Vertical Composition and Sound Payload Abstraction for Stateful Protocols

Sébastien Gondron*, Sebastian Mödersheim†
DTU Compute
Denmark, Kgs. Lyngby
{*spcg,†samo}@dtu.dk

2021 IEEE 34th Computer Security Foundations Symposium (CSF) | 978-1-7281-7607-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSF51468.2021.00038

***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.**

***Index Terms*—security protocols, formal methods and verification, vertical composition, stateful protocols**

I. 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 $f_{(\dots)}$ 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 service where each

Channel protocol Ch:
Setup:

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

Transport:

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

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

Login protocol App:

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

Fig. 1. Running Example

user has a personal key-card to look up the response for a given challenge N. The first factor, a password, we just omit for simplicity.

Most existing works on protocol composition have concentrated on *parallel* composition, i.e., when protocols run independently on the same network only sharing an infrastructure of fixed long-term keys [15, 12, 7]. 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. [11, 6] allow for interaction between the protocols that are being composed, albeit specialized to a particular form of interaction. [17] 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 [19] 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.

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 [8], 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 §II, we introduce the framework to model stateful protocols. In §III, we describe our paradigm for vertical composition, and we extend a typing result to support an abstract payload type. In §IV, 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 §V. Appendix A gives the proof of the main theorems. The full extension of the typing results and further examples can be found in the extended version [10].

II. Preliminaries

Most of the content of this section is adapted from [19].

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

B. 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 [19]). *We define $\vdash$ as the least relation that includes the knowledge, and is closed under composition with public functions and under analysis with* Ana*:*

$$\frac{}{M \vdash t}\ \begin{array}{l}(Axiom),\\ t \in M\end{array} \qquad \frac{M \vdash t_1 \ldots M \vdash t_n}{M \vdash f(t_1, \ldots, t_n)}\ \begin{array}{l}(Compose),\\ f \in \Sigma^n_{pub}\end{array}$$

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

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 [19] to hold:

1) $\mathsf{Ana}(t) = (K, T)$ implies that K is finite, $fv(K) \subseteq fv(t)$,
2) $\mathsf{Ana}(x) = (\emptyset, \emptyset)$ for variables $x \in \mathcal{V}$,
3) $\mathsf{Ana}(f(t_1, \ldots, t_n)) = (K, T)$ implies $T \subseteq \{t_1, \ldots, t_n\}$,
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 t 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 s.t.* f_1 *that the intruder can open without knowing any keys, e.g.,* $\mathsf{Ana}(f_1(t, t')) = (\emptyset, \{t, t'\})$*. 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.

C. 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: $R_i = \forall x_1 \in T_1, \ldots, x_n \in T_n.$ $\mathsf{new}\ y_1, \ldots, y_m.S$ where 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 $\mathcal{V}$:

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

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 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 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 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* $f_1(N, S)$ *to a set* $\mathsf{outbox}(S, P)$ *for being sent on a secure channel to* P*. Here* f_1 *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* $P \mathrel{\dot\in} \mathsf{alias}(A)$*. 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 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 transactions.

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

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

We extend $trms(\cdot)$ and $setops(\cdot)$ to transaction strands, rules and protocols as expected. The straightforward definition of the semantics, i.e., that an interpretation $\mathcal{I}$ is a *model* of

$\mathcal{A}$, written $\mathcal{I} \models \mathcal{A}$, is found in the extended version [10]. 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.

E. 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 S, written $dual(S)$ means "swapping" the direction of the sent and received messages of S: $dual(\mathsf{send}(t).S) = \mathsf{receive}(t).dual(S)$, $dual(\mathsf{receive}(t).S) = \mathsf{send}(t).dual(S)$ and otherwise $dual(\mathfrak{s}.S) = \mathfrak{s}.dual(S)$ for any other step $\mathfrak{s}$. The transition $\mathcal{A} \Rightarrow \mathcal{A}.dual(\alpha(\sigma(S)))$ is possible if the following conditions are met:

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

Note that by this 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 [1, 13], 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.

III. Stateful Vertical Composition

The compositionality result of Hess et al. [19, 18] allows for the *parallel* composition of stateful protocols. The protocols being composed may share sets. An example would be a server that maintains a database and runs several protocols that access and modify this database.[1] After specifying an appropriate interface how these protocols may access and modify the database, one can verify each protocol individually with respect to this interface and obtain the security of the composed system.

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

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 [19, 18] relies on a typing result, and this typing result is not powerful enough for our paradigm of vertical composition. The extension is in fact our first main contribution in §III-A. Note that §III-B comes mainly from [19, 18] 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 §III-C 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 §IV) is a sound abstraction of the payload messages of the application so that the channel can also be verified independent of the application.

A. Typed Model and Payloads

As already mentioned, the typing result of [19, 18] is not general enough for our purposes: since we want to define a channel protocol independent of the application that uses

[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.

the channel, we would like the messages that the channel transports to be of an abstract type $\mathfrak{p}$ (*payload*) that can, during composition, be instantiated by the concrete message types of the application protocol.

This requires, however, a substantial extension of the typing system and the typing results, 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 it is transporting. This is, however, incompatible with the typing result from [19, 18], 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 [19, 18] forbids.

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 [19, 18] 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 [19] and mark our extensions; the proof of the results under the extensions is given in the extended version [10].

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

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

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

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

For the payload extension, we define a partial order on types, formalizing that the abstract payload is a generalization of the types in $\mathfrak{T}_{\mathfrak{p}}$:

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

We say that two types τ and τ' are *compatible* when they can be compared with the partial order. We say a substitution σ is *well-typed* iff $\Gamma(x) \geq \Gamma(\sigma(x))$ for all $x \in \mathcal{V}$. This is a generalization of [19] 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 [19] 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 [19] are minor: we use our updated notion of well-typed, and we use the notion of compatible types instead of the same type. We give the definitions as an almost verbatim quote without pointing out these minor differences each time.

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

Definition 2 (Sub-message patterns [19])**.** *The sub-message patterns $\mathit{SMP}(M)$ for a set of messages M is defined as the least set satisfying the following rules:*

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

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

- *$\mathit{trms}(\mathcal{A}) \cup \mathit{setops}(\mathcal{A})$ 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 $\bar{x}$*

- *For all $\forall \bar{x}.t \mathrel{\dot{\notin}} t'$ 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 [19]. 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 [19])). *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 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.

B. Parallel Compositionality

We now review and adapt the *parallel composition* result from [19]. 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). 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 [19]). *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 [19]: 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 [19])). *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 $declassified_{\mathcal{DY}}(\mathcal{A}, \mathcal{I}) \equiv \mathcal{DY}(\{t \mid \star\colon \xleftarrow{\;t\;}$ occurs in $\mathcal{I}(\mathcal{A})\})$.*

This modification requires the update of several definitions and proofs from [19]. We provide the details of this extension in the extended version [10].

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 ([19])). *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 [19]). *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 $s' \in Sec$,*
3) *for all $l\colon (t, s)$, $l'\colon (t', s') \in labeledsetops(\mathcal{P}_1 \parallel \mathcal{P}_2)$, if (t, s) and (t', s') are unifiable then $l = l'$,*

[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.

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_1 : \forall C \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}|_{\mathsf{Hon}}.} \\
\hline
\mathsf{App}: & P \mathrel{\dot{\notin}} \mathsf{taken}. \\
\mathsf{App}: & P \to \mathsf{taken}. \\
\mathsf{App}: & P \to \mathsf{alias}(C)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_2 : \forall P \in \mathsf{Alias}|_{\mathsf{Dis}}.} \\
\hline
\star: & \xrightarrow{\mathsf{inv}(P)}
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_3 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, \mathsf{new}\ N.} \\
\hline
\mathsf{App}: & N \to \mathsf{sent}(S, P). \\
\star: & f_1(N, S) \to \mathsf{outbox}(S, P)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_4 : \forall S \in \mathsf{Agent}, P \in \mathsf{Alias}|_{\mathsf{Hon}}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.} \\
\hline
\star: & f_1(N, S) \leftarrow \mathsf{inbox}(S, P). \\
\mathsf{App}: & P \mathrel{\dot{\in}} \mathsf{alias}(C). \\
\star: & f_2(\mathsf{mac}(\mathsf{secret}(C, S), N)) \to \mathsf{outbox}(P, S)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_5 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, C \in \mathsf{Agent}.} \\
\hline
\star: & f_2(\mathsf{mac}(\mathsf{secret}(C, S), N)) \leftarrow \mathsf{inbox}(P, S). \\
\mathsf{App}: & N \leftarrow \mathsf{sent}(S, P)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{App}_6 : \forall S \in \mathsf{Agent}|_{\mathsf{Hon}}, P \in \mathsf{Alias}, C \in \mathsf{Agent}|_{\mathsf{Hon}}.} \\
\hline
\star: & f_2(\mathsf{mac}(\mathsf{secret}(C, S), N)) \leftarrow \mathsf{inbox}(P, S). \\
\mathsf{App}: & N \mathrel{\dot{\in}} \mathsf{sent}(S, P). \\
\mathsf{App}: & P \mathrel{\dot{\notin}} \mathsf{alias}(C). \\
\mathsf{App}: & \xrightarrow{\mathsf{attack}_{\mathsf{App}}}
\end{array}$$

Fig. 2. Example of a login protocol

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 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 $l\colon (\forall \bar{x}. t \notin s)$ occurs in $\mathcal{P}\}$.

Composition of secure, parallel composable protocols is secure:

Theorem 3 (Parallel Composition [19])**.** *If $(\mathcal{P}_1, \mathcal{P}_2, Sec)$ 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 in the typed model, then all goals of $\mathcal{P}_1$ hold in $\mathcal{P}_1 \parallel \mathcal{P}_2$.*

C. 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 [19], 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 will be 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 $\mathfrak{T}_\mathsf{p}$ 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 $\mathfrak{T}_\mathsf{p}$ 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.*

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. Note that N is of type* Nonce. *Then S uses the channel to P by inserting $f_1(N, S)$ into its outbox for P, where f_1 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*

[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 $\forall P \in \mathsf{Alias}_{\mathsf{Dis}}, C \in \mathsf{Agent}|_{\mathsf{Dis}}.\star\colon \mathsf{inv}(P) \to \mathsf{outbox}(C, C)$.

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_1 : \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \text{new } K.} \\ \hline
\mathsf{Ch}: & K \rightarrow \mathsf{sessKeys}(P,B). \\
\mathsf{Ch}: & \xrightarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_2 : \forall P \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.} \\ \hline
\mathsf{Ch}: & \xleftarrow{\mathsf{crypt}(\mathsf{pk}(B),\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))} . \\
\mathsf{Ch}: & K \rightarrow \mathsf{sessKeys}(B,P)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_3 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.} \\ \hline
\star: & X \leftarrow \mathsf{outbox}(A,B). \\
\mathsf{Ch}: & K \mathrel{\dot{\in}} \mathsf{sessKeys}(A,B). \\
\star: & X \rightarrow \mathsf{secCh}(A,B). \\
\mathsf{Ch}: & \xrightarrow{\mathsf{scrypt}(K,f_{pseudo}(A,B,X))}
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_4 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.} \\ \hline
\mathsf{Ch}: & \xleftarrow{\mathsf{scrypt}(K,f_{pseudo}(A,B,X))} . \\
\mathsf{Ch}: & K \mathrel{\dot{\in}} \mathsf{sessKeys}(B,A). \\
\star: & X \mathrel{\dot{\in}} \mathsf{secCh}(A,B). \\
\star: & X \rightarrow \mathsf{inbox}(A,B)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_5 : \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.} \\ \hline
\star: & X \leftarrow \mathsf{outbox}(A,B). \\
\star: & \xrightarrow{X}
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_6 : \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.} \\ \hline
\star: & \xleftarrow{X} . \\
\star: & X \rightarrow \mathsf{inbox}(A,B)
\end{array}$$

$$\begin{array}{rl}
\multicolumn{2}{l}{\mathsf{Ch}_7 : \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.} \\ \hline
\mathsf{Ch}: & \xleftarrow{\mathsf{scrypt}(K,f_{pseudo}(A,B,X))} . \\
\mathsf{Ch}: & K \mathrel{\dot{\in}} \mathsf{sessKeys}(A,B). \\
\star: & X \mathrel{\dot{\notin}} \mathsf{secCh}(A,B). \\
\mathsf{Ch}: & \xleftarrow{\mathsf{attack}_{\mathsf{Ch}}}
\end{array}$$

Fig. 3. Example for an unilaterally authenticated pseudonymous secure channel

the owner of P. The client computes a MAC of the challenge N with a pre-shared $secret(C,S)$ with the server. Here mac *is a public function and* secret *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, $f_2(\mathsf{mac}(\mathsf{secret}(C,S),N))$, 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 the extended version [10].*

The payload types of this application are $\mathfrak{T}_{\mathfrak{p}} = \{f_1(\mathsf{Nonce},\mathsf{Agent}), f_2(\mathsf{mac}(\mathsf{secret}(\mathsf{Agent},\mathsf{Agent}),\mathsf{Nonce}))\}$.

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, 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 the extended version [10] 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).

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 has a 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*.*

$$\begin{array}{ll}
\underline{\mathsf{Ch}_3^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.} & \underline{\mathsf{Ch}_4^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Hon}}, B \in \mathsf{Names}|_{\mathsf{Hon}}.} \\
\quad \star\colon \quad X \leftarrow \mathsf{outbox}(A,B). & \quad \star\colon \quad X \mathrel{\dot\in} \mathsf{secCh}(A,B). \\
\quad \star\colon \quad X \rightarrow \mathsf{secCh}(A,B). & \quad \star\colon \quad X \rightarrow \mathsf{inbox}(A,B) \\
\underline{\mathsf{Ch}_5^\star\colon \forall A \in \mathsf{Names}, B \in \mathsf{Names}|_{\mathsf{Dis}}.} & \underline{\mathsf{Ch}_6^\star\colon \forall A \in \mathsf{Names}|_{\mathsf{Dis}}, B \in \mathsf{Names}.} \\
\quad \star\colon \quad X \leftarrow \mathsf{outbox}(A,B). & \quad \star\colon \quad \xleftarrow{\quad X \quad} . \\
\quad \star\colon \quad \xrightarrow{\quad X \quad} & \quad \star\colon \quad X \rightarrow \mathsf{inbox}(A,B)
\end{array}$$

Fig. 4. Idealization of the channel protocol from Figure 3

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 sn example of a channel offering replay protection, see the extended version [10].)*

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.*

Example 5 (Perfect Forward secrecy)**.** *Figure 5 shows a modification of our running example that also provides perfect forward secrecy of 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*

[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 $\star$).

$$\begin{array}{ll}
\multicolumn{2}{l}{\underline{\mathsf{Ch}_{0a}: \forall B \in \mathsf{Agent}.}} \\
\mathsf{Ch}: & B \rightarrow \mathsf{compromized} \\
\mathsf{Ch}: & \xrightarrow{\mathsf{inv}(\mathsf{pk}(B))}
\end{array}
\qquad
\begin{array}{ll}
\multicolumn{2}{l}{\underline{\mathsf{Ch}_{0b}: \forall P \in \mathsf{Alias}_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ PK.}} \\
\mathsf{Ch}: & B \mathrel{\dot{\notin}} \mathsf{compromized} \\
\mathsf{Ch}: & PK \rightarrow \mathsf{tmpK}(B,P) \\
\mathsf{Ch}: & \xrightarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)),f_{PK}(B,P,PK))}
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\underline{\mathsf{Ch}_1: \forall P \in \mathsf{Alias}|_{\mathsf{Hon}}, B \in \mathsf{Agent}, \mathsf{new}\ K.}} \\
\mathsf{Ch}: & B \mathrel{\dot{\notin}} \mathsf{compromized} \\
\mathsf{Ch}: & \xleftarrow{\mathsf{sign}(\mathsf{inv}(\mathsf{pk}(B)),f_{PK}(B,P,PK))} \\
\mathsf{Ch}: & K \rightarrow \mathsf{sessKeys}(P,B). \\
\mathsf{Ch}: & \xrightarrow{\mathsf{crypt}(PK,\mathsf{sign}(\mathsf{inv}(P),f_{newSess}(P,B,K)))}
\end{array}
\qquad
\begin{array}{ll}
\multicolumn{2}{l}{\underline{\mathsf{Ch}_2: \forall A \in \mathsf{Alias}, B \in \mathsf{Agent}|_{\mathsf{Hon}}.}} \\
\mathsf{Ch}: & B \mathrel{\dot{\notin}} \mathsf{compromized} \\
\mathsf{Ch}: & \xleftarrow{\mathsf{crypt}(PK,\mathsf{sign}(\mathsf{inv}(P)),f_{newSess}(P,B,K))}\ . \\
\mathsf{Ch}: & PK \leftarrow \mathsf{tmpK}(B,P) \\
\mathsf{Ch}: & K \rightarrow \mathsf{sessKeys}(B,P)
\end{array}$$

Fig. 5. Example for a channel with perfect forward secrecy.

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.

IV. Abstracting the Payload

As our 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 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 to 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.

A. Abstract Constants

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

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

B. Translation from $\mathsf{Ch} \parallel \mathsf{App}^\star$ to $\mathsf{Ch}^\sharp$

We now explain formally the transformation of Ch into the protocol $\mathsf{Ch}^\sharp$. As explained, in the rules of $\mathsf{Ch}^\sharp$, the payload messages of type $\mathfrak{p}$ have been replaced by variables of type $\mathfrak{a}$, thus allowing us to verify the channel without considering the concrete terms from the application. Furthermore, since after this abstraction we do not need the interface with the application anymore, we drop the steps with outbox and inbox sets. We prove later in this section 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 ($\mathsf{Ch}^\sharp_{\mathsf{new}}$) is that any "new" abstract constant is first inserted in closed, and they are moved to opened and reveled to the intruder whenever they represent a payload that is declassified. Note that this setup handles both

[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.

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

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

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

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

$$\frac{\mathsf{Ch}^\sharp_{5a}\colon}{\begin{array}{rl}\star\colon & G \mathrel{\dot{\in}} \mathsf{opened}.\\ \star\colon & \xrightarrow{G}\end{array}}$$

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

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

$$\frac{\mathsf{Ch}^\sharp_{\mathsf{new}}\colon \text{new } G.}{\star\colon\ G \twoheadrightarrow \mathsf{closed}}$$

$$\frac{\mathsf{Ch}^\sharp_{5b}\colon}{\begin{array}{rl}\star\colon & G \twoheadleftarrow \mathsf{closed}\\ \star\colon & G \twoheadrightarrow \mathsf{opened}.\\ \star\colon & \xrightarrow{G}\end{array}}$$

Fig. 6. $\mathsf{Ch}^\sharp$ for our example channel Ch from 3

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:

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 $\mathsf{Ch}^\sharp_1$ and $\mathsf{Ch}^\sharp_2$ 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 ($\star\colon G \mathrel{\dot{\in}} \mathsf{opened}$) at the beginning whereas the rule $\mathsf{Ch}^\sharp_{3b}$ contains ($\star\colon G \mathrel{\dot{\in}} \mathsf{closed}$). The further transformations are similar for the two rules since there is no declassification step for the payload. The step containing the set operation for outbox *is dropped. The payload message inserted into the* secCh *set is replaced by the variable G of type $\mathfrak{a}$, as is the payload message in the transmitted message. The transformations for the rule Ch_4 are very similar. It needs to be split into two rules, the* inbox *step is dropped and the payload messages X are replaced by a variable G of type $\mathfrak{a}$.*

The rule Ch_5 also has to be split into two rules. Since the payload is declassified upon transmission to the intruder, the transformations are different for the two rules. In $\mathsf{Ch}^\sharp_{5a}$, we add the positive check ($\star\colon G \mathrel{\dot{\in}} \mathsf{opened}$). 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 ($\star\colon G \mathrel{\dot{\in}} \mathsf{closed}$). We also remove the step with* outbox*. Since, the remaining step, after replacing the payload with the variable of type $\mathfrak{a}$, is the declassification of that variable, and since that G is in* closed*, we need to replace the previously added positive check and the declassification step by ($\star\colon G \twoheadleftarrow \mathsf{closed}.\star\colon G \twoheadrightarrow \mathsf{opened}.\star\colon \xrightarrow{G}$). 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 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 replace 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 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 in full detail how to apply the results to our main example in the extended version [10].

V. 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 [15, 12, 7]. 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 [11] for vertical composition: a handshake protocol can generate secure keys that are then used to encrypt traffic of an application protocol; similarly, [6] 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 [24], 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 redone after changing the application. Indeed, the work [5] 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 [5] 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 [13]. Second, [5] 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, [5] 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 [22, 23], 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 [3], AVISPA [2], Maude-NPA [9], CPSA[14] or Tamarin [21]. We have verified for instance our running example in Isabelle with PSPSP [16].

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 [4, 20]. 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

[1] Omar Almousa et al. “Typing and Compositionality for Security Protocols: A Generalization to the Geometric Fragment”. In: *ESORICS*. 2015.

[2] Alessandro Armando et al. “The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications”. In: *CAV*. 2005.

[3] Bruno Blanchet. “An Efficient Cryptographic Protocol Verifier Based on Prolog Rules”. In: *CSFW-14*. 2001.

[4] Ran Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *FOCS*. 2001.

[5] Vincent Cheval, Véronique Cortier, and Eric le Morvan. “Secure Refinements of Communication Channels”. In: *FSTTCS*. 2015.

[6] Vincent Cheval, Véronique Cortier, and Bogdan Warinschi. “Secure Composition of PKIs with Public Key Protocols”. In: *CSF*. 2017.

[7] Véronique Cortier and Stéphanie Delaune. “Safely Composing Security Protocols”. In: *FMSD* (2009).

[8] Cas Cremers et al. “A Comprehensive Symbolic Analysis of TLS 1.3”. In: *CCS*. 2017.

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

[10] Sébastien Gondron and Sebastian Mödersheim. *Vertical Composition and Sound Payload Abstraction for Stateful Protocols (Extended Version)*. Tech. rep. 2021. URL: https://www.imm.dtu.dk/~samo/AbstractPayload.pdf.

[11] Thomas Groß and Sebastian Mödersheim. “Vertical Protocol Composition”. In: *CSF*. 2011.

[12] Joshua D. Guttman. “Cryptographic Protocol Composition via the Authentication Tests”. In: *FOSSACS*. 2009.

[13] Joshua D. Guttman. “Establishing and preserving protocol security goals”. In: *J. Comput. Secur.* (2014).

[14] Joshua D. Guttman. “Shapes: Surveying Crypto Protocol Runs”. In: *CIS 5*. 2011.

[15] Joshua D. Guttman and F. Javier Thayer. “Protocol Independence through Disjoint Encryption”. In: *CSFW*. 2000.

[16] Andreas Hess et al. “Performing Security Proofs of Stateful Protocols”. In: *CSF*. 2021.

[17] Andreas V. Hess, Sebastian A. Mödersheim, and Achim D. Brucker. “Stateful Protocol Composition”. In: *ESORICS*. 2018.

[18] Andreas Victor Hess, Sebastian Mödersheim, and Achim D. Brucker. “Stateful Protocol Composition and Typing”. In: *Arch. Formal Proofs* (2020).

[19] Andreas Viktor Hess. “Typing and Compositionality for Stateful Security Protocols”. PhD thesis. 2019.

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

[21] Simon Meier et al. “The TAMARIN Prover for the Symbolic Analysis of Security Protocols”. In: *CAV*. 2013.

[22] Sebastian Mödersheim and Luca Viganò. “Secure Pseudonymous Channels”. In: *ESORICS*. 2009.

[23] Sebastian Mödersheim and Luca Viganò. “Sufficient conditions for vertical composition of security protocols”. In: *ASIA CCS*. 2014.

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

Appendix

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 $M(\mathcal{A}(1, n)) = \{m \mid \xleftarrow{m}$ 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 IV 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 Ch $\|$ $\mathsf{App}^\star$, we want to reduce the problem of solving an intruder constraint representing a protocol execution of Ch $\|$ $\mathsf{App}^\star$ 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 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, it is split into a rule for every* $t \in \mathit{GSMP}_{\mathsf{App}}$ *s.t.* $\mathsf{Ch}_{i,t}^{\mathsf{App}} = \mathsf{Ch}_i[X \mapsto t]$ *where the payload is instantiated with a ground subterm from the application. All others with a set operation for a set of the set families* inbox *or* outbox *are dropped.*

Note that if Ch_i is a well-formed rule, then $\mathsf{Ch}_{i,t}^{\mathsf{App}}$, for every $t \in \mathit{GSMP}_{\mathsf{App}}$, are also well-formed rules. Indeed, instead of retrieving a variable from an outbox set, we instantiate it with a ground term from $\mathit{GSMP}_{\mathsf{App}}$. 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* Ch $\|$ $\mathsf{App}^\star$ *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* Ch $\|$ $\mathsf{App}^\star$ *then there is an attack against* $\mathsf{Ch}^{\mathsf{App}}$*.*

Proof. Let $\mathcal{A}$ be a constraint of Ch $\|$ $\mathsf{App}^\star$ 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,
- 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 the 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 $H(n)$ 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,
(d) $\mathcal{I} \models \mathcal{A}'(1, n)$

For $n = 0$, it is obvious. Let us now assume that $H(n)$ 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 $\mathcal{I}(X) \in \mathit{GSMP}_{\mathsf{App}}$, 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 $\mathit{GSMP}_{\mathsf{App}}$ 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 Ch $\|$ $\mathsf{App}^\star$, there is an attack against $\mathsf{Ch}^{\mathsf{App}}$. $\square$

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 $\mathit{GSMP}^{\bullet}_{\mathsf{App}} = \mathit{GSMP}_{\mathsf{App}} \setminus \{t \mid \emptyset \vdash t\}$ — 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 IV-A. 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 $g(f(t_1,\dots,t_n)) = f(g(t_1),\dots,g(t_n))$ whenever $f(t_1,\dots,t_n) \notin GSMP^\bullet_{\mathsf{App}}$. When $g(t) \in \mathfrak{G}$, we may sometimes write simply g_t to denote the element of $\mathfrak{G}$ that t maps to.*

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 $g \in$ 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 $g \in$ 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, $g \in declassified_{\mathcal{DY}}(tr'(0,n)) \cap \mathfrak{G}$.

For the other direction, 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. 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., $declassified_{\mathcal{DY}}(tr'(0,n)) \cap \mathfrak{G} = g(declassified_{\mathcal{DY}}(tr(1,n))) \cap GSMP^\bullet_{\mathsf{App}}$. $\square$

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

Note that, if $g \in declassified_{\mathcal{DY}}(tr'(1,n+1)) \cap \mathfrak{G}$, then by Lemma 1, there exists a $t \in declassified_{\mathcal{DY}}(tr(0,n+1)) \cap GSMP^\bullet_{\mathsf{App}}$ 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 $M^\star \vdash t$.

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

1) $f(t_1,\dots,t_n) \in GSMP^\bullet_{\mathsf{App}}$: We have $g(f(t_1,\dots,t_n)) \in \mathfrak{G}$, and $t_i \in GSMP_{\mathsf{App}}$ for $1 \le i \le n$. For each $1 \le i \le n$, 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, $g(t_i) \in declassified_{\mathcal{DY}}(tr'(0,n+1)) \cap \mathfrak{G}$ since $tr'(0,n+1)$ does not leak, and thus $t_i \in declassified_{\mathcal{DY}}(tr(1,n+1)) \cap GSMP^\bullet_{\mathsf{App}}$ by Lemma 1. Thus, $t_i \in declassified_{\mathcal{DY}}(tr(1,n+1))$ for all $1 \le i \le n$ (including public t_i). Thus, by $\mathcal{DY}$-closure also $f(t_1,\dots,t_n) \in declassified_{\mathcal{DY}}(tr(1,n+1))$, and since

also $f(t_1,\ldots,t_n) \in \mathit{GSMP}^{\bullet}_{\mathsf{App}}$, again by Lemma 1, we have $g(f(t_1,\ldots,t_n)) \in \mathit{declassified}_{\mathcal{DY}}(tr'(1,n+1)) \cap \mathfrak{G}$ and thus $g(f(t_1,\ldots,t_n)) \in M^\star$ 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) $f(t_1,\ldots,t_n) \notin \mathit{GSMP}^{\bullet}_{\mathsf{App}}$: 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

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

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) $t_0 \in \mathit{GSMP}^{\bullet}_{\mathsf{App}}$: we have that $g(t_0) \in \mathfrak{G}$, and $t, t_1,\ldots,t_n \in \mathit{GSMP}_{\mathsf{App}}$. For each $0 \le i \le p$, we have either $t_i \in \{t \mid \emptyset \vdash t\}$, then $g(t_i) = t_i$, otherwise $t_i \in \mathit{GSMP}^{\bullet}_{\mathsf{App}}$ 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, $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1)) \cap \mathit{GSMP}^{\bullet}_{\mathsf{App}}$. Thus $t_i \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1))$ for all $0 \le i \le p$ (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 $t \in \mathit{GSMP}_{\mathsf{App}}$, we have $t \in \mathit{declassified}_{\mathcal{DY}}(tr(1,n+1)) \cap \mathit{GSMP}^{\star}_{\mathsf{App}}$ 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) $t_0 \notin \mathit{GSMP}^{\bullet}_{\mathsf{App}}$: 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 $\mathit{GSMP}^{\bullet}_{\mathsf{App}}$ either. Thus, $g(t_0) = g(f(t_1,\ldots,t_n)) = f(g(t_1),\ldots,g(t_n)) = f(t_1,\ldots,t_p,g(t_{p+1}),\ldots,g(t_n))$. By induction, we have that $M^\star \vdash g(t_0)$ and $M^\star \vdash g(t_i) = t_i$ for $1 \le i \le p$. 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.
trace $tr(1,n)$ and the $n+1$ blocks of steps of the translated trace $tr'(0,n)$ defined in Definition 14:

We proceed now by induction, where the induction hypothesis $H(n)$ is concerned with the first n blocks of the original

- either $tr'(0,n)$ is a valid trace of $\mathsf{Ch}^\sharp$, and $g(\mathcal{DY}(M(tr(1,n)))) \subseteq \mathcal{DY}(M(tr'(0,n)))$,
- 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 $n \ge 0$, 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 $H(n+1)$ 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 $H(n+1)$ 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 $H(n+1)$.

By induction, we proved the theorem. □

Finally, the composition of vertical composable and secure application and channel protocols is secure. The proof of Corollary 1 is a direct consequence of Theorems 3 and 5.