate 1. Let $v_1, \ldots, v_6$ be random variables and v_7, v_8 are the votes controlled by the intruder.

$$\begin{array}{lcl} \alpha & \equiv & v_1 \in \{0, 1\} \\ \beta & \equiv & \alpha \ \wedge \ v_2, \cdots, v_6 \in \{0, 1\} \ \wedge \ v_7, v_8 \in \{0, 1\} \\ & & \wedge \ \sum_{i=1}^{8} v_i = 5 \ \wedge \ v_7 \doteq v_1 \ \wedge \ v_8 \doteq v_1 \, . \end{array}$$

By analyzing the previous elections, the intruder knows the random variables are chosen following the probability distribution $\{0 \colon \frac{2}{3}, 1 \colon \frac{1}{3}\}$. It is clear that α has only two models: $\mathcal{I}_1$ s.t. $\mathcal{I}_1(v_1) = 0$ and $\mathcal{P}_\eta([\mathcal{I}_1]) = \frac{2}{3}$, $\mathcal{I}_2$ s.t. $\mathcal{I}_2(v_1) = 1$ and $\mathcal{P}_\eta([\mathcal{I}_2]) = \frac{1}{3}$. Both these models can be extended to models of β. Possibilistically, (α, β)-privacy holds. However, if we normalize the sum of the probabilities of the interpretation class for the β models that extend these two models, we obtain for the first one $\mathcal{P}_1 \approx 2.44$ and for the second $\mathcal{P}_2 \approx 97.56$. In this case, there is nearly no doubt for the intruder that voter a voted for the candidate 0.

In general, the intruder may set a threshold for being *convinced* for a particular model, e.g., when all other models together have a probability of at most 0.05. One can thus also calculate, given the number of honest and dishonest voters, as well the probability distribution, whether the intruder has even a chance to find a significant result. For instance, in an election with 100 votes where the intruder controls just five votes, a's privacy is well protected, unless there is a candidate with a very low popularity, say 0.01, and a votes for that candidate. We explain in Section 5.4 how to model voting protocols with dynamic (α, β)-privacy.

5.2.3 Background Knowledge

The authors of [MV19] explained what happens when the intruder can use background knowledge outside our formal method. Consider a small village where everybody votes the same way. A new person settles in, and in the next election, a vote for the opposition party is cast. Even a perfect privacy-preserving voting system cannot prevent the intruder to infer that it is quasi-certain this vote comes from the newcomer. The authors proved that the possibilistic (α, β)-privacy is *stable* under an arbitrary consistent intruder background knowledge. One may wonder if the stability under background knowledge that holds for possibilistic (α, β)-privacy also holds for probabilistic (α, β)-privacy. Unfortunately, this is not the case in general:

$$
\begin{array}{l}
\star\ x \leftarrow \{0\colon \tfrac{1}{2}, 1\colon \tfrac{1}{2}\}. \\
\star\ y \in \{0, 1\}. \\
\mathsf{if}\ x \doteq 1\ \mathsf{then} \\
\quad \star\ z \leftarrow \{0\colon \tfrac{1}{2}, 1\colon \tfrac{1}{2}\}. \\
\quad \mathsf{snd}(y \oplus z) \\
\mathsf{else} \\
\quad \star\ z \leftarrow \{0\colon 0, 1\colon 1\}. \\
\quad \mathsf{snd}(z)
\end{array}
$$

While this is an artificial example, it has some similar patterns to Σ-protocols, i.e., giving out a "secret" y "blinded" by z in one case, and just z in another case. Suppose this process executes and the intruder observes that it sends the value 1. Then, we have the models depicted in the following table where (we have arranged the items, so that we can summarize them to model classes) we have shaded red all those models that get excluded by the fact that we have observed sending 1 (because this means that either $x = 1$, and then y and z must be different, or $x = 0$, then z must be 1).

So far, (α, β)-privacy holds probabilistically: all model classes of β where $x \doteq 1$ have together probability $\frac{1}{2}$ as does the corresponding α-class, and the same for $x \doteq 0$. If we now have the background knowledge that $y \doteq 1$, then this excludes some further models that are shaded in blue in the from x. While this still preserves *possibilistic* (α, β) privacy, it violates *probabilistic* (α, β) privacy: according to β, $x \doteq 1$ is half as likely as $x \doteq 0$.

Indeed, this is a very constructed example (where for $x \doteq 0$, z is not really random any more), and actually in many practically relevant examples, background knowledge indeed also preserves probabilistic (α, β)-privacy for extensible pair

α-Prob	x	y	z	β-Prob
$\frac{1}{2}$	0	0	0	0
	0	1	0	
	0	0	1	$\frac{1}{2}$
	0	1	1	
$\frac{1}{2}$	1	0	0	$\frac{1}{4}$
	1	1	0	
	1	0	1	$\frac{1}{4}$
	1	1	1	

Table 5.3: Example: for non-extensible (α, β), probabilistic privacy is in general not stable under background knowledge.

(α, β):

Theorem 5.11 (Stability Under Background Knowledge)**.** *Let α and β be given so that (α, β)-privacy holds possibilistically, and (α, β) is extensible. Let α_0 be a Σ_0-formula. Then $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy holds probabilistically. We extend this result for dynamic (α, β)-privacy as expected.*

Proof. If (α, β) is extensible, then also $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$ is extensible. Since (α, β)-privacy holds possibilistically, by stability under background knowledge, $(\alpha \wedge \alpha_0, \beta \wedge \alpha_0)$-privacy holds possibilistically, and since it is still extensible, it also holds probabilistically by Theorem 5.10. □

5.3 DP-3T

As a concrete, and topical example, let us consider the decentralized, privacy-preserving proximity tracing system DP-3T [DP-20], which has been developed to help slow the spread of the SARS-CoV-2 virus by identifying people who have been in contact with an infected person. The DP-3T system aims to minimize privacy and security risks for individuals and communities, and to guarantee the highest level of data protection.

5.3.1 Modeling

For every agent and for every day, we have a day key, and the day is further separated into periods (e.g., of 15 minutes), and for each period, each agent

generates a new ephemeral identity. In order to avoid any complications with infinite numbers of models, we consider finite (but arbitrarily large) sets of agents, day keys, and ephemeral IDs. Moreover, we use these sets as *sorts*, so that we can define interpreted functions between these sorts without inducing infinitely many models for these functions. We use the following sorts:

- Agent is the sort of all participating agents,
- $Day = \{0, \ldots, \mathsf{D} - 1\}$ identifies days,
- $Period = \{0, \ldots, \mathsf{P} - 1\}$ identifies a particular period of a day, i.e., a day is partitioned into P periods (e.g., of 15 minutes).,
- SK is the sort of daily identities, and
- $EphID$ is the sort of ephemeral identities (changing, e.g., every 15 minutes),

Let all elements of these sorts but SK be part of Σ_0, so that α formulae can talk about agents, days, and ephemeral identities. On these sorts, we define the following functions and relations:

- $\mathsf{sk}_0[\cdot] \colon \mathsf{Agent} \to SK$ maps every agent to their first-day key. We assume that this key is distinct for every agent, i.e., $\mathsf{sk}_0[a] \neq \mathsf{sk}_0[b]$ for any $a \neq b$,
- $h[\cdot] \colon SK \to SK$ is a hash function that maps every daily identity to the next day. We assume that for every $a \colon \mathsf{Agent}$, we have a seed value $\mathsf{sk}_0[a] \in SK$ such that $h^i[\mathsf{sk}_0[a]] \neq h^j[\mathsf{sk}_0[b]]$ for any $a, b \in \mathsf{Agent}$, $i, j \in Day$ with $(a, i) \neq (b, j)$: every daily identity of an agent is unique[4],
- $prg[\cdot, \cdot] \colon SK \times Period \to EphID$ models a pseudo-random number generator to generate the ephemeral identities. We assume prg is injective on the domain $SK \times Period$, so that there is also no collision between the ephemeral identities of any agents (with respect to any timepoints).
- $pwnr[\cdot] \colon EphID \to \mathsf{Agent}$ relates, in our model, an ephemeral ID to its actual owner, i.e., for $e = prg[h^i[\mathsf{sk}(a)], j]$, we have $pwnr[e] = a$,
- $dayof[\cdot] \colon EphID \to Day$ tells the day an ephemeral ID is issued, and
- $sick \subseteq EphID \times Day$ is a relation where $sick(e, d)$ means that the agent identified by e has declared sick on day d. In contrast, $dayof[e]$ is the day when e was used.

[4] SK is a finite set, so h must have collisions, and we merely exclude that these collisions are relevant to the protocol. We abstract from cryptography and thus from the negligible probability of collisions between agents.

The functions h and *prg* are cryptographic functions, and sk_0 is a cryptographic setup. We regard them as technical/implementation related, so they are only part of $\Sigma \setminus \Sigma_0$ and cannot be used in α. We have made several assumptions about absence of collisions in these functions: these assumptions are part of β in the initial state. The function *pwnr* and the relation *sick* are part of the high-level modeling, and thus part of Σ_0.

We use the following memory cells with their initial values:

- $\mathsf{sk}_l(A\colon \mathsf{Agent}) \coloneqq \mathsf{sk}_0[A]$ is whatever is the opposite of a look-ahead: it represents the day ID of agent A of l days ago, where l is the period how far back we want to report the sickness after a positive test (e.g. five days),
- $\mathsf{sk}(A\colon \mathsf{Agent}) \coloneqq h^l[\mathsf{sk}_0[A]]$. The current day ID of A is l hashes ahead of sk_0. Thus, within the first l days of the app, we have some "virtual" past days where we can report sickness—this is to keep the model simple,
- $\mathit{today}() \coloneqq l$ is the current day counter (it is the same for all agents),
- $\mathit{period}() \coloneqq 0$, where 0 identifies the first period of a day,
- $\mathit{ephid}(A\colon \mathsf{Agent}) \coloneqq \mathit{prg}[\mathsf{sk}(A), \mathit{period}()]$ is the current ephemeral ID, and
- $\mathit{isSick}(A\colon \mathsf{Agent}) \coloneqq \mathsf{false}$ is a flag to indicate that the agent has reported sick and should no longer use the app and should quarantine.

We consider the agent transactions in Figure 5.5. The transaction *New Day or Period* advances a global clock, and when a day is finished, automatically triggers the generation of new day keys for each agent. This ignores any privacy problems that could arise from de-synchronized clocks and the like. The *Agent Advertise* transaction models that an agent can at any time communicate its current ephemeral identity e and that the intruder never learns more than the owner of e is some agent $x \in \mathsf{Agent}$. Here, our model ignores the details of how two agents' phones actually exchange IDs, which can cause also several problems [Vau20]. Finally, the *Agent Sick* Transaction models that an agent declares sick and publishes the day keys in their sickness period (for simplicity, we publish only the oldest, the others can be generated by everybody themselves). We specify that the intruder should now only learn that all ephemeral IDs belong to an agent that has just declared sick. The model actually omits the details of how this sick report is communicated to a central server (who must also somehow check with health authorities whether the agent is indeed sick), which again is not trivial to get right [Vau20]. Our model thus focuses on the core privacy question that arises, even if all exchange protocols work perfectly.

New Day or Period

if $(period() < \mathsf{P} - 1)$ then
  $period() := period() + 1$
else
  $period() := 0$
  if $(today() < \mathsf{D} - 1)$ then
    $today() := today() + 1$
    for x: Agent
      $\mathsf{sk}(x) := h(\mathsf{sk}(x))$
      $\mathsf{sk}_l(x) := h(\mathsf{sk}_l(x))$

Agent Advertise

$\star\ x \in$ Agent
if $\neg isSick(x)$ then
  let $z = prg[\mathsf{sk}(x), period()]$
  $\star\ pwnr[\mathcal{I}(z)] = x \wedge dayof[\mathcal{I}(z)] = today()$
  $\mathsf{snd}(z)$

Agent Sick

$\star\ x \in$ Agent
if $\neg isSick(x)$ then
  $isSick(x) := \mathsf{true}$
  let $y = \mathsf{sk}_l(x)$
  for $i \in Period\ \wedge\ j \in \{0, \ldots, l\}$
    $\star\ sick(\mathcal{I}(prg[h^j[y], i]), \mathcal{I}(today()))$
  $\mathsf{snd}(y)$

Figure 5.5: A model of DP-3T (with insufficient α).

5.3.2 Privacy violated

Suppose that we have two advertisements by the same agent a in the first two periods of the first day (numbered l), i.e., let $\mathsf{sk}_l = h^l[\mathsf{sk}_0[a]]$ be the day key, and $e_0 = prg[\mathsf{sk}_l, 0]$ and $e_1 = prg[\mathsf{sk}_l, 1]$ be the released ephemeral IDs. On the same day, a releases a sick note $\mathsf{sk}_0[a]$ that gives rise to further ephemeral IDs $e_2, \ldots, e_n$. Then, α in the reached state is:

$$\begin{aligned}\alpha \equiv\ & x_1 \in \mathsf{Agent} \wedge pwnr[e_0] = x_1 \wedge dayof[e_0] = l \\ & \wedge x_2 \in \mathsf{Agent} \wedge pwnr[e_1] = x_2 \wedge dayof[e_1] = l \\ & \wedge x_3 \in \mathsf{Agent} \wedge sick(e_0, l) \wedge \ldots \wedge sick(e_n, l)\end{aligned}$$

where $e_0, \ldots, e_n$ are all ephemeral keys of a released in the sick report. The following can be derived from β, for some labels m_1, m_2 and m_3 where the sent

messages are stored:

$$\begin{array}{ll} concr[m_1] = e_0 & struct[m_1] = prg[h^l[\mathsf{sk}_0[x_1]], 0] \\ concr[m_2] = e_1 & struct[m_2] = prg[h^l[\mathsf{sk}_0[x_2]], 1] \\ concr[m_3] = \mathsf{sk}_l & struct[m_3] = h^l[\mathsf{sk}_0[x_3]] \end{array}$$

Intruder deductions:

$$concr[prg[m_3, 0]] = prg[h^l[\mathsf{sk}[a]], 0] = e_0 = concr[m_1]$$
$$concr[prg[m_3, 1]] = prg[h^l[\mathsf{sk}[a]], 1] = e_1 = concr[m_2]$$

Using $\phi_\sim$:

$$struct[prg[m_3, 0]] = struct[m_1]$$
$$struct[prg[m_3, 1]] = struct[m_2]$$
$$prg[h^1[\mathsf{sk}_1[x_3]], 0] = prg[h^l[\mathsf{sk}_0[x_1]], 0]$$
$$prg[h^l[\mathsf{sk}_0[x_3]], 1] = prg[h^l[\mathsf{sk}_0[x_2]], 1]$$

By the properties of prg, h and $\mathsf{sk}_0 : x_3 = x_2 \wedge x_3 = x_1$, and thus $x_1 = x_2$

This last statement is however not compatible with all models of α, so dynamic possibilistic (α, β)-privacy is indeed violated. Note that we do not find out that $x_1 = a$, but we have linkability of pseudonyms of sick persons.

5.3.3 The Actual Privacy Guarantee

Actually, the protocol releases more information than we have specified so far in α. This corresponds to the privacy problem that the intruder gets to know that all the ephemeral identities of a day are related to the same agent. This could be practically relevant if, e.g., the intruder surveys in several places for ephemeral identities and can then build partial profiles of users who declared sick.

We at least need to add the following information: in the sick release by the information there is one particular agent who is the owner of all released sick-predicates, i.e., in the Agent Sick transaction we have the α release:

$$\star\ sick(\mathcal{I}(prg[h^j[y], i]), \mathcal{I}(today())) \wedge pwnr[\mathcal{I}(prg[h^j[y], i])] = x$$

This provides the link between all ephemeral IDs released by an agent, because the owner is the same agent x (who of course still remains anonymous, hence the variable).

As a consequence, "admitting" in α this additional information, what we could find out in the concrete scenario before, namely that $x_1 = x_2 = x_3$, no longer

$$
\begin{aligned}
&\forall E, F \in \mathit{EphID}, C, D \in \mathit{Day}\colon\ \mathit{sick}(E, C) \wedge \mathit{sick}(F, D) \wedge \\
&\qquad \mathit{pwnr}[E] \doteq \mathit{pwnr}[F] \implies C = D \\
&\wedge\ \forall E, F \in \mathit{EphID}, D \in \mathit{Day}\colon\ \mathit{sick}(E, D) \wedge \\
&\qquad \mathit{pwnr}[E] \doteq \mathit{pwnr}[F] \implies \mathit{dayof}[F] \leq D \\
&\wedge\ \forall E, F \in \mathit{EphID}, D \in \mathit{Day}\colon\ \mathit{pwnr}[E] \doteq \mathit{pwnr}[F] \wedge \\
&\qquad \mathit{dayof}[E] \doteq \mathit{dayof}[F] \wedge \mathit{sick}(E, D) \implies \mathit{sick}(F, D) \\
&\wedge\ \forall E_0, E_1, \ldots, E_{\mathsf{P}} \in \mathit{EphID}\colon\ \textstyle\bigwedge_{i,j \in \{0,\ldots,P\}, i \neq j} E_i \neq E_j \\
&\qquad \mathit{pwnr}[E_1] \doteq \ldots \doteq \mathit{pwnr}[E_{\mathsf{P}}] \wedge \\
&\qquad \mathit{dayof}[E_1] \doteq \ldots \doteq \mathit{dayof}[E_{\mathsf{P}}] \\
&\qquad \implies \mathit{pwnr}[E_0] \neq \mathit{pwnr}[E_1] \vee \mathit{dayof}[E_0] \neq \mathit{dayof}[E_1]
\end{aligned}
$$

Figure 5.6: Axioms for DP3T

count as an attacks, as we explicitly declare that we want to release this information.

However, this extended α *still* does not cover all the information we release. For instance, if the two agents $x_1 = a$ and $x_2 = b$ have released ephemeral IDs e_a and e_b, respectively, and a has declared sick, then we can still observe that $x_1 \neq x_2$ because e_b does not belong to any of the keys that have been released with a sick note. Similarly, we can distinguish agents that have declared sick; for instance, if both a and b have declared sick, then we can also derive $x_1 \neq x_2$, because we have distinct day keys and moreover, when two day keys belong to the same agent, then they are related by the hash function, i.e., $\mathsf{sk}_1 = h^k[\mathsf{sk}_2]$ or vice-versa.

So, actually, what we really give out here is much more, and it is not easy to keep track of it without basically copying into α most of what is going on in β, and thus basically making the implementation be also the specification. However, as most logicians will agree, there is almost always a declarative way to describe things. In this case, we can actually formalize a few relevant properties of the implementation as axioms on the Σ_0 level, without talking about the day keys SK or how they are generated and how the ephemeral IDs are generated. These axioms are given in Figure 5.6, and we obtained them from failed attempts of proving dynamic possibilistic (α, β)-privacy, adding missing aspects until we could prove it. Here is what these axioms respectively express:

- an agent declares sick only once,
- after declaring sick, the agent does not use the app anymore. In fact, they could, if we had a reset operation that installs a new initial key, but we refrained from further complicating the model,

- when an agent reports sick for a particular day, this entails all ephemeral identities for that day, and
- finally, let $\mathsf{P} = |Period|$ denote the number of periods in a day; then there cannot be more P ephemeral IDs that belong to the same agent on the same day.

We shall thus, from now on, consider the axioms in Figure 5.6 as part of α in our initial state.

Now, it may not be entirely intuitive anymore what this actually implies. So, let us look at the general form that α has after a number of transitions, and how to compute the models (satisfying interpretations) of α.

In general, in any reachable state the formula α consists of conjuncts of the following forms:

- from Agent Advertise: $x \in \mathsf{Agent} \wedge pwnr[e] = x \wedge dayof[e] = d$, where $e \in EphID$, $d \in Day$, and x is a variable that occurs nowhere else in α, and
- from Agent Sick: $\bigwedge_{e \in E} pwnr[e] = x \wedge sick(e, d_r)$, where E is a set of ephemeral IDs that are released on reporting day d_r. Amongst all agent sick reports, the set E is pairwise disjoint. Moreover, the variable x occurs nowhere else in α. Finally, the size of E is $|Period| \times l$, i.e., for every of the l days and for every time period of a day, we identify exactly one ephemeral ID as sick.

Lemma 5.12. *Every model $\mathcal{I}$ of α can be computed by the following* non-determinstic *algorithm:*

1. *Consider every conjunct that arose from Agent Sick and consider the variable x of that conjunct.*
   (a) *For every such x,* choose *a unique $a \in \mathsf{Agent}$ and set $\mathcal{I}(x) = a$. (Unique here means: two different Agent-sick conjuncts with variables x and x' must be interpreted as different agents $\mathcal{I}(x) \neq \mathcal{I}(x')$).*
   (b) *For every e that occurs in this conjunct, we have $\mathcal{I}(pwnr[e]) = a$.*
2. *Consider every conjunct that arose from Agent Advertise and let x be the variable occurring in there and e be the ephemeral ID in there.*
   (a) *If $\mathcal{I}(pwnr[e]) = a$ has been determined already, then $\mathcal{I}(x) = a$.*

(b) If $\mathcal{I}(pwnr[e])$ has not yet been determined, then let d be the day that is has been declared. Let Agent_s be the set of agents that have declared sick on day d or before, i.e., $\mathcal{I}(x')$ for every x' such that α contains $sick(e, d') \wedge pwnr[e] = x' \wedge dayof[e] = d_0$ and $d_0 \leq d$. Further, let Agent_e denote all the agents a for which $\mathcal{I}(pwnr[e]) = a$, and $\mathcal{I}(dayof[e]) = d$ for P different ephemeral IDs e. Then, choose $a \in \mathsf{Agent} \setminus \mathsf{Agent}_s \setminus \mathsf{Agent}_e$ *arbitrarily and set $\mathcal{I}(x) = a$ and $\mathcal{I}(pwnr[e]) = a$.*

3. All remaining aspects of $\mathcal{I}$ are actually irrelevant (i.e., $\mathcal{I}(pwnr[e])$ for e that did not occur in the formula).

In a nutshell: α does not reveal any agent names, but allows one to distinguish all sick agents from each other and from the non-sick, and it allows one to link all ephemeral IDs of every sick agent from the first day of sickness on.

Proof. Soundness (i.e., the algorithm produces only models of α): the algorithm respects obviously every conjunct of α produced during transactions, and for the axioms the distinct choice of sick-reported agents is actually sufficient.

Completeness (i.e., every model of α is produced by the algorithm): we have first to show that α enforces $\mathcal{I}(x_i) \neq \mathcal{I}(x_j)$ for every pair of variables x_i and x_j that occur in distinct sickness reports. Suppose this were not true, i.e., we have a model $\mathcal{I}$ of α such that $\mathcal{I}(x_i) = \mathcal{I}(x_j)$ for the variables x_i and x_j from distinct agent sickness reports. From the construction, we know each sick report contains exactly $\mathsf{P} \cdot l$ ephemeral IDs (l days reporting, and P periods per day), and the ephemeral IDs from distinct sick reports are disjoint. Moreover, each sick report has a reporting day, say d_i and d_j. Let thus e_i and e_j be ephemeral IDs from the two sick reports, then $\mathcal{I} \models pwnr[e_i] \doteq x_i \doteq x_j \doteq pwnr[e_j]$ and therefore the axioms entail $d_i = d_j$ (same day of reporting). Thus, α contains for each sick report P ephemeral IDs for l days up to reporting day $d_i = d_j$. That is however impossible by the axiom that not more than P different ephemeral IDs can have the same day and the same owner (while we have $2 \cdot \mathsf{P}$ according to assumption). Thus, $\mathcal{I}(x_i) \doteq \mathcal{I}(x_j)$ is absurd.

That all distinct sickness reports must be interpreted as being done by different agents shows the completeness of the choice in step 1a. Steps 1b and 2a are directly enforced by α. For step 2b, we have an ephemeral ID e for an agent x, such that e is not contained in any sick-report. By $dayof[e] = d$ we can check all sick reports that have been done on day d or before, and which agents we have reported there according to a given model $\mathcal{I}$, which the algorithm calls the set Agent_s. Suppose $\mathcal{I}(x) \in \mathsf{Agent}_s$, i.e., there is a sick report for an agent x' and $\mathcal{I}(x') = \mathcal{I}(x)$ that has at least one ephemeral id e' that is included in the

sick report for day $d' \leq d$. If $d = d'$, this contradicts the axiom that an agent releases all their ephemeral IDs for a given sick day, because we were considering an e that was not reported sick. If $d' < d$, this contradicts the axiom that the agent stops using the app after the sick report, i.e., $dayof[e]$ must be before the sick report. Finally, we have to show that also $\mathcal{I}(x) \in \mathsf{Agent}_e$ is not possible, because Agent_e contains all agents for which we have interpreted already P different ephemeral IDs for this day. This directly follows from the axiom that there are at most P different ephemeral IDs for the same agent on the same day. This shows that the choice in step 2b of an agent outside Agent_s and Agent_e is complete.

Hence, the algorithm allows all choices that are not excluded by α itself, and is thus complete. □

This characterization of the models of α of any reachable state allows us to prove dynamic possibilistic (α, β)-privacy as follows.

Theorem 5.13. *DP-3T with the extended α specification given in this section satisfies dynamic possibilistic (α, β)-privacy.*

Proof. We have to show that in every reachable state, any model $\mathcal{I}_0$ of α can be extended to a model $\mathcal{I}$ of β. Note that β must have a model $\mathcal{I}_r$ that corresponds to what really happened (and it is also a model of α). The idea is that we incrementally construct $\mathcal{I}$ close to $\mathcal{I}_r$.

First, we choose a key from SK for every agent a and every day d that occur in β; let us call it $\mathsf{sk}_{a,d}$. The principle here is: if, according to $\mathcal{I}$, agent a declares sick at some point, then β will contain the publication of the corresponding day keys of some agent x, where $\mathcal{I}(x) = a$. So, we have to set $\mathsf{sk}_{a,d}$ for those days d and a accordingly. All remaining keys can be set to arbitrary distinct values from SK, disjoint from those occurring in β. $\mathsf{sk}_{a,d} = \mathsf{sk}_{b,c}$ implies $a = b$ and $c = d$ by construction now, so set $\mathcal{I}(\mathsf{sk}_0[a]) = \mathsf{sk}_{a,0}$, and $\mathcal{I}(h[\mathsf{sk}_{a,d}]) = \mathsf{sk}_{a,d+1}$ for any agent a and day d occurring in β.

For prg, we can already pick some values in a convenient way: for those sk that are part of a sick report (i.e., not arbitrarily chosen from SK in the previous step), we can choose the ephemeral IDs derived from them to be identical to those in $\mathcal{I}_r$, i.e., set $\mathcal{I}(prg[\mathsf{sk}, i]) = \mathcal{I}_r(prg[\mathsf{sk}, i])$ for every period $i \in Period$ and every day key sk that is covered by a sickness report. The remaining ephemeral IDs (that did not occur in sickness reports) will be chosen "on the fly" now. It is yet to be proved that this is consistent with the rest of β.

For the initial state, we have thus an "intruder interpretation", i.e., what the

initial value of the memory cells $\mathsf{sk}_l(a)$ and $\mathsf{sk}(a)$ of every agent a is, namely $\mathcal{I}(\mathsf{sk}_0[a])$ and $\mathcal{I}(h^l[\mathsf{sk}_0[a]])$, respectively (while the real initial values are $\mathcal{I}_r(\mathsf{sk}_0[a])$ and $\mathcal{I}_r(h^l[\mathsf{sk}_0[a]])$). The intruder cannot see all the concrete values sk that occur here: the intruder can only see those values that have been explicitly released and apply the hash function further to them. Let us speak in the following of the *virtual state* of the memory cells, i.e., what value they would have (after a given sequence of transaction) if $\mathcal{I}$ were the reality.

The next day and the next period transactions just change the state; the virtual state is changed in a way that is completely determined by what we have determined in $\mathcal{I}$ so far.

For an agent advertisement transaction, let x be the variable for the agent in the transaction and $\mathcal{I}(x) = a$ the concrete agent according to $\mathcal{I}$ and e the ephemeral ID advertised. Let further sk, i, and d be the current values of $\mathsf{sk}(a)$, $\mathit{period}()$ and $\mathit{today}()$ in the virtual state. We distinguish two cases: first, if sk is a day key published in a sick report later, then we have already determined $\mathcal{I}(\mathit{prg}[\mathsf{sk}, i]) = \mathcal{I}_r(\mathit{prg}[\mathsf{sk}, i])$ previously, and $\mathcal{I}_r(\mathit{prg}[\mathsf{sk}, i]) = e$ because this is indeed the advertisement of the agent $\mathcal{I}_r(x)$ (which may have a name different from $\mathcal{I}(x)$) at this day and time period and sk is indeed the current day key this agent. Otherwise, if sk is not reported sick later, then $\mathcal{I}(\mathit{prg}[\mathsf{sk}, i])$ is not yet determined, unless we run the same advertisement a second time for the same agent on the same day and time period, and so it is already set to e, and we can set it to e. This is possible since in every other reached virtual state, sk and i are necessarily different, so $\mathit{prg}[\mathsf{sk}, i]$ has not yet been assigned a different interpretation yet. The formula β now contains (for an appropriate label m): $\mathit{concr}[m] = e \wedge \mathit{struct}[m] = \mathit{prg}[h^d[\mathsf{sk}_0[x]], i]$. This is because d and i in the virtual state are equal to the value in reality. Under $\mathcal{I}$, the struct term thus also equals e. We show below also for the other transitions that on every introduced label m it holds that $\mathcal{I} \models \mathit{concr}[m] = \mathit{struct}[m]$, and thus concr and struct will be trivially in static equivalence under $\mathcal{I}$.

For a sick report, let x be the variable for the agent in the transition and $\mathcal{I}(x) = a$ the concrete agent according to $\mathcal{I}$, and let sk_l, i, and d be the current values of $\mathsf{sk}_l(a)$, $\mathit{period}()$, and $\mathit{today}()$ in the current virtual state. The formula β now contains $\mathit{concr}[m] = \mathsf{sk}_l$ and $\mathit{struct}[m] = h^{d-l}[\mathsf{sk}_0[x]]$. Observe also here that we have $\mathcal{I} \models \mathit{concr}[m] = \mathit{struct}[m]$ because $\mathsf{sk}_l(x)$ is x's key from l days ago. □

5.4 Voting Protocols

In the previous chapter, we have modeled privacy goals for voting protocols with static possibilistic (α, β)-privacy. To illustrate the expressive power of our approach, we show how to adapt and formalize these privacy goals, namely voting privacy and receipt-freeness, with our dynamic extension of (α, β)-privacy. We show the formalization of these goals with a simple voting protocol. Indeed, while voting protocols can use more complicated cryptographic primitives, such as homomorphic encryption or blind signature as we have seen in the previous chapter, the principles behind the formalization of the goals remain similar. The state we discuss at the end of this section is rather complex, but the specification of the privacy goals is actually simple and intuitive with (α, β)-privacy, and we obtain a reachability problem out of this.

In our example, we consider a finite set of n voters $\mathsf{Voters} = \{x_1, \ldots, x_n\}$, two candidates candA and candB, and a trusted third-party a that acts both as the administrator of the election and the counter. Let $\mathsf{scrypt}/2$, $\mathsf{dscrypt}/2$ and $\mathsf{sk}/2$ be public functions, modeling symmetric encryption with a secret key. We consider the algebraic equation $\mathsf{dscrypt}(\mathsf{sk}(a,b), \mathsf{scrypt}(\mathsf{sk}(a,b), m)) \approx m$. Let $\mathsf{ballot}/2$ and $\mathsf{open}/1$ be two public functions modeling the ballot as a message format with the algebraic function $\mathsf{open}(\mathsf{ballot}(t_1, t_2)) \approx t_1, t_2$. We assume that each voter x_i shares an encryption key with the administrator, $\mathsf{sk}(x_i, a)$. We also assume that the intruder knows the key of dishonest voters, i.e., if x_i is dishonest, the intruder knows $\mathsf{sk}(x_i, a)$. We consider four families of memory cells: one for the status of the election $\mathsf{status}(\cdot)$, one for the status of a voter $\mathsf{voted}(\cdot)$, one for recording that the administrator accepts the vote of a voter $\mathsf{cast}(\cdot)$, and one for the result of a candidate $\mathsf{result}(\cdot)$. The initial values are voting, no, no and 0, respectively. Each voter X owns their own cell $\mathsf{voted}(X)$ and the administrator owns the three other entire families, i.e., the election public information. The administrator can update the status of the election to over when the tallying phase starts. A voter X updates their status $\mathsf{voted}(X)$ to yes when they vote, and the administrator updates $\mathsf{cast}(X)$ to yes when he accepts the vote of a voter X. Finally, the administrator updates the result of a candidate, say candA, with the memory cell $\mathsf{result}(\mathsf{candA})$, each time a valid vote for this candidate is counted. Moreover, let $v/1$ and $c/1$ be two interpreted functions that model respectively the voting function and the check for counted votes. Finally, we allow for dishonest voters, and we define the predicate $\mathsf{dishonest}$. For every dishonest $X \in \mathsf{Voters}$, $\mathsf{dishonest}(X)$ holds, and, conversely, for every honest $X \in \mathsf{Voters}$, $\neg\mathsf{dishonest}(X)$ holds.

The election is divided in two phases. The voting phase is modeled by the *Cast* and *Admin* processes in Figure 5.7. During a *Cast* process transaction, a honest voter X can choose their vote V. If the voter did not vote already, a new random

Cast

if status$(\cdot) \doteq$ voting then
  $\star\ X \in$ Voters.
  if $(\neg$dishonest$(X))$ then
    $\star\ V \in \{$candA, candB$\}$.
    $s :=$ voted(X).
    if $(s \doteq$ no$)$ then
      $\nu R.$ voted$(X) :=$ yes.
      $\diamond\ v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.
      snd(scrypt(sk(X, a), ballot(R, V)))

Admin

if status$(\cdot) \doteq$ voting then
  rcv(scrypt(sk(X, a), ballot(R, V))).
  $s :=$ cast(X).
  if $(X \in$ Voters $\wedge\ s \doteq$ no$)$ then
    cast$(X) :=$ yes.
    if $($dishonest$(X))$ then
      $\diamond\ v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.
      $\star\ v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$.

Counter

status$(\cdot) :=$ over.
for$X$: Voters
  $s :=$ cast(X).
  if $(s \doteq$ yes $\wedge\ v[\mathcal{I}(X)] \doteq$ candA$)$ then
    result $:=$ result(candA).
    result(candA) $:=$ result $+ 1$.
    $\diamond\ c[\mathcal{I}(X)] \doteq$ true.
  if $(s \doteq$ yes $\wedge\ v[\mathcal{I}(X)] \doteq$ candB$)$ then
    result $:=$ result(candB).
    result(candB) $:=$ result $+ 1$.
    $\diamond\ c[\mathcal{I}(X)] \doteq$ true.
result$_A :=$ result(candA).
result$_B :=$ result(candB).
if (result$_A \not\doteq$ result_candA $\vee$ result$_B \not\doteq$ result_candB) then
  attack.
else
  $\star$ result_candA $\doteq \mathcal{I}($result_candA$) \wedge$ result_candB $\doteq \mathcal{I}($result_candB$)$.

$$\text{result\_candA} = \Sigma_{X \in \text{Voters} \wedge v[X] \doteq \text{candA} \wedge c[X] \doteq \text{yes}} 1$$
$$\text{result\_candB} = \Sigma_{X \in \text{Voters} \wedge v[X] \doteq \text{candB} \wedge c[X] \doteq \text{yes}} 1$$

Figure 5.7: Voting Protocol

value R is generated. Their status $\mathsf{voted}(X)$ is updated to yes. We publish in γ the true value of their vote, i.e., $v[\mathcal{I}(X)] \doteq \mathcal{I}(V)$: this is used later in the tallying phase to formalize the core of the privacy goal. Finally, the voter sends their vote to the administrator that they encrypt with their shared secret key. Note that this transaction can only be taken for honest voters. The intruder can send any messages that he wants, but ultimately, for the dishonest voters that he controls, he has to produce a vote that the administrator later accepts if he wants the vote to be counted.

During a *Admin* process transaction, every time the administrator receives a ballot from a voter X, they first check that the sender is a legit voter, and they also check that the legit voter did not cast a vote already by checking $\mathsf{cast}(X)$. If these requirements are met, the administrator updates $\mathsf{cast}(X)$ to yes. Besides, if the voter is dishonest, i.e., if $\mathsf{dishonest}(X)$ holds, the vote is also disclosed in γ and in α. This does not mean that the intruder necessarily knows the value of the vote, but that it should not count as an attack if he learns it. This can be seen as a sort of declassification: the relation between a dishonest voter and their vote is not a secret. This is important in systems like an early version of Helios, where an intruder can cast a copy of another voter's vote as his own vote. It would be obscuring the attack from our analysis, if we simply gave to the intruder the information what his vote is; leaving this information classified would lead to a false positive in general (because typically the intruder knows what he voted). (α, β)-privacy thus allows us to stay clear of both problems by just declassifying the relation between dishonest voter and their vote. This is somewhat analog to our declassification definition that we introduced in Chapter 2.

At the beginning of a *Counter* process transaction, the administrator who is also acting as the counter can switch the status of the election to over. The administrator is going through all the voters. We use a for construct as a syntactic sugar. We need to unroll this loop, i.e., repeat the body for each voter. This notational sugar allows us to keep our formalization parametrized over an arbitrary number of voters, while a concrete specification that results from unrolling this loop has the number of voters fixed. This is because we do not wish to formalize an unbounded number of steps in a single step, which would have undesirable consequences on the semantics. For each voter, the administrator checks whether they cast a vote that has been accepted. If this is the case, a distinction is made following the value of the interpreted function $v[\mathcal{I}(X)]$ stored in γ. This does not represent that the administrator knows the value of the vote, but that we make a case distinction depending of what the truth γ is. Depending on $v[\mathcal{I}(X)]$ being candA or candB, the administrator updates the result memory cell for the corresponding candidate. Finally, the administrator sets the check for counted vote of the voter to true. Once this is done for all voters, the administrator checks that the result is correct. We

encode this correctness property for candA for instance by $\mathsf{result}_A \neq \mathsf{result\_candA}$. $\mathsf{result\_candA}$ is computed directly from the information published in γ, i.e., $\mathsf{result\_candA} = \sum_{X \in \mathsf{Voters} \wedge v[X] \doteq \mathsf{candA} \wedge c[X] \doteq \mathsf{true}} 1$, whereas result_A is the actual computation done by the administrator. Again, this does not mean that the administrator knows the value of each individual vote, but this means that we require that the result the counter computes corresponds to the truth. If this is not the case, there is an attack, i.e., it triggers a violation of (α, β)-privacy, since the basic correctness of the protocol is shown violated if this branched is reached. Otherwise, we can publish the result in α. We would like to emphasize the key role of the information published in γ through interpreted functions to express the privacy goals of a voting protocol (similarly to what was done in the static approach in the previous chapter).

To give another example on how dynamic possibilistic (α, β)-privacy works, we show how to reach a state that we call "final", i.e., a state where the result has been printed and no new transactions can be taken (see Figure 5.8). For this example, we consider that there are three voters, i.e., $\mathsf{Voters} = \{x_1, x_2, x_3\}$. x_1 and x_2 are honest voters, and x_3 is dishonest, i.e., $\mathsf{dishonest}(x_3)$ holds. For the sake of simplicity, we consider that first the three voters cast their vote, and then only that the administrator registers their votes.

This means that a *Cast* process transaction is taken two times for the honest voters. For the two instantiations (lines $1, 2$ in Figure 5.8), public information about the voter is released in α, the intruder can observe the exchange of messages with the administrator, the true value of the votes is released in γ, and the status of each of the voters is updated to yes. Note that the intruder can send whatever messages that he can compose, especially he is able to send a valid vote to the administrator since he knows $\mathsf{sk}(x_3, a)$.

Then, a process *Admin* transaction is also taken three times. This is reflected for the three instantiations (line $3, 4, 5$ in Figure 5.8) by the update of $\mathsf{cast}(X)$ to yes. Note that for the dishonest voter x_3, the true result of the vote is released in both γ and α, since this is a vote that was produced by the intruder. Again, this does not mean that the intruder learns the value of the vote at this point, but that it will not count as an attack if he does.

Finally, the only transaction still possible is the instantiation of a *Counter* process. This means the administrator is going through the votes and updates the result. Every time the administrator counts the vote of a voter, it is released in γ that the vote has been counted. At the end of the process, the result of the election is released in α.

Before concluding on the security of this "final" state, we need to recapitulate the privacy goals that we expressed. Let us have a look at the information we

	α	β	γ	δ
1	$X_1 \in \mathsf{Voters}$ $\wedge\ V_1 \in \{\mathsf{candA}, \mathsf{candB}\}$	$concr[l_1] = \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA}))$ $\wedge\ struct[l_1] = \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1))$	$X_1 \doteq x_1$ $\wedge\ V_1 \doteq \mathsf{candA}$ $\wedge\ v[x_1] \doteq \mathsf{candA}$	$\mathsf{voted}(X_1) := \mathsf{yes}$ if ϕ_1
2	$X_2 \in \mathsf{Voters}$ $\wedge\ V_2 \in \{\mathsf{candA}, \mathsf{candB}\}$	$concr[l_2] = \mathsf{scrypt}(\mathsf{sk}(x_2, a), \mathsf{ballot}(R_2, \mathsf{candB}))$ $\wedge\ struct[l_2] = \mathsf{scrypt}(\mathsf{sk}(X_2, a), \mathsf{ballot}(R_2, V_2))$	$X_2 \doteq x_2$ $\wedge\ V_2 \doteq \mathsf{candB}$ $\wedge\ v[x_2] \doteq \mathsf{candB}$	$\mathsf{voted}(X_2) := \mathsf{yes}$ if $\phi_1 \wedge \phi_2$
3				$\mathsf{cast}(X_1) := \mathsf{yes}$ if ϕ_3
4				$\mathsf{cast}(X_2) := \mathsf{yes}$ if $\phi_3 \wedge \phi_4$
5	$v[x_3] \doteq \mathsf{candA}$		$v[x_3] \doteq \mathsf{candA}$	$\mathsf{cast}(X_3) := \mathsf{yes}$ if $\phi_3 \wedge \phi_4 \wedge \phi_5$
6	$\mathsf{result\_candA} \doteq 2$ $\wedge\ \mathsf{result\_candB} \doteq 1$		$c[x_1] \doteq \mathsf{true}$ $\wedge\ c[x_2] \doteq \mathsf{true}$ $\wedge\ c[x_3] \doteq \mathsf{true}$	$\mathsf{status}(\cdot) := \mathsf{over}$ if ϕ_6 $\mathsf{result}(\mathsf{candA}) := 2$ if $\phi_6 \wedge \phi_{vote}$ $\mathsf{result}(\mathsf{candB}) := 1$ if $\phi_6 \wedge \phi_{vote}$

$\phi_1 \equiv \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_1 \doteq \mathsf{no} \wedge \neg\mathsf{dishonest}(X_1)$ $\quad$ $\phi_2 \equiv \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_2 \doteq \mathsf{no} \wedge \neg\mathsf{dishonest}(X_2)$

$\phi_3 \equiv \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge X_1 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_3 \doteq \mathsf{no}$

$\phi_4 \equiv X_2 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_4 \doteq \mathsf{no}$ $\quad$ $\phi_5 \equiv X_3 \in \mathsf{Voters} \wedge \mathsf{status}(\cdot) \doteq \mathsf{voting} \wedge s_5 \doteq \mathsf{no}$

$\phi_6 \equiv \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge s_6 \doteq \mathsf{yes} \wedge s_8 \doteq \mathsf{yes} \wedge s_9 \doteq \mathsf{yes}$

$\phi_{vote} \equiv (v[x_1] \doteq \mathsf{candA} \wedge v[x_2] \doteq \mathsf{candB} \wedge v[x_3] \doteq \mathsf{candA}) \vee (v[x_1] \doteq \mathsf{candB} \wedge v[x_2] \doteq \mathsf{candA} \wedge v[x_3] \doteq \mathsf{candA})$

Figure 5.8: Execution of the voting protocol

have intentionally released, i.e., the formula α:

$$\begin{aligned}\alpha \equiv\ & X_1 \in \mathsf{Voters} \wedge V_1 \in \{\mathsf{candA}, \mathsf{candB}\} \\ & \wedge\ X_2 \in \mathsf{Voters} \wedge V_2 \in \{\mathsf{candA}, \mathsf{candB}\} \\ & \wedge\ v[x_3] \doteq \mathsf{candA} \\ & \wedge\ \mathsf{result\_candA} \doteq 2 \wedge \mathsf{result\_candB} \doteq 1\end{aligned}$$

α has just two models (before the release of the results, it has four models). Also, observe that the intruder can now deduce that V_1 and V_2 are different, and that is a legal consequence of α. This is something that the cardinality of the election allows, and the best protocol cannot prevent it. Let us have now a look on the technical information: β is the conjunction of ϕ_{gen}, ϕ_{hom}, $\phi_{\sim}$ (see preliminaries) and the following frames *concr* and *struct* (note that x_i are concrete voters and that X_i are the privacy variables picked during transitions):

$$\begin{aligned}\mathit{concr} = \{\!|\ & l_1 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA})), \\ & l_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_2, a), \mathsf{ballot}(R_2, \mathsf{candB}))|\!\} \\ \mathit{struct} = \{\!|\ & l_1 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1)), \\ & l_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_2, a), \mathsf{ballot}(R_2, V_2))|\!\}\end{aligned}$$

The payload formula α specifies that the intruder can learn that the three voters voted (he can indeed observe that there are three votes on the bulletin board) and the result of the election. As explained, he can also learn the true vote of voter x_3 (in our specific case, he knows it since he knows $\mathsf{sk}(x_3, a)$). But he should not learn more. The technical information β takes into account the messages that the intruder observed. In any instantiations of the variables that is compatible with α, *concr* and *struct* are statically equivalent, since the intruder cannot decrypt anything else that the vote of the dishonest voter (or compose and check other messages). Thus, all the models of α can be extended to models of β, and this simple voting protocol ensures voting privacy in its "final" state.

Now let us say that we want this protocol to ensure stronger privacy goals, such as *receipt freeness*. We gave the following definition in the previous chapter: "no voter has a way to prove how they voted". The property is formalized with respect to a specific voter, say x_1, that the intruder is trying to influence. The question is whether voter x_1 can *prove* to the intruder how they voted by a kind of "receipt". The idea is to force the coerced voter to reveal their *entire knowledge*. But the voter can lie and give to the intruder anything that they can construct from their own knowledge, as long as their *story* is consistent with what the intruder already knows, e.g., from observing the messages exchanged between the voters and the administrator. Similarly to the frames *concr* and *struct* that represent the knowledge of the intruder, we reason about the frames concr_{x_1} and struct_{x_1} that represent the knowledge of voter x_1. The core idea is

then that what voter x_1 can lie about is the content of $concr_{x_1}$. We augment β by the following axiom ϕ_{lie}, that is updated for every transition as $\phi_\sim$ for instance, and where $\{d_1, \ldots, d_l\}$ is the domain of the frames $concr_{x_1}$ and $struct_{x_1}$:

$$\begin{aligned}\phi_{lie} \equiv\ & struct[d_1] = struct_{x_1}[d_1] \wedge \ldots \wedge\ struct[d_l] = struct_{x_1}[d_l] \\ & \wedge\ \exists s_1, \ldots, s_l.\, gen_{D_{x_1}}(s_1) \wedge\ gen_{D_{x_1}}(s_l). \\ & (concr[d_1] = concr_{x_1}[s_1] \wedge \ldots \wedge\ concr[d_l] = concr_{x_1}[s_l]])\end{aligned}$$

In our simple voting protocol, the knowledge of the voter x_1 is very simple. They know their shared key with the administrator, the random value that they generate when voting and they know the messages that they send:

$$\begin{aligned}concr_{x_1} =\ & \{\!|\ d_1 \mapsto \mathsf{sk}(x_1, a), d_2 \mapsto R_1, \\ & \quad d_3 \mapsto \mathsf{scrypt}(\mathsf{sk}(x_1, a), \mathsf{ballot}(R_1, \mathsf{candA}))\ |\!\} \\ struct_{x_1} =\ & \{\!|\ d_1 \mapsto \mathsf{sk}(X_1, a), d_2 \mapsto R_1, \\ & \quad d_2 \mapsto \mathsf{scrypt}(\mathsf{sk}(X_1, a), \mathsf{ballot}(R_1, V_1))\ |\!\}\end{aligned}$$

There is no way for the voter x_1 to lie about their true vote because they know neither the secret key that the other voters share with the administrator, nor the other random values. This means that the intruder can exclude some models of α when we require as part of β the receipt-freeness axiom: this protocol is not receipt-free.

Receipt-freeness is a difficult property to express with approaches such as the ones based on Applied-π calculus [DKR06]. We have shown here how to refine our privacy goal in dynamic (α, β)-privacy. Note that was done with an additional axiom to β rather than something we could already express with (α, β)-privacy directly.

5.5 Comparison with Trace Equivalence Approaches

The gold standard for privacy in security protocols are the notions of *observational equivalence* and *trace equivalence* (see, e.g., [DH17] for a survey). Roughly, a pair of processes is trace equivalent if all transitions of one process can be simulated by the other. This entails substantial difficulties for automated verification [CCD17], especially when systems have a long-term mutable state [Ara+17], but still privacy notions are typically formulated as such an equivalence between two alternative worlds, rather than reachability problem. Interesting in this context is the notion of *diff-equivalence* [BAF08] that is implemented in the most popular verification tools ProVerif and Tamarin: here the processes are

parametrized over a binary choice in terms and this gets close to a reachability problem, because the processes are practically in lockstep. This is, however, so restrictive that only a very limited class of privacy properties can be considered.

(α, β)-*privacy* was introduced in [MV19] as a more declarative way to formalize and reason about privacy than the indistinguishability of two alternatives. There is an underlying notion of equivalence in (α, β)-privacy though: the static equivalence of the frames *concr* and *struct*. These describe not two alternative worlds but rather different levels of knowledge of the intruder: the concrete messages and the structural knowledge. However, (α, β)-privacy is a static notion, describing a fixed state of the world, and does not reason about the interaction of the intruder with his environment. We showed in this chapter how to lift (α, β)-privacy to full-fledged transition systems, and have thus re-cast privacy as a reachability problem without the limitations that come, e.g., in diff-equivalence.

In a nutshell, all that can be expressed with trace equivalence, and more, can be expressed with (α, β)-privacy. We now give a formal comparison[5].

5.5.1 Visibility of Transactions

It is inherent in the semantics of (α, β)-privacy that the intruder knows which transaction is currently being executed; but the intruder does *not* know which of the if-then-else branches is taken, unless this can be inferred from the communication behavior of the transaction. In contrast, most trace-based approaches are formulated in a variant of the Applied-π calculus and do not have a notion of transaction in the first place; the intruder view is thus limited to the communication behavior.

If desired, it is easy to express the same limited intruder view in (α, β)-privacy transactions:[6] given a specification of transactions $T_1, \ldots, T_n$, one can transform them into a single transaction T as follows (where z is a variable that does not

[5]The reader should bear in mind that trace equivalence and (α, β)-privacy are two quite different "games", so bridging between them often leads to constructions, and requires restrictions, that are somewhat artificial, but that at least give an idea of how the two approaches relate.

[6]It is similarly possible to equip a process calculus specification with additional messages that tell the intruder a particular point has been reached.

occur in any of the T_i):

$$\begin{array}{l}\diamond\ z \in \{1, \ldots, n\}.\\ \mathsf{if}\ (z \doteq 1)\ \mathsf{then}\ T_1.\\ \mathsf{else\ if}\ (z \doteq 2)\ \mathsf{then}\ T_2.\\ \ldots\\ \mathsf{else\ if}\ (z \doteq n)\ \mathsf{then}\ T_n\end{array}$$

This transaction allows all the same behaviors as the T_is together, except that the intruder does not see a priori which of the T_is is taken. Depending on the output messages of the T_is, the intruder may anyway find out which T_i it is (or just narrow it down to a few candidates), but that in itself is not a violation of privacy since the non-deterministic choice of z was not released in α (and thus learning the value of z does not exclude any models of α).

In our opinion, it is better to let the intruder know the transaction by default, and have the modeler explicitly specify otherwise (with the above construction), when the protocol privacy indeed relies on this. This makes it less likely that such a reliance is overlooked upon implementation. For the rest of this discussion, we will speak of transactions $T_1, \ldots, T_n$, but allowing for the case that $n = 1$ with the above construction.

5.5.2 Restrictions

We consider two restrictions (R1) and (R2) that do not seem utterly necessary, but greatly simplify the exposition. (R1): for this discussion, we consider (α, β)-privacy without interpreted functions except *concr* and *struct* and without relation symbols except *gen*. Hence, there are only the following "sources" of non-determinism:

- variables that are introduced as $\star\ x \in D$;[7] let us call such an x an α-*variable* (because it is part of α),
- variables that are introduced as $\diamond\ y \in D$; let us call such a y a β-*variable* (because it is *not* part of α),
- the non-determinism of the transition relation itself, i.e., in a sequence of steps, which transaction is performed next, and

[7] We also ignore the probabilistic aspect in this section as it is not part of the common trace equivalence notions.

- for a transaction that receives a message, which of all available messages is received.

Thus, for a given choice of transactions to perform and recipes of the intruder to send for the inputs, the α- and β-variables are the only non-determinism.

(R2): we restrict transactions to having exactly one input and one output (on every path through its if-the-else conditions). This simplifies the problem as the intruder may not directly infer anything about the conditionals from the number of messages sent or received by a transaction; of course, the intruder may still be able to conclude from the observed output which path was taken by a transaction. Hence, this is not a significant restriction in practice. In a trace of k steps, the intruder has to give k inputs and receives k outputs. Thus, in each reached state after k steps, every *struct* frame and the *concr* frame have the same domain $\{l_1, \ldots, l_k\}$, where each l_i labels the output of the i-th transaction. Similarly, the input from the intruder to each transaction is thus simply a recipe r_i which uses only labels $\{l_0, \ldots, l_{i-1}\}$, i.e., all outputs received so far.

Definition 5.14. *Given a transaction specification with the restrictions (R1) and (R2), we define a* trace *tr as a tuple* $((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$*, where*

- *each* a_i *identifies one of the transactions,*
- *each* r_i *is an intruder recipe over labels* $\{l_1, \ldots, l_{i-1}\}$*, and*
- $(\mathcal{S}, \mathcal{P})$ *is any configuration reached by the given sequence of transactions when the inputs are bound to the* r_i *and the outputs labeled* l_i*. (This is according to our definition of transaction semantics in Section 5.1.2.)*

We refer to the $\alpha(\mathcal{S})$*,* $\beta(\mathcal{S})$*, and* $\gamma(\mathcal{S})$ *of a trace as expected; we may also refer to the* $\mathrm{concr}(\mathcal{S})$ *of a trace, i.e., the (unique) ground messages bound to the labels* l_i *according to* $\beta(\mathcal{S})$*.*

We call a sequence $(a_1, r_1), \ldots, (a_k, r_k)$ *a* symbolic trace *that represents all those traces that have this sequence of* (a_i, r_i) *transactions and inputs. The set of represented traces is finite, corresponding to the possible interpretations of the non-deterministic* α *and* β *variables.*

We say that (α, β)*-privacy* holds *in a trace* $((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$ *if it holds in state* $\mathcal{S}$*, and that it* holds *in a symbolic trace tr if it holds in all traces represented by tr.*

We call two traces $tr = ((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}, \mathcal{P}))$ *and* $tr' = ((a_1, r_1), \ldots, (a_k, r_k), (\mathcal{S}', \mathcal{P}'))$ equivalent*, and write* $tr \approx tr'$*, if* $concr(\mathcal{S}) \sim concr(\mathcal{S}')$ *(and, as indicated by pattern matching, the* a_i*,* r_i*, and* k *are the same).*

Let $traces(Spec)$ *be the set of traces produced by a specification of* (α, β)*-privacy transactions. We call two specifications Spec and* $Spec'$ trace equivalent*, and write* $Spec \approx Spec'$*, if for every trace* $tr \in traces(Spec)$*, there is a* $tr' \in traces(Spec')$ *with* $tr \approx tr'$*, and vice versa.*

A binary privacy question *is a specification of* (α, β)*-privacy transactions that do not contain any* α*-variables and make no* α*-release, together with a special transaction* $T_{bin} = \mathsf{if}\ (\mathsf{init} \doteq \bot)\ \mathsf{then}\ \star\ x \in \{0,1\}.\ \mathsf{init} := x$*, where* init *is a distinguished memory cell initialized to* $\bot$ *and the other transactions may only read, but not modify, the value of* init*.*

The traces represented by a symbolic trace are actually easy to compute thanks to the restrictions (R1) and (R2): we follow the normal semantics, but for every step "$\star\ x \in D_x$" and for every step "$\diamond\ y \in D_y$", we keep the choice symbolic, and compute a set of corresponding α and γ that we attach to the respective possibility $(\mathcal{P}_i, \phi_i, struct_i)$ in the configurations. The δ is the same for all, and the β can be reconstructed from γ and the configuration. This is taking advantage of the fact that we already have a representation for all the possibilities (the $(\mathcal{P}_i, \phi_i, struct_i)$) at a given point. Now, there is however no possibility $(\mathcal{P}_i, \phi_i, struct_i)$ marked, but that marking is actually only needed in case the different possibilities have differences in the number of sent and received messages, which we do not consider here due to the restrictions (R1) and (R2).

Note that every trace has at least one interpretation since every if-then-else has at least one branch that can execute, i.e., every transaction is applicable in every trace (it may just fail to actually do something).

This definition expresses the fact that trace equivalence is about the ability to distinguish between two systems that each reflect a particular choice of the privacy information. Relating this to the terms of (α, β)-privacy means thus that α is simply the secrecy of a bit x. We can now relate (α, β)-privacy in the binary case with trace equivalence (we first prove Theorem 5.16 as it will come in handy to prove Theorem 5.15):

Theorem 5.15. *Consider a binary privacy question Spec that meets (R1) and (R2). For each* $b \in \{0,1\}$*, let* $Spec_b$ *be the specialization of Spec where* T_{bin} *sets the choice of* x *to* $\{b\}$*. Then* (α, β)*-privacy holds in Spec iff* $Spec_0 \approx Spec_1$*.*

Here, one can see two fundamental differences between (α, β)-privacy and the trace equivalence approach: in trace equivalence, we do not have to introduce a

distinction between high-level and low-level (but we simply have a single bit a secret); on the other hand, we cannot express more than a binary choice between two systems in one go: of course one can specify several binary questions, but each is an independent binary question. In contrast, in (α, β)-privacy we can have a choice between any finite number of models and we can let this develop during transitions, also dependent on the actions of the intruder. For this reason, we also formulate a different equivalence notion that is based on traces, but that, instead of distinguishing two systems, is based on the models of a formula α in a single system:

Theorem 5.16. *(α, β)-privacy holds in a symbolic trace $tr = (a_1, r_1), \ldots, (a_k, r_k)$ iff for every trace $(tr, (\mathcal{S}, \mathcal{P}))$ and every Σ_0-interpretation $\mathcal{I}_0 \models \alpha(\mathcal{S})$, there exists a trace $(tr, (\mathcal{S}', \mathcal{P}'))$ such that $\mathcal{I}_0 \models \gamma(\mathcal{S}')$ and $concr(\mathcal{S}) \sim concr(\mathcal{S}')$.*

Proof. Let $tr = (a_1, r_1), \ldots, (a_k, r_k)$ and first suppose (α, β)-privacy is violated in tr, i.e., for some trace $(tr, (\mathcal{S}, \mathcal{P}))$, (α, β)-privacy is violated in $\mathcal{S}$. This means that there is one model $\mathcal{I}_0$ of $\alpha(\mathcal{S})$ that cannot be extended to a model of β, i.e., for every $(\mathcal{P}_i, struct_i, \phi_i) \in \mathcal{P}$, either $\mathcal{I}_0 \not\models \phi_i$ or the $\mathcal{I}_0(struct_i) \not\sim concr(\mathcal{S})$. Thus, the intruder can exclude in state $\mathcal{S}$ every trace $(tr, (\mathcal{S}', \mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$. Since only the α- and β-variables are to interpret, this means that in every trace $(tr, (\mathcal{S}', \mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$, we have $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$.

Vice-versa, suppose there is a trace $(tr, (\mathcal{S}, \mathcal{P}))$ and a model $\mathcal{I}_0$ of $\alpha(\mathcal{S})$ such that for every trace $(tr, (\mathcal{S}', \mathcal{P}'))$ where $\mathcal{I}_0 \models \gamma(\mathcal{S}')$, $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$. Then, similarly, for every $(\mathcal{P}_i, struct_i, \phi_i) \in \mathcal{P}$, either $\mathcal{I}_0 \not\models \phi_i$ or $\mathcal{I}_0(struct_i) \not\sim concr(\mathcal{S})$. Thus, $(tr, (\mathcal{S}, \mathcal{P}))$ violates (α, β)-privacy. □

We can finally prove Theorem 5.15:

Proof. Note that *Spec*, $Spec_0$ and $Spec_1$ have the same set of symbolic traces. If a symbolic trace tr does not contain the special transaction T_{bin}, then all the concrete traces it represents in $Spec_0$, $Spec_1$ and *Spec* are also the same, so up to taking the special transaction, there is trivially no violation of (α, β)-privacy or trace distinction possible. Thus, for the rest of this theorem, we consider only a symbolic trace tr that includes the special transaction T_{bin}. Observe that in *Spec*, all concrete traces $(tr, \mathcal{S}, \mathcal{P})$ represented by tr have thus $\alpha(\mathcal{S}) \equiv x \in \{0, 1\}$.

Suppose now (α, β)-privacy holds in *Spec* and suppose $(tr, \mathcal{S}, \mathcal{P})$ is a trace that tr represents in $Spec_0$. Then, $\gamma(\mathcal{S})(x) \equiv 0$. This trace is also possible in *Spec*, and since the privacy holds, by Theorem 5.16, there exists a trace $(tr, \mathcal{S}', \mathcal{P}')$ in *Spec* that supports the other model of α, namely $\gamma(\mathcal{S}')(x) \equiv 1$, and such that $concr(\mathcal{S}) \sim concr(\mathcal{S}')$. By construction, $(tr, \mathcal{S}', \mathcal{P}')$ is a trace of $Spec_1$. Thus,

for every trace in $Spec_0$ exists an equivalent one $Spec_1$. By a similar proof, every trace in $Spec_1$ has an equivalent in $Spec_0$. Hence, $Spec_0$ and $Spec_1$ are trace equivalent.

Suppose now, for the sake of contradiction, that (α, β)-privacy is violated in *Spec*. Then, by Theorem 5.16, there exists a trace $(tr, \mathcal{S}, \mathcal{P})$ in *Spec*, say with $\gamma(\mathcal{S})(x) \equiv 0$ (the proof for the case $\gamma(\mathcal{S})(x) \equiv 1$ is analogous), and there is no trace $(tr, \mathcal{S}', \mathcal{P}')$ of *Spec* such that both $\gamma(\mathcal{S})(x) \equiv 1$ and $concr(\mathcal{S}) \sim concr(\mathcal{S}')$. Obviously, $(tr, \mathcal{S}, \mathcal{P})$ is a trace of $Spec_0$, but for all $(tr, \mathcal{S}', \mathcal{P}')$ of $Spec_1$, $concr(\mathcal{S}) \not\sim concr(\mathcal{S}')$ (since they have $\gamma(\mathcal{S})(x) \equiv 1$). Thus, $Spec_0$ and $Spec_1$ are not trace equivalent. □

5.6 Comparison with information flow

For many applications, it is interesting to take into account probabilities. We see no obvious way to reason with them in equivalence-based specifications, but the fact that every state in dynamic possibilistic (α, β)-privacy represents a single reality allows us to make a declarative extension that integrates nondeterminism and probabilistic aspects. In fact, the two theorems on extensible specification allow for adding probabilities and background knowledge to an otherwise possibilistic specification without further proofs.

Our work is also related to privacy approaches based on *non-interference* and *information flow* [GM82; MSS11; TJ11]. In fact, one may wonder if dynamic probabilistic (α, β)-privacy is not simply information flow in disguise. That is, one may think that α is basically akin to high variables, and β is akin to low variables. While there are similarities and parallels, there are also key conceptual differences. Most importantly, the β information is "low" in quite a different sense: it is regarded as technical information, but without stipulating whether it is privileged. For instance, β may contain cryptographic keys and nonces, and some nonces (and even some keys) may be obtained by the intruder without violating dynamic probabilistic (α, β)-privacy; in fact, knowing them does not in itself constitute a violation. However, sometimes even just knowing that two low-level messages are identical can be sufficient to deduce a violation, because it rules out some model of α.

Our explicit focus on probabilities bears similarities with approaches in *quantitative information flow* [Alv+20; CHM01; DHW04; Gru08; LMT10; Low02; Smi09], as well as *differential privacy* [Dwo08; YCW17], *k-anonymity* [Swe02], *l-diversity* [Mac+07], and *t-closeness* [LLV07], which aim at quantifying privacy to capture leaks or information disclosure in a system. Differential privacy,

in particular, is concerned with statistical queries to databases, i.e., how do you reveal accurate statistics about a set of individuals while preserving their privacy? In this approach, private and public information are not completely disjoint, e.g., one might want to reveal the average age of the participants in a survey, but if a new participant is added after the reveal of the average, then one can learn their age.

Differential privacy (and related approaches) and approaches for protocol security and privacy (like ours) are typically seen as two different disciplines that look at privacy in distinct, particular ways. We regard our work also as an attempt to start bridging this gap. While our approach is rooted in protocol verification, we have tried to import here notions that have been successful on the other side of the gap.

Still, it is important to note an interesting relationship between our work in this chapter and information flow. This comparison underlines the fact that in general, many information flow approaches are a form of static analysis that over-approximates and classifies many potential problems as violations, whereas (α, β)-privacy is usually more precise. The comparison with information flow is a bit more difficult, since it is farther away from (α, β)-privacy than trace equivalence, and thus we give a less formal comparison.

An intuition is that the semantics of transactions in (α, β)-privacy reflects what would be called explicit and implicit flows in information flow approaches: the messages sent out by a transaction are visible to the intruder. This may give rise to an explicit flow, for instance, if the sent message contains directly the value of an α-variable. It may give rise to an implicit flow, if the sent message depends on a condition and the intruder may thus be able to determine the value of the condition. In fact, an important part of the semantics is to formalize what the intruder knows about the shape of the message (i.e., *struct*) depending on the different outcomes of conditions taken so far.

For example, consider the following simple program:

```
if (x>10){
  y:=y*y;
}
else{
  y:=y*x;
}
```

Suppose x is of level H (high) and y of level L (low). Then, we obviously have two violations of information flow. We could model this also in (α, β)-privacy,

but since we are modifying variables, we would have to use here memory cells Xcell and Ycell. We can initialize these memory cells with an arbitrary value from, say, the set of integers. The privacy condition is that the intruder cannot find out anything about the initial value of x. However, he should be able to see the value of y before and after the transaction. The initialization can be formalized as follows (where init is a boolean memory cell, initialized to false):

$$
\begin{array}{l}
\mathsf{if}\ (\mathsf{init} \doteq \mathsf{false})\ \mathsf{then} \\
\quad \mathsf{init} \coloneqq \mathsf{true}. \\
\quad \star\ X \in \{0, \ldots, \mathsf{MAXINT}\}. \\
\quad \diamond\ Y \in \{0, \ldots, \mathsf{MAXINT}\}. \\
\quad \mathsf{Xcell} \coloneqq X. \\
\quad \mathsf{Ycell} \coloneqq Y. \\
\quad \mathsf{snd}(\mathsf{Ycell})
\end{array}
$$

and the actual program is:

$$
\begin{array}{l}
\mathsf{if}\ (\mathsf{init} \doteq \mathsf{true})\ \mathsf{then} \\
\quad \mathsf{if}\ (\mathsf{Xcell} > 10)\ \mathsf{then} \\
\quad\quad \mathsf{Ycell} \coloneqq \mathsf{Ycell} * \mathsf{Ycell}. \\
\quad \mathsf{else} \\
\quad\quad \mathsf{Ycell} \coloneqq \mathsf{Ycell} * \mathsf{Xcell}. \\
\quad \mathsf{snd}(\mathsf{Ycell})
\end{array}
$$

A possible trace is now that the intruder observes an initialization transaction with the concrete value 1 (for y) and then the same value again in a program transaction. Then, he can exclude the second branch (because y would have to be at least 11 for that), and thus knows $X > 10$, violating the (α, β)-privacy privacy goal.

Here we can however observe five major differences between (α, β)-privacy and information flow. The first major difference is that (α, β)-privacy will not trigger if there is no actual information flow that can be observed. In the above example, for instance, if y gets initialized with 0, then the result is not telling anything about x, and hence in that state, (α, β)-privacy holds. Thus, (α, β)-privacy does not share the false positives that can arise from the over-approximation of static information flow analysis. While it is nice to avoid false positives, the simplicity and efficiency of static analysis naturally raise the question whether this can also be an effective technique to analyze (α, β)-privacy problems and whether we can over-approximate in a similar way—a problem we must leave open at this point.

A second major difference is that (α, β)-privacy allows for sending also classified values on a public channel when they are encrypted such that the intruder cannot

decrypt them. While one can of course define an appropriate declassification to allow also for this in information flow, this is not suitable for the analysis of privacy in protocols, and one would need here (α, β)-privacy or trace equivalence approaches: this is because one needs to analyze in general whether the employed encryption regime indeed excludes attacks, so that the intruder cannot obtain any information, for instance by comparison of encrypted messages (or their reuse).

For a third major difference, recall that (α, β)-privacy by default does not hide which transaction is taken, but this information can be hidden from the intruder (cf. the comparison with trace equivalence in Section 5.5). It is important to note that the *number* of transactions taken *cannot* be hidden from the intruder in (α, β)-privacy. For instance, this program

```
while (x>0){
  x:=x-1;
  y:=y*x;
}
```

would be fine in classical information flow if x and y are both H (or both L), as all implicit and explicit flows are allowed. However, the number of transactions that can be taken now reveals information about x. One may consider this as a (rather blunt) timing leak. In fact, we cannot directly implement this program as one transaction in (α, β)-privacy. Instead we would have to also model a program counter—as a β-variable that is *not* given to the intruder:

$$
\begin{array}{l}
\mathsf{if}\ (\mathsf{PC} \doteq \mathsf{start} \wedge \mathsf{Xcell} > 0)\ \mathsf{then} \\
\quad \mathsf{Xcell} := \mathsf{Xcell} - 1. \\
\quad \mathsf{Ycell} := \mathsf{Ycell} * \mathsf{Xcell}. \\
\mathsf{else\ if}\ (\mathsf{PC} \doteq \mathsf{start} \wedge \mathsf{Xcell} \doteq 0)\ \mathsf{then} \\
\quad \mathsf{PC} \doteq \mathsf{end}
\end{array}
$$

Now after only two such transactions, the intruder can infer that the initial value of x is greater then 0 and thus violate privacy.

A fourth difference between the approaches concerns declassifications. While in information flow one would declassify a particular value, (α, β)-privacy in general allows one to rather declassify a statement, and declassification is closed under logical deduction. For instance, when we release the result of an election, we do not just declassify the number of votes received by each candidate (that was computed based on the classified votes), but we also release the statement that it is the sum of all accepted votes. Thus, if we have three dishonest voters voting

for candidate A, and candidate A received only three votes, then the intruder can infer—and is allowed to infer—that no honest voter voted for candidate A. Similarly, in a unanimous vote, the intruder also learns the value of every single vote, and that is permitted by the goal.

As a final difference, consider the security lattices that can be used for the different levels in information flow. While we, in (α, β)-privacy, basically consider only two levels (i.e., whether the intruder may know or not), this can be used anyway for a more complex lattice by several separate analyses, one for each security level that the intruder may realistically obtain. There is, however, one aspect that (α, β)-privacy cannot handle at all: how information flow can use lattices for integrity analysis, e.g., that information cannot flow from an untrusted variable into a trusted one. Here, (α, β)-privacy can only offer the formulation of the usual authentication goals like injective agreement that are standard in protocol verification.

5.7 Future Work

Having introduced transition systems, it is natural to consider the automation of our approach. Since it is based on FOL, our formalization is expressive but not decidable, not even semi-decidable, which presents challenges for automation. Still, some fragments of (α, β)-privacy are decidable [MV19] and are, in some sense, equivalent to the classical static equivalence of frames, so there is hope for automation for fragments of dynamic probabilistic (α, β)-privacy too. We also plan to extend our approach to formalize other quantitative aspects of privacy in addition to probabilities, such as costs and timing leaks as in [Di +07; Koc96].

Chapter 6

Conclusion

The work of this thesis is divided into two main parts. We first developed a framework to study vertical composition of stateful protocol. Then, we developed applications of (α, β)-privacy to voting protocols and then leveraged (α, β)-privacy, first to a dynamic approach, and second to a probabilistic approach.

Compositionality results In Chapter 2, we started by defining what we mean by the vertical composition of stateful protocols. Our definition allowed to recast this problem into a parallel composition of stateful protocols that play the role of a channel and of an application protocol. We could then adapt the results from [Hes19] to study the security of our vertical composition. As a first step for applying these results, we extend several definitions and theorems, most notably the underlying typing result that Hess et al. build upon, in order to consider the particularities of our problem, namely that a channel can transmit any messages from an application. We then developed an abstraction for the channel and proved this abstraction sound. This abstraction allows us to make the verification of the channel truly independent of the application. This actually justifies the usual abstraction of payloads by fresh nonces, but shows that it is actually a bit more complicated i.e., one has to consider constants that can be fresh or reused, and known or unknown to the intruder. Our results can be applied to a large class of protocols, given that they respect a number of easy to check syntactic conditions and is, to our knowledge, the first result of vertical compositionality of stateful protocols in the symbolic model not limited to a specific class of protocols.

Voting privacy and receipt-freeness In Chapter 4, we applied (α, β)-privacy to voting protocols, and showed how to model voting privacy goals. We gave proof techniques to prove or disprove that privacy holds, in a particular (α, β)-privacy state. In particular, we showed how this novel approach can express in a rather declarative way a receipt-freeness goal. The idea is that a coerced voter should be able to come up with a consistent story for everything

that could have happened. In addition, we proved a hierarchy between these two properties: receipt-freeness entails voting privacy.

Privacy as reachability In the first part of Chapter 5, we extended (α, β)-privacy with transition systems. This means that we lifted (α, β)-privacy from a static approach to a dynamic one. We defined a transaction-process formalism for distributed systems, where we keep track of several possibilities in each sate until some can be excluded by observation, that can exchange cryptographic messages and that includes privacy variables, can work with long-term mutable states and allows one to specify the consciously released information. Hence, every sate of our transition system is an (α, β)-privacy problem, i.e., expressing privacy as a pure reachability problem that supports a wide variety of privacy goals.

Extension to probabilities In the second part of Chapter 5, we developed a conservative extension of (dynamic) (α, β)-privacy by probabilistic variables. These privacy variables can be sampled from finite domain with a probabilistic distribution. We showed with examples that a probabilistic analysis can benefit many problems. We also defined a notion of extensibility, which says that β does not exclude choices of probabilistic variables. We proved that if (α, β)-privacy holds possibilistically for an extensible pair (α, β), then it also holds probabilistically. Finally, we formalized the relationship between our approach and trace equivalence.

Future work

Algebraic properties A limitation inherent in the work of [Hes19] is that the term model is currently based on a free algebra, in which terms are equal if they are syntactically equal. The question is thus whether it is possible to support algebraic properties such as the properties of exponentiation needed to model Diffie-Hellman-based protocols, which are omnipresent in channel protocol design. The ProVerif tool supports algebraic properties to some extend by a suitable encoding in Horn clauses (which are in the free algebra). A similar transformation may also be possible in our setting. Another possibility identified by [Hes19] would be to work directly with quotient algebras in Isabelle.

NuFMC The worked presented in this thesis has focused on establishing vertical compositionality results. As for now, to apply our results, one would need

to manually model and prove protocols correct. Integrating protocol verification with our results is thus crucial to make them practically available. Though our results are not limited to the protocol formalism we used in Chapter 2, we are working on developing a new hybrid tool based on OFMC [BMV05], that uses set-abstraction techniques to model the stateful aspects of protocols. This tool uses lazy data-types as a simple way of building efficient on-the-fly model-checkers for protocols with very large state-spaces and integrates symbolic techniques and optimizations for modeling a lazy Dolev-Yao intruder. We use set-based abstraction techniques from [Möd10] to handle the stateful aspects of protocols.

Refined privacy goals We have formalized in this work a number of privacy goals, e.g., voting secrecy, receipt-freeness, unlinkability, etc. An interesting question for future research is to extend the scope of domains, such as e-health protocols, and the scope of properties that we can formalize such as stronger flavors of secrecy or coercion-resistance.

Automation for (α, β)-privacy Having introduced transition systems, it is natural to consider the automation of our approach. Since it is based on First-Order Logic, our formalization is expressive but not decidable, and Herbrand Logic is not even semi-decidable, which presents challenges for automation. Still, some fragments of (α, β)-privacy are decidable [MV19] and are, in some sense, equivalent to the classical static equivalence of frames, so there is hope for automation for fragments of dynamic probabilistic (α, β)-privacy too.

Compositionality results for privacy Can we compose privacy properties? In other words, given similar requirements as for confidentiality and authentication goals, can we verify privacy goals with the same compositionality techniques? An interesting question for future research is thus to combine our latest results that recast privacy as a reachability problem with (α, β)-privacy and our compositionality results.

Bibliography

[ABF18] Martín Abadi, Bruno Blanchet, and Cédric Fournet. "The Applied Pi Calculus: Mobile Values, New Names, and Secure Communication". In: *J. ACM* 65.1 (2018), 1:1–1:41.

[ACD15] Myrto Arapinis, Vincent Cheval, and Stéphanie Delaune. "Composing Security Protocols: From Confidentiality to Privacy". In: *ETAPS 2015*. 2015, pp. 324–343.

[Alm+15] Omar Almousa et al. "Typing and Compositionality for Security Protocols: A Generalization to the Geometric Fragment". In: *ESORICS*. 2015.

[Alv+20] Mário S. Alvim et al. *The Science of Quantitative Information Flow*. 2020.

[And+08] Suzana Andova et al. "A framework for compositional verification of security protocols". In: *Inf. Comput.* 206.2-4 (2008), pp. 425–459.

[Ara+17] Myrto Arapinis et al. "Stateful applied pi calculus: Observational equivalence and labelled bisimilarity". In: *J. Log. Algebraic Methods Program.* 89 (2017), pp. 95–149.

[Arm+05] Alessandro Armando et al. "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications". In: *CAV*. 2005.

[Arm+08] Alessandro Armando et al. "Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps". In: *FMSE*. 2008, pp. 1–10.

[BAF08] Bruno Blanchet, Martín Abadi, and Cédric Fournet. "Automated verification of selected equivalences for security protocols". In: *J. Log. Algebraic Methods Program.* 75.1 (2008), pp. 3–51.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate". In: *SP*. 2017.

[BCH10] Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. "Formal Verification of Privacy for RFID Systems". In: *CSF*. 2010, pp. 75–88.

[BCM13] David A. Basin, Cas Cremers, and Simon Meier. "Provably repairing the ISO/IEC 9798 standard for entity authentication". In: *J. Comput. Secur.* 21.6 (2013), pp. 817–846.

[BDP15] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Alfredo Pironti. "Verified Contributive Channel Bindings for Compound Authentication". In: *NDSS*. 2015.

[Bla01] Bruno Blanchet. "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules". In: *CSFW-14*. 2001.

[BMV05] David A. Basin, Sebastian Mödersheim, and Luca Viganò. "OFMC: A symbolic model checker for security protocols". In: *Int. J. Inf. Sec.* (2005), pp. 181–208.

[BS18] Bruno Blanchet and Ben Smyth. "Automated reasoning for equivalences in the applied pi calculus with barriers". In: *J. Comput. Secur.* 26.3 (2018), pp. 367–422.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS*. 2001.

[CC10] Ştefan Ciobâcă and Véronique Cortier. "Protocol Composition for Arbitrary Primitives". In: *CSF 2010*. 2010, pp. 322–336.

[CCD17] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. "A procedure for deciding symbolic equivalence between sets of constraint systems". In: *Inf. Comput.* 255 (2017), pp. 94–125.

[CCM15] Vincent Cheval, Véronique Cortier, and Eric le Morvan. "Secure Refinements of Communication Channels". In: *FSTTCS*. 2015.

[CCW17] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. "Secure Composition of PKIs with Public Key Protocols". In: *CSF*. 2017.

[CD09] Véronique Cortier and Stéphanie Delaune. "Safely Composing Security Protocols". In: *FMSD* (2009).

[Cha+16] Rohit Chadha et al. "Automated Verification of Equivalence Properties of Cryptographic Protocols". In: *ACM Trans. Comput. Log.* 17.4 (2016), 23:1–23:32.

[Che+13] Céline Chevalier et al. "Composition of password-based protocols". In: *Formal Methods Syst. Des.* 43.3 (2013), pp. 369–413.

[CHM01] David Clark, Sebastian Hunt, and Pasquale Malacaria. "Quantitative Analysis of the Leakage of Confidential Data". In: *ENTCS* 59.3 (2001), pp. 238–251.

[CKR18] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. "The DEEPSEC Prover". In: *FloC 2018*. 2018, pp. 28–36.

[Cre+17] Cas Cremers et al. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *CCS*. 2017.

[CRZ07] Véronique Cortier, Michaël Rusinowitch, and Eugen Zalinescu. "Relating two standard notions of secrecy". In: *Log. Methods Comput. Sci.* 3.3 (2007).

[CS11] Véronique Cortier and Ben Smyth. "Attacking and Fixing Helios: An Analysis of Ballot Secrecy". In: *CSF*. 2011, pp. 297–311.

[DH17] Stéphanie Delaune and Lucca Hirschi. "A survey of symbolic methods for establishing equivalence-based properties in cryptographic protocols". In: *JLAMP* 87 (2017), pp. 127–144.

[DHW04] A. Di Pierro, C. Hankin, and H. Wiklicky. "Approximate Non-Interference". In: *J. Comput. Secur.* 12.1 (2004), pp. 37–81.

[Di +07] A. Di Pierro et al. "Tempus Fugit: How to Plug It". In: *JLAP* 72.2 (2007), pp. 173–190.

[DJP12] Naipeng Dong, Hugo Jonker, and Jun Pang. "Formal Analysis of Privacy in an eHealth Protocol". In: *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*. Vol. 7459. Lecture Notes in Computer Science. 2012, pp. 325–342.

[DKR06] Stéphanie Delaune, Steve Kremer, and Mark Ryan. "Coercion-Resistance and Receipt-Freeness in Electronic Voting". In: *CSFW*. 2006.

[DKR09] Stéphanie Delaune, Steve Kremer, and Mark Ryan. "Verifying privacy-type properties of electronic voting protocols". In: *J. Comput. Secur.* 17.4 (2009), pp. 435–487.

[DP-20] DP-3T. *DP-3T – Decentralized Privacy-Preserving Proximity Tracing*. 2020. URL: `https://github.com/DP-3T/documents/blob/master/DP3T%20White%20Paper.pdf`.

[DRS08] Stéphanie Delaune, Mark Ryan, and Ben Smyth. "Automatic Verification of Privacy Properties in the Applied pi Calculus". In: *Trust Management II - Proceedings of IFIPTM 2008: Joint iTrust and PST Conferences on Privacy, Trust Management and Security, June 18-20, 2008, Trondheim, Norway*. Vol. 263. IFIP Advances in Information and Communication Technology. 2008, pp. 263–278.

[Dwo08] Cynthia Dwork. "Differential Privacy: A Survey of Results". In: *TAMC*. LNCS 4978. 2008.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". In: *IEEE Trans. Inf. Theory* 29.2 (1983), pp. 198–207.

[EMM07] Santiago Escobar, Catherine A. Meadows, and José Meseguer. "Equational Cryptographic Reasoning in the Maude-NRL Protocol Analyzer". In: *ENTCS* (2007).

[FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. "A Practical Secret Voting Scheme for Large Scale Elections". In: *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*. Vol. 718. Lecture Notes in Computer Science. 1992, pp. 244–251.

[GM11] Thomas Groß and Sebastian Mödersheim. "Vertical Protocol Composition". In: *CSF*. 2011.

[GM19] Sébastien Gondron and Sebastian Mödersheim. "Formalizing and Proving Privacy Properties of Voting Protocols Using Alpha-Beta Privacy". In: *ESORICS*. LNCS 11735. 2019, pp. 535–555.

[GM21] Sébastien Gondron and Sebastian Mödersheim. "Vertical Composition and Sound Payload Abstraction for Stateful Protocols". In: *CSF*. 2021.

[GM82] Joseph A. Goguen and José Meseguer. "Security Policies and Security Models". In: *IEEE Symposium on Security and Privacy*. 1982, pp. 11–20.

[GMV21] Sébastien Gondron, Sebastian Mödersheim, and Luca Viganò. *Privacy as Reachability*. Tech. rep. 2021.

[Gru08] Damas P. Gruska. "Probabilistic Information Flow Security". In: *Fundam. Inform.* 85 (2008), pp. 173–187.

[GT00] Joshua D. Guttman and F. Javier Thayer. "Protocol Independence through Disjoint Encryption". In: *CSFW*. 2000.

[Gut09] Joshua D. Guttman. "Cryptographic Protocol Composition via the Authentication Tests". In: *FOSSACS*. 2009.

[Gut11] Joshua D. Guttman. "Shapes: Surveying Crypto Protocol Runs". In: *CIS 5*. 2011.

[Gut14] Joshua D. Guttman. "Establishing and preserving protocol security goals". In: *J. Comput. Secur.* (2014).

[Hes+21] Andreas Hess et al. "Performing Security Proofs of Stateful Protocols". In: *CSF*. 2021.

[Hes19] Andreas Viktor Hess. "Typing and Compositionality for Stateful Security Protocols". PhD thesis. 2019.

[HG06] T. Hinrichs and M. Genesereth. *Herbrand Logic*. Tech. rep. Stanford University, 2006.

[HMB18] Andreas V. Hess, Sebastian A. Mödersheim, and Achim D. Brucker. "Stateful Protocol Composition". In: *ESORICS*. 2018.

[HMB20] Andreas Victor Hess, Sebastian Mödersheim, and Achim D. Brucker. "Stateful Protocol Composition and Typing". In: *Arch. Formal Proofs* (2020).

[HT96] Nevin Heintze and J. D. Tygar. "A Model for Secure Protocols and Their Compositions". In: *IEEE Trans. Software Eng.* 22.1 (1996), pp. 16–30.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology*. 1996, pp. 104–113.

[KR05] Steve Kremer and Mark Ryan. "Analysis of an Electronic Voting Protocol in the Applied Pi Calculus". In: *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*. Vol. 3444. Lecture Notes in Computer Science. 2005, pp. 186–200.

[KT11] Ralf Küsters and Max Tuengerthal. "Composition Theorems Without Pre-Established Session Identifiers". In: *CCS*. 2011.

[LLV07] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity". In: *ICDE*. 2007, pp. 106–115.

[LMT10] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. "Time and Probability-Based Information Flow Analysis". In: *IEEE Trans. Software Eng.* 36.5 (2010), pp. 719–734.

[Low02] Gavin Lowe. "Quantifying Information Flow". In: *CSF*. 2002, pp. 18–31.

[Low95] Gavin Lowe. "An Attack on the Needham-Schroeder Public-Key Authentication Protocol". In: *Inf. Process. Lett.* 56.3 (1995), pp. 131–133.

[Mac+07] Ashwin Machanavajjhala et al. "L-diversity: Privacy beyond k-anonymity". In: *ACM Trans. Knowl. Discov. Data* 1 (2007).

[Mei+13] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *CAV*. 2013.

[Mit02] John C. Mitchell. "Multiset Rewriting and Security Protocol Analysis". In: *RTA 2002*. 2002, pp. 19–22.

[Möd10] Sebastian Mödersheim. "Abstraction by set-membership: verifying security protocols and web services with databases". In: *CCS*. 2010, pp. 351–360.

[MSS11] Heiko Mantel, David Sands, and Henning Sudbrock. "Assumptions and Guarantees for Compositional Noninterference". In: *CSF*. 2011, pp. 218–232.

[MV09] Sebastian Mödersheim and Luca Viganò. "Secure Pseudonymous Channels". In: *ESORICS*. 2009.

[MV14] Sebastian Mödersheim and Luca Viganò. “Sufficient conditions for vertical composition of security protocols”. In: *ASIA CCS*. 2014.

[MV19] Sebastian Mödersheim and Luca Viganò. “Alpha-Beta Privacy”. In: *ACM Trans. Priv. Secur.* 22.1 (2019), 7:1–7:35.

[Net21] Nets. *NEM ID*. 2021. URL: `https://www.nemid.nu/dk-en/get_started/code_token/` (visited on 06/22/2021).

[OSK03] M. Ohkubo, K. Suzuki, and S. Kinoshita. “Cryptographic approach to "privacy-friendly" tags”. In: *RFID Privacy Workshop* (2003).

[SB18] Christoph Sprenger and David A. Basin. “Refining security protocols”. In: *J. Comput. Secur.* (2018).

[Sel75] Steve Selvin. “On the Monty Hall problem (letter to the editor)”. In: *The American Statistician* 29.3 (1975).

[Smi09] Geoffrey Smith. “On the Foundations of Quantitative Information Flow”. In: *FoSSaCS*. LNCS 5504. 2009, pp. 288–302.

[Swe02] Latanya Sweeney. “k-Anonymity: A Model for Protecting Privacy”. In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10.5 (2002), pp. 557–570.

[TJ11] Henk C. A. van Tilborg and Sushil Jajodia, eds. *Encyclopedia of Cryptography and Security, 2nd Ed.* 2011.

[Vau20] Serge Vaudenay. *Analysis of DP3T*. Cryptology ePrint Archive, Report 2020/399. 2020.

[YCW17] Jiannan Yang, Yongzhi Cao, and Hanpin Wang. “Differential privacy in probabilistic systems”. In: *Inf. Comput.* 254 (2017), pp. 84–104.