Vertical Composition and Sound Payload Abstraction for Stateful Protocols (Extended Version)

Sébastien Gondron and Sebastian Mödersheim
DTU Compute, Denmark, Kgs. Lyngby
spcg,samo@dtu.dk

June 23, 2021

Abstract

This paper deals with a problem that arises in vertical composition of protocols, i.e., when a channel protocol is used to encrypt and transport arbitrary data from an application protocol that uses the channel. Our work proves that we can verify that the channel protocol ensures its security goals *independent* of a particular application. More in detail, we build a general paradigm to express vertical composition of an application protocol and a channel protocol, and we give a transformation of the channel protocol where the application payload messages are replaced by abstract constants in a particular way that is feasible for standard automated verification tools. We prove that this transformation is sound for a large class of channel and application protocols. The requirements that channel and application have to satisfy for the vertical composition are all of an easy-to-check syntactic nature.

Keywords— security protocols, formal methods and verification, vertical composition, stateful protocols

1 Introduction

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 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.

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 1: Running Example

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 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 confidential 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 paper 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 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 paper 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, we introduce the framework to model stateful protocols. In §3, we describe our paradigm for vertical composition, and we extend a typing result to support an abstract payload type. In §4, 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. We relate our work to others and conclude in §5. Appendix A gives the proof of the main theorems. Appendix B gives the full extension of the typing results. Appendix C show how to apply the vertical composability definition to our running example. Appendix D gives a battery of examples to illustrate the scope of our results. Finally, Appendix E shows how our results can be used to study channel bindings.

2 Preliminaries

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

2.1 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 $\sigma(t) = \sigma(t')$.

2.2 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 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*:*

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

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

($Axiom$) 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. $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ implies $T \subseteq \{t_1, \ldots, t_n\}$, and
4. $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ implies $\mathsf{Ana}(\sigma(f(t_1, \ldots, t_n))) = (\sigma(K), \sigma(T))$.

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 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 paper, for simplicity, we stick with this free term algebra model.

2.3 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 $\mathcal{V}$:

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

As syntactic sugar, we may write $t \not\doteq t'$ and $t \mathrel{\dot\notin} t'$ in lieu of $\forall \bar{x}.\ t \not\doteq t'$ and $\forall \bar{x}.\ t \mathrel{\dot\notin} t'$ when $\bar{x}$ is the empty sequence. We may also write $t \to t'$ for $\mathsf{insert}(t, t')$ and $t \leftarrow t'$ for $t \mathrel{\dot\in} t'.\mathsf{delete}(t, t')$. 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. *In Figure 2, we formalize the* App *from Figure 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 paper we restrict ourselves to well-formed transaction rules.

2.4 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:

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

$\mathcal{I}$ 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 paper we consider only well-formed constraints without further mention.

2.5 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 3 and Example 4.

3 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

[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.

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.

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 §3.1. Note that §3.2 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 §3.3 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 §4) is a sound abstraction of the payload messages of the application so that the channel can also be verified independent of the application.

3.1 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 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, among 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 Appendix B.

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.

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 1. *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 com-

patible 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 (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 3 (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 $\forall t, t' \in SMP(M) \backslash \mathcal{V}$. $(\exists \sigma. \sigma(t) = \sigma(t')) \rightarrow \Gamma(t) \geq \Gamma(t') \vee \Gamma(t) \leq \Gamma(t')$. 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 1 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 ((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.

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: Example of a login protocol

3.2 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 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 4 (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 5 (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 Appendix B.

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 6 (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 $\star$.*

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

Definition 7 (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 $(\mathcal{P}_1, \mathcal{P}_2, Sec)$ 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 $s \in Sec$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in Sec$,*

[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.

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 3 (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* $\mathcal{P}_1 \parallel \mathcal{P}_2$.

3.3 Channels and Applications

As our second contribution, 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 8 (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 3. *We formalize the running example from Figure 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,* 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 Appendix D.*

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 9 (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 4 (Unilaterally authenticated secure channel)**.** *We now model the channel protocol from Figure 1 in our framework as a unilaterally authenticated secure channel,*

[3] This declassification step is in principle forbidden by Definition 8. 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 $\forall P \in \mathsf{Alias}_{\mathsf{Dis}}, C \in \mathsf{Agent}|_{\mathsf{Dis}}.\star\colon \mathsf{inv}(P) \rightarrow \mathsf{outbox}(C, C)$.

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

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

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

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

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

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

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

Figure 3: Example for an unilaterally authenticated pseudonymous secure channel

similar to what TLS without client authentication would establish. We consider the same sets of agents that we used in Example 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 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 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 Appendix D, 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*

$\mathsf{sessKeys}$ *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 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 Appendix D.*

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

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

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

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

Figure 4: Idealization of the channel protocol from Figure 3

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 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 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 5: Example for a channel with perfect forward secrecy.

Example 5 (Perfect Forward secrecy)**.** *Figure 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 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 3 we can derive:

Theorem 4 (Vertical Composition (with unabstracted payload))**.** *Given a channel protocol* Ch *and an application protocol* App *w.r.t. a ground set Sec of terms where the only shared sets are the* inbox *and* outbox *sets*[5] *s.t.* $(\mathsf{Ch}, \mathsf{App}, Sec)$ *is parallel composable. If both* $\mathsf{App} \parallel \mathsf{Ch}^\star$ *and* $\mathsf{App}^\star \parallel \mathsf{Ch}$ *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 $\mathsf{App} \parallel \mathsf{Ch}^\star$ is now independent of the concrete channel, however the verification of $\mathsf{App}^\star \parallel \mathsf{Ch}$ 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.

4 Abstracting the Payload

As the third and core contribution, 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 $\mathsf{Ch} \parallel \mathsf{App}^\star$ 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 $\mathsf{Ch}^\sharp$ such that each transformation is sound (if there is an

[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 $\mathsf{compromized}$ would have to be a shared set (i.e., operations labeled $\star$).

[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.

attack, then so there is after the transformation). If we manage to verify $\mathsf{Ch}^\sharp$, then we have also verified $\mathsf{Ch} \parallel \mathsf{App}^\star$ 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 $\mathsf{Ch}^\sharp$ 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.

4.1 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 $\mathfrak{G} \subseteq \mathit{Sec}$ be an infinite set of constants disjoint from $\mathit{GSMP}_{\mathsf{Ch}}$ and from $\mathit{GSMP}_{\mathsf{App}}$. 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.

4.2 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 that $\mathsf{Ch}^\sharp$ has an attack if $\mathsf{Ch} \parallel \mathsf{App}^\star$ has, i.e., this abstraction is sound.

Definition 10 (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 ($\mathsf{Ch}^\sharp_{\mathrm{new}}$) 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 3:

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

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

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

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

$$\frac{\mathsf{Ch}_{5a}^\sharp\colon}{\begin{array}{l}\star\colon G \mathrel{\dot\in} \mathsf{opened}.\\ \star\colon \xrightarrow{G}\end{array}}$$

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

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

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

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

Figure 6: Abstraction for our example channel Ch from 3

Example 6 (Abstraction of the channel from Example 4)**.** *In Figure 6, we give the set of rules of* $\mathsf{Ch}^\sharp$ *transformed from the set of rules given in Figure 3 following Definition 10 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 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 $\mathsf{Ch}^\sharp_{3a}$ contains the positive check $(\star\colon G \dot\in \mathsf{opened})$ at the beginning, whereas the rule $\mathsf{Ch}^\sharp_{3b}$ contains $(\star\colon G \dot\in \mathsf{closed})$. 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 $\mathsf{Ch}^\sharp_{5a}$, 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 $\mathsf{Ch}^\sharp_{5b}$, we add the positive check $(\star\colon G \dot\in \mathsf{closed})$. 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 $\mathsf{Ch}^\sharp_{new}$ 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 11 (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 4. 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 5. *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 Appendix A 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 1. *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 11). This means that one has to prove $(\mathsf{Ch}, \mathsf{App}, Sec)$ is parallel composable (Definition 7) and $\mathsf{Ch} \parallel \mathsf{App}$ is type-flaw resistant (Definition 3). 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 1. We show how to apply the results to the protocols from our main examples in Appendix C.

5 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 paper 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 paper 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 paper 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.

References

[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.

[And+08] Suzana Andova et al. "A framework for compositional verification of security protocols". In: *Inf. Comput.* 206.2-4 (2008), pp. 425–459.

[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.

[BBK17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate". In: *SP*. 2017.

[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.

[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.

[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).

[Che+13] Céline Chevalier et al. "Composition of password-based protocols". In: *Formal Methods Syst. Des.* 43.3 (2013), pp. 369–413.

[Cre+17] Cas Cremers et al. "A Comprehensive Symbolic Analysis of TLS 1.3". In: *CCS*. 2017.

[EMM07] Santiago Escobar, Catherine A. Meadows, and José Meseguer. "Equational Cryptographic Reasoning in the Maude-NRL Protocol Analyzer". In: *ENTCS* (2007).

[GM11] Thomas Groß and Sebastian Mödersheim. "Vertical Protocol Composition". In: *CSF*. 2011.

[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.

[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.

[KT11] Ralf Küsters and Max Tuengerthal. "Composition Theorems Without Pre-Established Session Identifiers". In: *CCS*. 2011.

[Low95] Gavin Lowe. "An Attack on the Needham-Schroeder Public-Key Authentication Protocol". In: *Inf. Process. Lett.* 56.3 (1995), pp. 131–133.

[Mei+13] Simon Meier et al. "The TAMARIN Prover for the Symbolic Analysis of Security Protocols". In: *CAV*. 2013.

[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.

[Net21] Nets. *NEM ID*. 2021. URL: `https://www.nemid.nu/dk-en/get_started/code_token/` (visited on 06/22/2021).

[SB18] Christoph Sprenger and David A. Basin. "Refining security protocols". In: *J. Comput. Secur.* (2018).

A 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 4 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 12 (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. $\mathsf{Ch}^{\mathsf{App}}_{i,t} = \mathsf{Ch}_i[X \mapsto t]$ 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 $\mathsf{Ch}^{\mathsf{App}}_{i,t}$, 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 11, 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 6. *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 12 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 4.1. This function leaves unchanged the terms labeled Ch or *Pub*:

Definition 13 (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 14 (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 1. *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 13, $g \in \mathfrak{G}$. 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 14 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 $g \in declassified_{\mathcal{DY}}(tr'(0,n))$. Thus, $g \in declassified_{\mathcal{DY}}(tr'(0,n)) \cap \mathfrak{G}$.

Let $M = declassified_{\mathcal{DY}}(tr(1,n))$ and $M' = declassified_{\mathcal{DY}}(tr'(0,n))$, 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., $t \in declassified_{\mathcal{DY}}(tr(1,n)) \cap GSMP^\bullet_{\mathsf{App}}$. 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 $declassified_{\mathcal{DY}}(tr'(0,n)) \subseteq g(declassified_{\mathcal{DY}}(tr(1,n)))$.

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. *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 $g \to \mathsf{closed}$ occurs in the trace) and $g \notin \mathit{declassified}_{\mathcal{DY}}(tr'(0,n+1))$ (so $g \to \mathsf{opened}$ 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 1, 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 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. $f(t_1, \ldots, t_p) \in \mathit{GSMP}^{\bullet}_{\mathsf{App}}$: we have $g(f(t_1, \ldots, t_p)) \in \mathfrak{G}$, and $t_i \in \mathit{GSMP}_{\mathsf{App}}$ for $1 \le i \le p$. For each $1 \le i \le p$, 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 $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1)) \cap \mathit{GSMP}^{\bullet}_{\mathsf{App}}$ by Lemma 1. Thus, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1))$ for all $1 \le i \le p$ (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 $f(t_1, \ldots, t_p) \in \mathit{GSMP}^{\bullet}_{\mathsf{App}}$, again by Lemma 1, 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. $f(t_1, \ldots, t_p) \notin \mathit{GSMP}^{\bullet}_{\mathsf{App}}$: 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

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

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 1, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1, n+1)) \cap GSMP^\bullet_{\mathsf{App}}$. 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 1, $g(t) \in \mathit{declassified}_{\mathcal{DY}}(tr'(0, n+1)) \cap \mathfrak{G}$, 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 5. *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 6, 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 14.

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 14:

- 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 $\mathcal{DY}(M(tr'(0, n))) \cap (g(\mathit{Sec} \setminus \mathit{declassified}_{\mathcal{DY}}(tr(1, n))) \cup \{\mathsf{attack}_{\mathsf{Ch}}\}) \neq \emptyset$

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 14 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 14, 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 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, 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 13, 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 1. *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 5 and Theorem 3. □

B Extension of the typing results

B.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 paper).

Lemma 3. *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 4. *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 1. *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 3. 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 3. By Lemma 4, 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.

□

B.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 5 ([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 1, 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 6 ([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.

B.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 paper, 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 paper a variant of declassification that is closed under Dolev-Yao deduction.

B.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 paper 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 15 (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 7 (Update of [Hes19] (Lemma 5.10))**.** *If $\mathcal{A}$ is a constraint of $\mathcal{P}$, then every $t \in GSMP_{\mathcal{A}}$ is homogeneous.*

An important next step is that the intruder never needs to construct heterogeneous terms. For that we first define homogeneous intruder deduction:

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

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

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

Lemma 8 (Update of [Hes19] (Lemma 5.12))**.** *Given a finite set of messages $M \subset GSMP_{\mathcal{P}}$ and a term $t \in GSMP_{\mathcal{P}}$, 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 9 (Update of [Hes19] (Lemma 5.13))**.** *Given a constraint $\mathcal{A}$ of $\mathcal{P}$ and a well-typed model $\mathcal{I}$ such that $\forall s \in Sec \setminus declassified_{\mathcal{DY}}(\mathcal{A}, \mathcal{I})$. $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \not\vdash_{\text{hom}} s$ 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 $t \notin Sec \setminus declassified_{\mathcal{DY}}(\mathcal{A}, \mathcal{I})$ and if $t \in GSMP(\mathcal{A}|_i)$ 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 10 (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 $\mathsf{ik}(\mathcal{I}(\mathcal{A})) \vdash t \in GSMP(\mathcal{A}|_i)$, we have $\mathsf{ik}(\mathcal{I}(\mathcal{A}|_i)) \vdash t$ where i is a protocol-specific label.*

The next step is:

Lemma 11 (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 10 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 10, 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.

C 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 and 3) as summarized in the end of Section 4. Thus, following Definition 11, 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 7). 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 $s \in \mathit{Sec}$ and $s' \sqsubseteq s$, either $\emptyset \vdash s'$ or $s' \in \mathit{Sec}$,

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 3, 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 $\Gamma(\{X\}) = \{\mathfrak{p}\}$:

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

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 among 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, $(P \dot{\notin} \mathsf{taken})$, $(P \dot{\notin} \mathsf{alias}(C))$ and $(X \dot{\notin} \mathsf{secCh}(A,B))$, 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 $GSMP_{\mathsf{Ch}\|\mathsf{App}^\star}$ 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 11.

$GSMP_{\mathsf{App}}$ 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 $GSMP_{\mathsf{Ch}^\sharp}$ 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 1 that $\frac{\mathsf{App}}{\mathsf{Ch}}$ is secure.

D 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 7),
- a channel providing authentication without secrecy (Figures 8 to 10),
- two different channel mechanisms to guarantee replay protections that both expose the same interface, and (Figures 11 to 14)

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

D.1 Key-exchange with certificate

Consider first the variant of Ch in Figure 7 where an agent is authenticated with a certificate. Let CAuthority be a set of the public constants representing the honest certification authorities. Let 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 4.

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

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

Figure 7: Example for the key-exchange part of a channel with certificates

D.2 Authenticated channel without secrecy

We continue with an example of a channel that provides unilateral authentication but without secrecy in Figure 8. We consider a similar setting than the one in Figure 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, 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 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 9 compared to

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 8: Example for an unilateraly authenticated channel without secrecy

the one in Figure 4 is that messages transported on the channel are declassified.

For concision, we keep in the abstraction in Figure 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.

$\underline{\mathsf{Ch}_3^\star \colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.}$

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

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

$\star\colon\ X \mathrel{\dot\in} \mathsf{authCh}(A,B).$
$\star\colon\ X \rightarrow \mathsf{inbox}(A,B)$

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

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

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

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

Figure 9: Idealization of the channel protocol from Figure 8

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

D.3 Channel with replay protection

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 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 10: Abstraction for our example channel Ch from Figure 8

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\star: \ \xrightarrow{X}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 11: Example for an unilaterally authenticated pseudonymous channel with replay protection

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 rules Ch_4, Ch_7 and Ch_8; they are applicable when a message that looks like a legitimate message from honest A to

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

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

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

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

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

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

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

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

Figure 12: Idealization of the channel protocol from Figure 11

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 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 13. Note that we once again removed the redundant rules and the ones that do not perform an action anymore.

D.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

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

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

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

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

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

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

$\star$: $G, N \to \mathsf{secCh}(A, B).$

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

$\mathsf{Ch}_{4a,b}^\sharp$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\mathsf{Ch}_{8a,b}^\sharp$: $\forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.$

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

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

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

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

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

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

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

Figure 13: Abstraction for our example channel Ch from Figure 11

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 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 14: Second example for an unilaterally authenticated pseudonymous channel with replay protection

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 en-

cryption 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 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.

$$\underline{\mathsf{App}_1 : \forall C \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.}$$
$$\mathsf{App}: \; P \dot{\notin} \mathsf{taken}.$$
$$\mathsf{App}: \; P \rightarrow \mathsf{taken}.$$
$$\mathsf{App}: \; P \rightarrow \mathsf{alias}(C)$$

$$\underline{\mathsf{App}_2 : \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.}$$
$$\star: \; \xrightarrow{\mathsf{inv}(P)}$$

$$\underline{\mathsf{App}_3 : \forall S \in \mathsf{Agent}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}, \mathsf{new}N.}$$
$$\mathsf{App}: \; P \dot{\in} \mathsf{alias}(C).$$
$$\mathsf{App}: \; N \rightarrow \mathsf{loginCounter}(P, S).$$
$$\star: \; f_2(\mathsf{secret}(C, S)) \rightarrow \mathsf{outbox}(P, S)$$

$$\underline{\mathsf{App}_4 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.}$$
$$\star: \; f_2(\mathsf{secret}(C, S)) \leftarrow \mathsf{inbox}(P, S).$$
$$\mathsf{App}: \; P \dot{\in} \mathsf{alias}(C).$$
$$\mathsf{App}: \; N \leftarrow \mathsf{loginCounter}(P, S)$$

$$\underline{\mathsf{App}_4 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.}$$
$$\star: \; f_2(\mathsf{secret}(C, S)) \leftarrow \mathsf{inbox}(P, S).$$
$$\mathsf{App}: \; P \dot{\notin} \mathsf{alias}(C).$$
$$\mathsf{App}: \; \xleftarrow{\mathsf{attack}_{\mathsf{App}}}$$

Figure 15: Example of a login protocol without replay protection

Now with the additional guarantee for replay protection from the channel, we consider in Figure 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 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.

E Channel Bindings

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

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

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

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

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

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

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

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

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

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

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

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

Figure 16: Example of an application (to deploy over a unilaterally authenticated secure channel) that has a renegotiation-style flaw.

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 16 is meant to be run over the unilaterally authenticated channel from Figure 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 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 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 $\mathsf{critCmd}(N) \notin \mathsf{begin}(A, S)$, 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, 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 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 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 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 17: Binding the alias P to a real name with an asymmetric credential.

We conclude with the remark that the application in Figure 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.