

# Sufficient Conditions for Vertical Composition of Security Protocols (Extended Version)<sup>\*</sup>

Sebastian Mödersheim<sup>1</sup>    Luca Viganò<sup>2</sup>

<sup>1</sup> DTU Informatics, Lyngby, Denmark, [samo@imm.dtu.dk](mailto:samo@imm.dtu.dk)

<sup>2</sup> Dipartimento di Informatica, Università di Verona, Italy, [luca.vigano@univr.it](mailto:luca.vigano@univr.it)

IMM-Technical Report-2011-18

**Abstract.** Vertical composition of security protocols means that an application protocol (e.g., a banking service) runs over a channel established by another protocol (e.g., a secure channel provided by TLS). This naturally gives rise to a compositionality question: given a secure protocol  $P_1$  that provides a certain kind of channel as a goal and another secure protocol  $P_2$  that assumes this kind of channel, can we then derive that their vertical composition  $P_2[P_1]$  is secure? It is, however, well known that protocol composition can lead to attacks even when the individual protocols are all secure in isolation. We distinguish two kinds of aspects for this failure of composition: (i) a logical aspect, a mismatch between the behavior of a channel as an assumption and as a goal, and (ii) a static aspect, an interaction between the message formats, i.e., when message parts of  $P_1$  may be confused with message parts of  $P_2$ .

In a previous work, we concentrated on the logical aspect, simply assuming that the static aspect can somehow be solved similar to existing disjoint encryption notions. In this paper, we solve this intricate open problem by formalizing seven easy-to-check static conditions that support a large class of channels and applications, and that we prove to be sufficient for vertical protocol composition.

## 1 Introduction

It is preferable to verify the security of a protocol like TLS in isolation: independent of what other protocols may later be used on the same network, and independent of what application protocols one later wants to use the TLS channels for. One should prefer to verify small or medium-size protocols in isolation because:

1. The composition of several protocols immediately leads to very complex systems that are infeasible for formal verification, be it manual or automated.

---

<sup>\*</sup> The work presented in this paper was partially supported by the FP7-ICT-2009-5 Project no. 257876, “SPaCIoS: Secure Provision and Consumption in the Internet of Services”.

2. One should not have to repeat a security proof of all existing protocols whenever a new protocol is added to a system.
3. One should strive for general and reusable results, e.g., that TLS provides a secure channel no matter what application it is used for. Similarly, the verification of an application protocol that relies on a secure channel should be completely independent of how (i.e., by which other protocol) that channel is actually realized.

In general, however, the composition of secure protocols may lead to attacks even when the individual protocols are all secure in isolation. Compositional reasoning thus aims at identifying conditions for security protocols that are sufficient to prove a statement of the form: whenever secure protocols (that satisfy the conditions) are composed, then also their composition is secure.

There are several results for the *parallel* composition of security protocols (e.g., [10,11]), i.e., when two or more protocols are used (independently) on the same network. The general idea is here that the message formats of the involved protocols need to be sufficiently different, so that message parts of one protocol cannot be accidentally mistaken for message parts of another protocol, to avoid any risk that confusions could be exploited by an intruder. Note that, in general, such conditions are sufficient for compositionality<sup>1</sup>, but not necessary: some protocols with confusable protocol formats may still be sound to compose when confusions cannot be exploited by an intruder.

Another line of work concerns *sequential* protocol composition (e.g., [9,11,16]), where the result of one protocol, e.g., a shared session key, becomes input to a second, subsequent secure channel protocol. One may interpret TLS as an example of sequential composition where the TLS handshake protocol establishes a pair of symmetric keys and the TLS transport protocol uses them to encrypt messages of an application protocol. A disadvantage of this view is that we must consider the agglomeration of TLS transport with the application protocol, while we would rather like to have a clear-cut dividing line between channel protocol and application protocol, as is common in the layered Internet.

For such a view of layered protocols we will use the term *vertical* protocol composition, in contrast to sequential and parallel composition that we refer to as *horizontal* compositions. Maurer et al. [22] (but see also the more recent [20,21]) have initiated an idea to capture very different means of communication, based on cryptography, protocols, and non-technical means (e.g., trust relationships or humans who know each other) with a notion of *channels* that may build on each other. For instance, a banking service may run over a secure channel that is provided by another protocol such as TLS. Moreover, if *A* and *B* have authentic channels with each other (we distinguish the direction here), they can use, e.g., Diffie-Hellman to establish secure channels with each other; in fact, for Diffie-Hellman, it does not matter how authenticity is achieved, or what the established secure channels are later used for.

Although Maurer et al.'s first paper [22] did not have a formal definition of channels (and the later papers [20,21] have notions that are not suitable

---

<sup>1</sup> In this paper, we use the terms *compositionality* and *composability* interchangeably.

for protocol verification, as they lack a trace-based semantics), this work has inspired our own paper on secure pseudonymous channels [29]. There, we defined for protocol verification a notion of *channels as assumptions* and a notion of *channels as goals*. This naturally gives rise to a compositionality question: given a secure protocol  $P_1$  that provides a certain kind of channel as a goal (e.g., TLS) and another secure protocol  $P_2$  that assumes this kind of channel (e.g., a banking service), can we then derive that their vertical composition  $P_2[P_1]$  (e.g., a banking service over TLS) is secure?

We distinguish two aspects for the potential failure of such a composition:

1. *Logical aspect*. A mismatch between the behavior of a channel as an assumption and as a goal.<sup>2</sup>
2. *Static aspect*. An interaction between the message formats, i.e., when message parts of  $P_1$  could be confused with message parts of  $P_2$ .

In [29], we concentrated on the logical aspect, simply assuming that the static aspect can somehow be solved similar to existing disjoint encryption notions.

*Contributions* This open problem of how to deal with the static aspect of vertical protocol composition turned out to be intricate for two main reasons.

First, in contrast to all horizontal composition types, the vertical composition has to deal with messages that are composed from the channel protocol  $P_1$  and the application protocol  $P_2$ , because the payload messages of  $P_2$  are *embedded* into message-templates of  $P_1$  that are used for transferring messages with the desired properties. Second, we have that the channel protocol is *parameterized* over a payload message.<sup>3</sup> We want to be able to use standard existing verification methods to verify  $P_1$ , especially independent of payload messages of a particular  $P_2$ . This is in fact why we call this problem the static aspect: we want to see this independent of the dynamic behavior of  $P_2$  using a form of abstract interpretation for the payload.

The contributions of this paper are therefore:

- The definition of a precise interface for the abstract payload in a “pure”  $P_1$  that is suitable for all standard protocol verification methods. This further gives rise to the notion of *static vertical composition*  $P_1^{P_2}$ , where “statically”

---

<sup>2</sup> For instance, standard authentication goals characterize almost exactly our notion of authentic channels as assumption, but are slightly too weak in one aspect: the intruder can send on an authentic channel any message  $m$  that he knows under his real name (obviously, he cannot send any message  $m$  that he does not know), but standard authentication goals do not explicitly require that a message that the intruder authenticated is actually known to him—and, in fact, many people may argue that this is more a question of secrecy. So, we had to require a slightly stronger authentication goal to prove the compositionality.

<sup>3</sup> As we describe in more detail below, the fact that the payload is used as a placeholder for data from an arbitrary protocol makes this a complex problem that cannot be solved simply by requiring the two protocols to be disjoint (which is the typical solution for other kinds of compositionality).

all possible payloads of a protocol  $P_2$  are inserted into  $P_1$  and this serves as an interface to the results of [29].

- We give a set of seven syntactic conditions on the protocols  $P_1$  and  $P_2$  that are easy to check statically. In a nutshell, they require the disjointness of the message formats of  $P_1$  and  $P_2$ , and that the payloads of  $P_2$  are embedded into  $P_1$  under a unique context to define a sharp borderline. These conditions and the other minor conditions are satisfied by a large class of protocols in practice.
- We show that the seven conditions are sufficient for static vertical composition, i.e., if protocols  $P_1$  and  $P_2$  are secure in isolation (which is established using any classical method) and they satisfy the seven conditions, then also  $P_1^{P_2} \parallel P_2$  is secure, where  $\parallel$  denotes parallel composition. That in turn is sufficient for the result in [29] to infer that  $P_2[P_1]$  is secure.
- We formally show that we can also support negative conditions (that are useful for advanced protocols and goals) in the application protocol (not in the channel protocol, however, due to abstract interpretation).
- Finally, we discuss how to extend the result to channel protocols that support more than one channel type and to an arbitrary number of message transmissions, and discuss this for a TLS-based example. These results are left informal though, since the general vertical protocol composition framework of [29] needs to be extended to this end.

In Appendix B, we provide a proof of concept of our conditions by considering a concrete example of vertical protocol composition.

*Related Work* Like the previous results in horizontal protocol composition (see [1,12,14,15] in addition to the works already cited above), our result requires disjoint message formats of the different protocols involved to avoid confusions. Vertical protocol composition, however, makes a difference in that the several layers of a protocol stack can be verified independently, even though messages of the composed protocols are themselves composed from the different layers. For instance, in contrast to [9], we can consider an application protocol completely independent from a transport layer (such as TLS).

The work most similar to ours is [13], which also considers vertical protocol composition. The difference is that [13] supports only one particular kind of channel protocol, namely one that establishes a pair of symmetric keys (one for each communication direction) and then encrypts all messages of the application protocol with the key for the respective direction. In contrast, our results in the present paper are compatible with many channel types (for which the logical connection between channels as assumptions and as goals has to be proved like in [29]) and the transmission over the channel is not limited to symmetric encryption but may be any realization of the desired channel type. Despite being less general in this respect, [13] allows for arbitrary stacks where the same protocol may occur several times, which is here excluded due to disjointness.

Vertical protocol composition is conceptually close to the view of many cryptographers such as [20,21,22] and the Universal Composability (UC) framework [7]. The original UC has however very restrictive assumptions that forbid its

application to many practical compositions, e.g., that different protocols use the same key-material. Recent extensions and modifications of UC have improved the situation drastically [18]. A comparison is difficult here because the UC-works are rooted in the cryptographic world based on the indistinguishability of an ideal and real system, while our approach is based on trace-based protocol semantics and properties (and treats cryptography as black boxes). It is therefore immediate to apply our results with the established protocol verification approaches and the sufficient conditions that we require are both easy to check and satisfied by many protocols that are used in practice.

*Organization* After some preliminaries useful for the formal modeling of protocols (Section 2), in Section A.2 we review the lazy intruder, a constraint reduction technique that we use here for proving compositionality results in a convenient way. In Section 3, we formalize the notion of vertical protocol composition based on the use of channels as assumption and goals. In Section 4, we give the static conditions and our main result that they are sufficient for vertical protocol composition (the proof can be found in Appendix A). In Section 5, we discuss extensions of our compositionality result, and we then conclude in Section 6. In Appendix B, we consider a concrete example of vertical protocol composition.

## 2 Preliminaries: protocol messages and transition system

### 2.1 Messages

Following the line of black-box cryptography models, we employ a term algebra to model the messages that participants exchange. Let  $\Sigma$  be countable signature and  $\mathcal{V}$  be a countable set of variable symbols disjoint from  $\Sigma$ . The signature is partitioned into the set  $\Sigma^0$  of constants and the set of  $\Sigma_p$  of “public” operations, where we will sometimes write  $\Sigma_p^n$  to stress the arity  $n$  of an operation. We use standard notions about terms such as *ground* (without variables), *atomic*, *subterm* (denoted  $s \sqsubseteq t$ ), *substitution* (denoted  $\sigma$  or  $\tau$ ), *set of most general unifiers* (denoted  $mgu(\cdot)$ ). We write  $\mathcal{T}_\Sigma$  to denote the set of ground terms and  $\mathcal{T}_\Sigma(\mathcal{V})$  to denote all terms.

The constants represent agents, keys, nonces, and the like. The function symbols of  $\Sigma_p$  represent operations on messages that every agent can perform. In this paper, we use the following function symbols:

- $\{m\}_k$  represents the asymmetric encryption of message  $m$  with public key  $k$ .
- $\{m\}_k$  represents the symmetric encryption of message  $m$  with symmetric key  $k$ ; we assume that this primitive includes also integrity protection such as a MAC.
- $[m_1, \dots, m_n]_n$ , for every  $n \geq 2$ , represents the concatenation of  $n$  messages  $m_1, \dots, m_n$ . We use this family of operators to abstract from the details of structuring messages in the implementation.

We also use a set of meta-symbols  $\Sigma_m$  that we refer to as *mappings*. We use them to specify mappings that do not necessarily correspond to operations that agents can perform on messages. For instance,  $\text{pubk}(s)$  and  $\text{privk}(s)$  represent the public and private keys resulting from a seed  $s$ . Moreover,  $\text{pk}(A)$  may represent the seed for the public key of agent  $A$  to model a fixed public-key infrastructure. Most importantly, we use the mapping  $\text{payload}(A, B)$  to denote an *abstract* payload message that agent  $A$  wants to send to agent  $B$  (we will make precise the details of payload messages below).

Formally, these mappings are injective functions on  $\Sigma^0$  (rather than function symbols of the term algebra). As a consequence, the expression  $\text{payload}(A, B)$  represents a constant of  $\Sigma^0$ , and is thus regarded as atomic.

## 2.2 Transition system (ASLan)

For concreteness, we use here the *AVANTSSAR Specification Language ASLan* [4,3] but all our results carry over to other protocol formalisms such as Strands, Applied Pi calculus, and so on. ASLan is (i) expressive enough that many high-level languages (e.g., BPMN or Alice-and-Bob-style languages such as the one we will consider in the following for channels and composition) can be translated to it and (ii) amenable to formal analysis (e.g., with OFMC [30] and the other backends of the AVANTSSAR Platform [3]).

ASLan provides the user with an expressive language for specifying security protocols and their properties, based on set rewriting. At its core, ASLan describes a *transition system*, where states are sets of *facts* (i.e., ground terms) separated by dots (“.”). These facts model the state of honest agents, the knowledge of the intruder, communication channels, and facts used in formulating goals. Transitions are specified as *rewriting rules* over sets of facts. We give here one example of an ASLan transition rule, pointing to the references for more details. Consider the following message exchange (that is part of the protocol  $P_1$  that we will consider in Fig. 1 below)

$$A \rightarrow B : \{ \{ [p, A, B, \text{payload}(A, B)]_4 \}_{\text{privk}(\text{pk}(A))} \}_{\text{pubk}(\text{pk}(B))}$$

in which  $A$  first signs with its private key a payload, along with information on sender and receiver that is needed to achieve a secure channel (cf. the discussion about the logical aspect in the introduction and see [29] for more details on why the receiver’s name must indeed be signed as well). The tag  $p$  signals that this concatenation contains the payload transmission.  $A$  then encrypts the message with  $B$ ’s public key.

This message exchange is formalized by two ASLan rules, one for the sender and one for the receiver. One way to model the sender’s transition is as follows:

$$\begin{aligned} \text{state}_{\mathcal{A}}(A, \text{step1}, \text{SID}, B) &\Rightarrow \\ \text{state}_{\mathcal{A}}(A, \text{step2}, \text{SID}, B) &\cdot \text{iknows}(\{ \{ [p, A, B, \text{payload}(A, B)]_4 \}_{\text{privk}(\text{pk}(A))} \}_{\text{pubk}(\text{pk}(B))}) \end{aligned}$$

where  $\text{state}_{\mathcal{A}}(\dots)$  formalizes the local state of an honest agent in role  $\mathcal{A}$ . Here we have chosen to model this state as consisting of the agent’s name  $A$ , its step

number in the protocol execution, a session identifier  $SID$ , and the name of the intended communication partner  $B$ . Note that  $A$ ,  $B$  and  $SID$  are here variables that allow for matching against arbitrary concrete facts. In contrast,  $step1$  is a constant, i.e., this rule can only be applied to an agent that is currently in this stage of the protocol execution. On the right-hand side of the rule, there is the updated state of the honest agent and a message that the agent sends out. Since we assume that the intruder can read all messages that are sent on insecure channels, we immediately add this message to the intruder knowledge formalized by the fact  $iknows(\cdot)$ . Note that the local state of an honest agent does not necessarily carry all the knowledge of the agent (like  $payload(A, B)$ ) but it is sufficient that it contains all those variables on which the terms depend that the agent is supposed to send and receive.

It is standard to define what the intruder can deduce (e.g., encryption and decryption with known keys) by rules on  $iknows(\cdot)$  facts to obtain a Dolev-Yao-style model. We usually also allow that the intruder may completely control several “compromised” agents (including knowing their long-term secrets). We use the predicate  $dishonest(\cdot)$  that holds true for every agent under the intruder’s control (from the initial state on), and the predicate  $honest(\cdot)$  to identify the honest agents.

We describe the goals of a protocol by *attack states*, i.e., states that violate the goals, which are in turn described by *attack rules*: a state at which the attack rule can fire is thus an attack state. For instance, we can formulate a *secrecy goal* as follows. We add the fact  $secret(M, \{A, B\})$  to the right-hand side of an honest agent rule, whenever a message  $M$  is supposed to be a secret between  $A$  and  $B$ , and then give the following attack rule, which expresses that, whenever the fact  $secret(M, \{A, B\})$  holds for two honest agents  $A$  and  $B$ , and the intruder has learned the message, then we can derive the fact **attack**:

$$secret(M, \{A, B\}).iknows(M).honest(A).honest(B) \Rightarrow \mathbf{attack} \quad (1)$$

### 3 Channels and Composition

The most common type of protocol composition is running two protocols in parallel over the same network, which is easy to define for many protocol formalisms. For instance, in a strand notation, we simply consider the union of the strands of the two protocols.

**Definition 1 (Parallel Composition  $P_1 \parallel P_2$  and Composability).** *Let  $P_1 \parallel P_2$  denote the protocol that results from the parallel composition of two protocols  $P_1$  and  $P_2$ .*

*We say that  $P_1$  and  $P_2$  are composable in parallel if the following holds: if  $P_1$  and  $P_2$  are secure in isolation, then also  $P_1 \parallel P_2$  is secure.*

The main idea to ensure parallel compositionality is that messages of the composed protocols should have sufficiently different formats so that no message part of one protocol can be mistaken for one of another. For simple Alice-and-Bob-style protocols, this is already sufficient, but in general more complex situations

$$\begin{array}{c}
\frac{A \rightarrow B : \{\{\{\mathbf{p}, A, B, \text{payload}(A, B)\}_4\}_{\text{privk}(\text{pk}(A))}\}_{\text{pubk}(\text{pk}(B))}}{A \bullet \rightarrow \bullet B : \text{payload}(A, B)} \quad \frac{C \bullet \rightarrow \bullet D : [N, M]_2 \quad D \rightarrow C : [N, C, D]_3}{C \bullet \rightarrow \bullet D : M} \\
\\
\frac{A \rightarrow B : \{\{\{\mathbf{p}, A, B, m_{A,B}\}_4\}_{\text{privk}(\text{pk}(A))}\}_{\text{pubk}(\text{pk}(B))} \quad C \rightarrow D : \{\{\{\mathbf{p}, C, D, [N, M]_2\}_4\}_{\text{privk}(\text{pk}(C))}\}_{\text{pubk}(\text{pk}(D))}}{A \bullet \rightarrow \bullet B : m_{A,B} \quad C \bullet \rightarrow \bullet D : M} \quad \frac{D \rightarrow C : [N, C, D]_3}{C \bullet \rightarrow \bullet D : M}
\end{array}$$

**Fig. 1.** Abstract channel protocol  $P_1$  (top left) and application protocol  $P_2$  (top right), their static vertical composition  $P_1^{P_2}$  (bottom left) and their vertical composition  $P_2[P_1]$  (bottom right).

may occur. For instance, if a web-service maintains a database of transactions that it was involved in, and several of the composed protocols involve reading or writing in this database, then this can result in a “side-channel” that may break compositionality.

As a first step towards compositionality also for protocols that use databases or other forms of (mutable) long-term knowledge, we introduce the notion of *execution independence*, which excludes that the protocols to be composed can mutually enable or disable each other’s transitions.

**Definition 2 (Execution independence of two protocols).** *We say that  $P_1$  and  $P_2$  are execution independent if they are formulated over disjoint sets of facts, except  $\text{iknows}(\cdot)$  and  $\text{attack}$ .*

For example, two protocols that only write into a database are execution independent, and so are two that read and write disjoint kinds of entries. However, if  $P_1$  writes a certain kind of entry and the presence of such an entry is a condition for a transition in  $P_2$ , then these two protocols are no longer execution independent. Note that compositionality of  $P_1$  and  $P_2$  may still be possible in such a case, but our result holds only for execution-independent protocols.

### 3.1 Channels as Assumptions, Channels as Goals

We use here the notion of channels that we introduced in [29], inspired by Maurer and Schmid’s bullet calculus [22]. We may use channels both as protocol assumptions (i.e., when a protocol relies on channels with particular properties for the transmission of some of its messages) and as protocol goals (i.e., a protocol’s objective is the establishment of a particular kind of channel).

Considering channels as assumptions allows us to enhance the standard insecure communication medium with security guarantees for message transmission.

We can express this, e.g., in an Alice-and-Bob-style notation, where a secure end-point of a channel is marked by a bullet, as follows:<sup>4</sup>

- $A \rightarrow B : M$  represents an *insecure channel* from  $A$  to  $B$ , meaning that the channel is under the complete control of the intruder.
- $A \bullet \rightarrow B : M$  represents an *authentic channel* from  $A$  to  $B$ , meaning that  $B$  can rely on the fact that  $A$  has sent the message  $M$ .
- $A \rightarrow \bullet B : M$  represents a *confidential* (or *secret*) *channel* from  $A$  to  $B$ , meaning that  $A$  can rely on the fact that only  $B$  can receive the message  $M$ .
- $A \bullet \rightarrow \bullet B : M$  represents a *secure channel*, i.e., a channel that is both authentic and confidential.

In [29], we gave two definitions for these channels: a cryptographic realization of the channels’ behavior (in which channel properties are ensured by encrypting and signing messages) and a more abstract characterization (in which we use predicates to formalize that an incoming or outgoing message is transmitted on a particular kind of channel). (Both definitions can immediately be used with the lazy intruder technique, which we will employ in the proof of our compositionality result.)

Further, in [29] we also gave a definition of channels as goals, e.g., to express that it is the goal of a protocol to authentically transmit a certain message.

This gives rise to a vertical composition question following the calculus of Maurer et al.: given a *channel protocol* that provides a certain kind of channel as a goal and an *application protocol* that assumes this kind of channel, is it safe to compose the two? While [29] tackles the logical aspect of this question, we look in this paper at the static aspect, i.e., the problems that arise from inserting the messages of one protocol into another protocol. In particular, for compositional reasoning we want to be able to verify channel protocol and application protocol separately. This means, especially, that we want to verify the channel protocol for an *abstract payload* that is independent of the application that uses the channel (and this requires more than mere protocol disjointness).

The easiest and most intuitive way to define vertical composition is at the level of Alice-and-Bob notation, and we use here the formal language AnB [25,29], which can be automatically translated to ASLan so that we can rejoice with our previous (and following) formalization of protocols. It is possible to make corresponding definitions on the ASLan level as well, but due to the handling of local state facts this is technically quite involved and distracting from our main point. An *AnB specification* of a protocol is, in a nutshell, a description of a protocol as an exchange of messages; the goal is specified as a result below a horizontal line using the channel notation about some message terms of the protocol.

**Definition 3 (Abstract Channel Protocol).** *Let  $\kappa(A, B)$  range over a set of defined channel types (e.g.,  $A \bullet \rightarrow B : M$ ) and let  $\text{payload}(A, B)$  be a mapping from pairs of agents to (abstract) payloads. An abstract channel protocol for*

---

<sup>4</sup> These are some examples of channel types, but note that in fact the details of the supported channel types do not matter for the results of this paper.

$\kappa(A, B)$  is an  $AnB$  specification that has

$$\kappa(A, B) : \text{payload}(A, B)$$

as a goal. Moreover, we require that every agent  $A$  initially knows  $\text{payload}(A, B)$  for every communication partner  $B$ .

An example of an abstract channel protocol is  $P_1$  in Fig. 1, where, following the explanation we already gave for the message in Section 2, we have that the goal of  $P_1$  is thus the transmission of the payload over a secure channel.

**Definition 4 (Application Protocol).** An application protocol for channel  $\kappa(A, B)$  is an  $AnB$  specification that contains as part of its message exchange one step

$$\kappa(A, B) : M$$

with some message term  $M$ .

An example of an application protocol is  $P_2$  in Fig. 1, in which  $C$  first sends to  $D$  a nonce  $N$  paired with a message  $M$  on a secure channel, and then  $D$  responds by sending back the challenge  $N$  together with both the agent names (this time on an insecure channel); the goal of this protocol is the secure transmission of the message  $M$  between the two agents.

### 3.2 Vertical Protocol Composition

**Definition 5 (Vertical Composition  $P_2[P_1]$ ).** Let  $P_1$  be an abstract channel protocol for  $\kappa(A, B)$  and  $P_2$  an application protocol for  $\kappa(C, D)$ .<sup>5</sup> The vertical composition  $P_2[P_1]$  is defined by replacing in  $P_2$  the step  $\kappa(C, D) : M$  with the entire protocol  $P_1$  under the renaming  $[A \mapsto C, B \mapsto D, \text{payload}(A, B) \mapsto M]$ .

An example is given in Fig. 1, where  $\text{payload}(A, B) \mapsto [N, M]_2$ .

As already mentioned, we separate the vertical composition question into a logical aspect (handled in [29]) and a static aspect. For this, we need two further definitions related to this composition.

We now define the notion of static vertical composition; it is based on a static characterization of all the messages that can occur in any run of the protocol  $P_2$  as a payload:

**Definition 6.** Let  $C$  be any honest agent, and let  $D$  be any agent. For a given protocol  $P_2$ , we define  $\mathcal{M}_{C,D}$  to be some set that includes every messages that  $C$  will ever send as payload intended for recipient  $D$ .

Some remarks about this definition are in order. First, we do not specify here precisely what the set  $\mathcal{M}_{C,D}$  is but rather allow any choice that satisfies the property that all payloads from any honest  $C$  to any  $D$  are included. In

<sup>5</sup> To avoid confusion, we assume here disjoint role names; but when there is no risk of confusion, later on in the paper, we will use the same role names in the two protocols.

fact, the entire result is parameterized over this choice. This is in the spirit of static analysis: every over-approximation of what can happen in a concrete run is possible—only if one chooses an over-approximation that is too coarse one may fail to prove a property of the concrete program. This is also the case here: if we chose, for instance, the coarsest over-approximation  $\mathcal{T}_\Sigma$  (i.e., all message terms) then we do not satisfy the seven conditions below (e.g., that the  $\mathcal{M}_{A,B}$  are pure  $P_2$  terms).

Second, a particular difficulty arises when such a payload depends on an input from another agent. Consider for instance a payload that contains a nonce (as a response) that was received (as a challenge) in a previous step. If this challenge may, in some runs, be chosen by the intruder—so we have an “unchecked input” to a payload—then we must in general put  $\mathcal{T}_\Sigma$  for the set of values that could occur as the nonce-subterm. This is because we cannot give a general limit to what messages the intruder may know at this point (which may depend, e.g., on what other protocols run in parallel with  $P_2$ ). As said above, such an over-approximation of  $\mathcal{T}_\Sigma$  is too coarse for our sufficient conditions.

As long as the payload does not contain such foreign inputs, the choice of  $\mathcal{M}_{C,D}$  is straightforward when defining fixed sets of constants from which  $C$  will choose fresh constants. For the general case when the payload does contain foreign input, we cannot give a general recipe, but we discuss some solutions for an example in Appendix B.

**Definition 7 (Static Vertical Composition  $P_1^{P_2}$ ).** *Let  $P_1$  be a channel protocol for  $\kappa(A, B)$  and  $P_2$  an application protocol for the channel  $\kappa(C, D)$ , and let  $\mathcal{M}_{C,D}$  be the set of ground messages that  $C$  can transmit over the channel  $\kappa(C, D)$  in any run of the protocol  $P_2$ . The static vertical composition  $P_1^{P_2}$  is the protocol that results from  $P_1$  by replacing payload( $A, B$ ) when it is first sent by  $A$  with a non-deterministically chosen element of  $\mathcal{M}_{C,D}$ .*

An example is given in Fig. 1, where we write  $m_{A,B}$  to denote an arbitrary message, non-deterministically chosen, from the set  $\mathcal{M}_{A,B}$ . Here,  $\mathcal{M}_{A,B} = \{[N, M]_2 \mid N, M \in \Sigma^0\}$  for honest  $A$ . A dishonest  $A$  can of course send any tuple from his knowledge as a payload and since we do not know statically the concrete knowledge, we over-approximate this choice by the set of all ground terms, i.e.,  $\mathcal{M}_{A,B} = \{[N, M]_2 \mid N, M \in \mathcal{T}_\Sigma\}$  for dishonest  $A$ .

The protocol  $P_1^{P_2}$  represents a certain “concretization” of  $P_1$  with “random” payload messages from  $P_2$ .<sup>6</sup> This notion has turned out to be very valuable because it indeed allowed us to divide the composition problem into two aspects, a logical and a static one:

**Definition 8 (Static Vertical Composability).** *Given an abstract channel protocol  $P_1$  and an application protocol  $P_2$  as in the definitions above, we say that  $P_1$  and  $P_2$  are statically vertically composable if the following implication holds: if  $P_1$  and  $P_2$  are secure in isolation, then also  $P_1^{P_2} \parallel P_2$  is secure.*

<sup>6</sup> In [29], we instead use the notion  $P_1^*$ , which, however, may be confusing here as it does not denote the protocol from which the payloads come.

We used these notions in [29] to show the desired compositionality result:

**Theorem 1 ([29]).** *If channel protocol  $P_1$  and application protocol  $P_2$  are secure protocols in isolation and they are both statically vertically composable and composable in parallel, then  $P_2[P_1]$  is secure.*

We call this result the logical aspect of the problem, because it proves that the definitions of channels as assumption and as goals have corresponding behavior, and what remains to show is static vertical composability. It turns out that the static aspect is in fact quite intricate and solving this open problem is the main contribution of this paper: the rest of this paper will concentrate on giving conditions that can be easily checked syntactically and proving that they are sufficient for a pair of protocols to be statically vertically composable, i.e., satisfying Definition 8.

As a result, we can check in isolation, with any protocol verification method, a channel protocol  $P_1$  with abstract payload as well as an application protocol  $P_2$  that uses the respective channel type. If this channel type is part of the ones defined in [29] and the sufficient conditions of this paper are satisfied for  $P_1$  and  $P_2$ , then we can combine Theorems 1 and 2 to infer that  $P_2[P_1]$  is secure.

## 4 The Conditions

We now present seven conditions on a pair of protocols, which are sufficient for the vertical composition result (cf. Theorem 2). We label the conditions with the names of the Ἐπτά ἐπὶ Θήβας [17].

**Structural Properties (Τυδεύς)** The first condition is that  $P_1$  is a channel protocol providing a  $\kappa(A, B)$  channel and  $P_2$  is an application protocol relying on a  $\kappa(A, B)$  channel according to Definitions 3 and 4, so that the compositions  $P_2[P_1]$  and  $P_1^{P_2}$  (Definitions 5 and 7) are actually defined. Let also  $\mathcal{M}_{A,B}$  be defined with respect to  $P_2$  according to Definition 7. We further assume that  $P_1$  and  $P_2$  are execution independent (Definition 2).

**Constants (Ἰππομέδων)** We require that the set  $\mathcal{A}$  of constants of the protocols is partitioned into 4 pairwise disjoint subsets  $\mathcal{A} = \mathcal{P} \uplus \mathcal{S} \uplus \mathcal{F} \uplus \mathcal{L}$  where:

- $\mathcal{S}$  is the set of *secret constants*, e.g., long-term private keys of honest agents, or long-term shared keys that are shared by only honest agents; we assume that these are never transported (they can of course be used for encryption and signing).
- $\mathcal{P} \subseteq M_0$  is the set of *public constants*, e.g., agent names, long-term public keys, long-term private keys of dishonest agents, long-term shared keys that are shared with a dishonest agent; these are part of the initial knowledge  $M_0$  of the intruder.

$$\begin{aligned}
MP(P_1) &= \{ \{ \{ [p, A, B, \text{payload}(A, B)]_4 \}_{\text{privk}(\text{pk}(A))} \}_{\text{pubk}(\text{pk}(B))}, \{ \{ [p, A_1, B_1, X_1]_4 \}_{\text{privk}(\text{pk}(A_1))} \}_{\text{pubk}(\text{pk}(B_1))} \} \\
SMP(P_1) &= MP(P_1) \cup \{ \{ [p, A_2, B_2, \text{payload}(A_2, B_2)]_4 \}_{\text{privk}(\text{pk}(A_2))}, \{ [p, A_3, B_3, X_3]_4 \}_{\text{privk}(\text{pk}(A_3))}, \\
&\quad \text{pubk}(\text{pk}(B_4)), [p, A_5, B_5, \text{payload}(A_5, B_5)]_4, [p, A_6, B_6, X_6]_4, \text{privk}(\text{pk}(A_7)), p, \text{payload}(A_8, B_8) \} \\
MP(P_2) &= SMP(P_2) = \{ [N_1, C_1, D_1]_3, [N_2, M_2]_2 \}
\end{aligned}$$

**Fig. 2.**  $MP$  and  $SMP$  for the example protocols of Fig. 1.

- $\mathcal{F}$  is the set of the *fresh constants*, i.e., whenever an agent randomly generates new keys or nonces, they will be picked uniquely from this set. As is standard, the intruder by default does not know the fresh constants created by honest agents, but may learn them from messages that the agents send.  $\mathcal{F}$  is further partitioned into the two disjoint subsets  $\mathcal{F}_1$  and  $\mathcal{F}_2$  of fresh constants of  $P_1$  and  $P_2$ , respectively.
- $\mathcal{L}$  is the set of *abstract payloads* (i.e., those denoted by  $\text{payload}(A, B)$ ). These may only occur in the protocol  $P_1$ , and are replaced by concrete payloads in  $P_1^{P_2}$  and  $P_2[P_1]$ . We discuss the initial knowledge of the abstract payloads below.

**Disjointness (Καπανεύς)** We require that the message formats are sufficiently different to distinguish  $P_1$  terms and  $P_2$  terms—except for constants like agent names, nonces, and keys. The reasons for the exception on constants are that constants (a) may be shared between protocols like agent names and keys, and (b) by their construction usually cannot be attributed to a unique protocol, e.g., nonces.

- The *message patterns*  $MP$  are the terms that represent messages sent and received by honest agents in the protocol description, where we ensure by renaming of variables that distinct elements of  $MP$  have disjoint variables. Let  $SMP$  be the *non-atomic subterms* of the message patterns (with the same variable renaming). For instance, for the protocols of Fig. 1,  $MP$  and  $SMP$  are as shown in Fig. 2, where we write  $A, B, C, D$  as placeholders for arbitrary agents for the sake of readability.

We require message patterns not to be atomic:<sup>7</sup>  $MP(P_i) \cap (\mathcal{V} \cup \Sigma^0) = \emptyset$  for  $i \in \{1, 2\}$ . Moreover, non-atomic subterms must be disjoint:  $SMP(P_1) \cap SMP(P_2) = \emptyset$ , where  $M \sqcap N = \{ \sigma \mid \exists m \in M, n \in N. m\sigma = n\sigma \}$ .

We exclude atomic message patterns since otherwise we would have messages for which we cannot ensure that they are attributed to a unique protocol.

- By the previous item, the following labeling is possible on all message patterns in the protocol description. Every non-atomic subterm  $m$  is labeled

<sup>7</sup> Recall that mappings like  $\text{pk}(a)$  map from atoms to atoms and thus  $\text{pk}(a)$  also counts as atomic in the sense of this definition.

- either  $P_1$  or  $P_2$ , in symbols  $m : P_i$ . There is only one unique such labeling because the spaces of non-atomic subterms of the  $P_i$  must be disjoint.
- Next, we can also label the atomic subterms except public and secret constants in a unique way: we label them by the label of the next surrounding operator. We will make one exception from this rule below for the payload (the payload is of type  $P_2$  but it is embedded into a  $P_1$  context), but in order not to break the flow of the argument we postpone this a bit.
  - We additionally require that the sets of variables used in the descriptions of  $P_1$  and  $P_2$  are disjoint, and that fresh constants are chosen from the disjoint sets  $\mathcal{F}_i$ . Therefore, no variable or constant can have an ambiguous labeling.

**Disjointness of Concrete Payloads (Πολυωνείκης)** We require that  $\mathcal{M}_{A,B}$  comprise only of ground  $P_2$  terms. Moreover, the concrete sets of payload terms  $\mathcal{M}_{A,B}$  must be pairwise disjoint for honest sender, i.e.,  $\mathcal{M}_{A,B} \cap \mathcal{M}_{A',B'} = \emptyset$  whenever  $A$  and  $A'$  are honest and  $(A \neq A'$  or  $B \neq B')$ .<sup>8</sup> This does not imply that honest agents or the intruder can recognize from a payload who is (the claimed)  $A$  and  $B$ .

**Payload Type and Context (Έπέοκλος)** We now make some provisions about the handling of payload messages in the protocols. First, without restricting the class of protocols, we label those subterms in the message patterns that represent payloads sent or received by honest agents (this is made precise below). Second, we require that these payload subterms when sent or received by honest agents are always embedded into a unique *payload context*  $C_P[\cdot]$  that unambiguously signals the subterm is meant as a payload and so that no other message parts are accidentally interpreted as payloads (this is also made precise below). We require that this context is an  $n$ -tuple of the form  $[m_1, \dots, m_n]_n$  where one of the  $m_i$  is the payload and the other  $m_i$  are all constants, e.g., tags or agent names. For instance, our example abstract channel protocol  $P_1$  in Fig. 1 uses the context  $C_P(A, B)[\cdot] = [\mathfrak{p}, A, B, \cdot]_4$ . Note that we here actually allow that contexts may be parameterized over additional information such as the involved agents, but for simplicity of notation only write  $C_P[\cdot]$  whenever no confusion arises. Note also that the context alone does not protect the payload against manipulation or even reading by other parts, as this depends on the goal of the channel protocol. Moreover, we will assume that an intruder can always create such contexts.

The precise requirements about the labeling and the occurrence of contexts are as follows:

1. ( $C_P[\cdot]$  identifies payload.) In  $P_1$  and  $P_1^{P_2}$  terms: For every term  $C_P[m]$  it holds that  $m$  is typed *Payload*, and all *Payload*-typed messages  $m$  occur as  $C_P[m]$ . Moreover, in  $P_1^{P_2}$  such a message  $m$  is a concrete payload from  $P_2$  is

<sup>8</sup> This could be achieved by inserting a constant into the payload chosen from a set  $\mathcal{X}_{A,B}$  of a family of disjoint sets  $\mathcal{X}_{A,B}$ .

thus labeled as belonging to  $P_2$ , while in  $P_1$ , we have abstract payloads that are labeled  $P_1$ .

2. ( $C_P[\cdot]$  has blank-or-concrete payloads): each payload that occurs under  $C_P[\cdot]$  (i.e., in all  $P_1$ -messages) is either ground or a variable. The ground case represents an honest agent sending a concrete payload from  $\mathcal{M}_{A,B}$ , and we therefore additionally label it  $\text{spayload}(A, B)$ . The variable case means that an agent receives a payload, and we thus label it  $\text{rpayload}(A, B)$  (where  $B$  is the name of the receiver and  $A$  is the supposed sender). The fact that we here allow only a variable means that the channel protocol  $P_1$  is “blind” for the structure of the payload messages from  $P_2$ , and thus any value is accepted. We also assume that if a variable occurs as payload under  $C_P[\cdot]$ , then all other occurrences of that variable are also under  $C_P[\cdot]$ .
3. (Payloads in  $P_2$ .) In  $P_2$  terms, the message  $m$  that is transmitted as payload over the  $\kappa(A, B)$ -channel is of type  $\text{Payload}$  and every ground instantiation  $m^{\mathcal{I}}$  must be a member of  $\mathcal{M}_{A,B}$ . Similarly to the labeling for  $P_1^{P_2}$  in the previous item,  $m$  is either ground and labeled  $\text{spayload}(A, B)$ , or it is a symbolic term (not necessarily a variable) and labeled  $\text{rpayload}(A, B)$ .
4. ( $C_P[\cdot]$  cannot be confused in  $P_1$ .) For any  $t \in \text{SMP}(P_1)$  where  $t \neq C_P[t']$  for any term  $t'$ , then  $\{t\} \cap \{C_P[s] \mid s \in \mathcal{T}_{\Sigma}(\mathcal{V})\} = \emptyset$ . By (Καπανεύς), it also follows that  $C_P[\cdot]$  is disjoint from  $P_2$ -typed message parts as it belongs to  $P_1$ .

This assumption ensures that every subterm  $m$  of a  $P_1$ -term that is of type  $\text{Payload}$  is either a variable or ground, and if ground, then  $m \in \mathcal{M}_{A,B}$  for some uniquely determined  $A$  and  $B$  and this is indicated by the appropriate label  $\text{spayload}(A, B)$ . In the proof in Appendix A, we show that any attack would equally work when replacing every concrete  $m$  with the abstract  $\text{payload}(A, B)$ .

**Abstract Payloads (Παρθενοναίος)** In the channel protocol  $P_1$  with abstract payload  $\text{payload}(A, B)$  we require that the intruder knows initially

$$\{\text{payload}(A, B) \mid \text{dishonest}(A) \vee \text{dishonest}(B)\}$$

if the channel-type  $\kappa(A, B)$  includes secrecy (i.e., if it is  $A \rightarrow \bullet B$  or  $A \bullet \rightarrow B$ , which we can denote by  $\text{secrecy} \in \kappa(A, B)$ ); otherwise he initially knows all payloads  $\{\text{payload}(A, B)\}$ . With this, we assume that the intruder may “in the worst case” know all payloads that are not explicitly secret (even though he may not find out the actual payload in a concrete run of  $P_2[P_1]$  or  $P_1^{P_2}$ ). With this, we are “on the safe side” in forbidding that the security of  $P_1$  could rely on the secrecy of some abstract payload that is possibly publicly known in a concrete application with  $P_2$ .

**Properties of Concrete Payloads (Αμφιάρσος)** For the concrete payloads, we similarly require that all payloads that are not explicitly secret are included in the initial intruder knowledge of  $P_1^{P_2}$  and  $P_2$ , i.e., initially the intruder knowledge  $M_0$  contains at least:

$$\bigcup_{\text{honest}(A) \wedge (\text{dishonest}(B) \vee \text{secrecy} \notin \kappa(A, B))} \mathcal{M}_{A, B}.$$

Moreover, all the other—secret—payloads (when  $A$  and  $B$  are honest and  $\kappa(A, B)$  entails secrecy) must be considered as secrets in  $P_1$  and  $P_2$  (and thus they are also secrets in  $P_1^{P_2}$ ). This can be expressed in ASLan for instance by adding the fact  $\text{secret}(M, \{A, B\})$  in every transition where an honest agent sends a payload  $M$  labeled  $\text{spayload}(A, B)$  or receives a payload  $M$  labeled  $\text{rpayload}(A, B)$  and using the general attack rule (1).<sup>9</sup> We are now ready for our main result (proved in Appendix A), namely that the conditions are sufficient for static vertical composability, i.e., if  $P_1$  and  $P_2$  are secure, then so is  $P_1^{P_2} \parallel P_2$ . Or, in other words, that we can reduce an attack against  $P_1^{P_2} \parallel P_2$  to an attack against one of the component protocols.

**Theorem 2.** *Consider two protocols  $P_1$  and  $P_2$  that satisfy all the seven conditions. If there is an attack against  $P_1^{P_2} \parallel P_2$ , then there is an attack against  $P_1$  or against  $P_2$ .*

Let us give here a proof sketch, postponing the full proof to in Appendix A. In the proof, we employ the constraint reduction technique that we refer to as the *lazy intruder* (see, e.g., [5]). While this technique is originally a verification technique (for a bounded number of sessions), we use it here for a proof argument for our compositionality result (for an unbounded number of sessions). The key idea of the lazy intruder is to model “symbolic executions” where variables in the messages that honest agents receive from the insecure network (i.e., from the intruder) are left un-instantiated. We use intruder constraints of the form  $M \vdash t$  where  $t$  is a (symbolic) term that an agent is able to receive and  $M$  is the current intruder knowledge. We use the fact that we can check satisfiability of such constraints using the lazy intruder calculus, and that we can reduce insecurity to satisfiability of lazy intruder constraints.

We thus assume we are given lazy intruder constraints for an attack against the composition  $P_1^{P_2}$  for any channel protocol  $P_1$  and application protocol  $P_2$  that satisfy our seven conditions. We then show that over all reductions with the lazy intruder calculus, the seven conditions and some further invariants are preserved, in particular, that the attack never requires a confusion between  $P_1$  and  $P_2$  messages. Note that this is because the lazy intruder technique never instantiates variables whose concrete value is irrelevant for the attack (hence we call it *lazy* in the first place); these still admit “ill-typed” instantiations (confusing  $P_1$  and  $P_2$ ), but they always also admit well-typed instantiations. As a consequence, we can show that there exists an attack against  $P_1$  in isolation or against  $P_2$  in isolation.

We emphasize once again that the results are independent both of the verification technique used for verifying the atomic components and of the formalism employed to model protocols such as rewriting, strands, or process calculi.

<sup>9</sup> Pedantically, to fulfill the condition of protocol independence, we should use two distinct facts  $\text{secret}_{P_1}$  and  $\text{secret}_{P_2}$  for secrecy in the respective subprotocols.

Let us thus take stock. We have formalized seven static conditions that are sufficient for vertical protocol composition for a large class of channels and applications. Our results tell us that we can check in isolation — with any protocol verification method<sup>10</sup> — a channel protocol  $P_1$  with abstract payload, as well as an application protocol  $P_2$  that uses the respective channel type. If this channel type is part of the ones defined in [29] and the sufficient conditions of this paper are satisfied for  $P_1$  and  $P_2$ , then we can combine Theorems 1 and 2 to infer that  $P_2[P_1]$  is secure.

## 5 Extension to more messages

We thus solved the static vertical composition question as it was posed in [29]. The reader may agree with us at this point that that problem is complex enough and forgive us for not complicating things further with more generality. (Our seven conditions appear complex because they are formulated at a deep technical level, but they actually reflect realistic static properties satisfied by many protocols.) Now, we want, however, to discuss what are the limitations of the composability result so far and how we can extend it to the case of more messages for what concerns both the static aspect and the logical aspect of vertical protocol composition. We return to this in more detail in Appendix A.4, in which we consider more formally the extension of our composability result with more messages (as well as with constraints that represent the negation of a substitution).

The composability result of [29] refers to only one single payload message of the application protocol being transmitted over the channel provided by the channel protocol. There are two reasons why this is a limitation. First, if the channel protocol is complex (and consisting of many steps), it is not desirable to execute this entire protocol for every message transmission of an application protocol. Second, disjointness conditions would not even allow repeated applications of the composability result, i.e.,  $P_2[P_1[P_1]]$  when we have two messages in  $P_2$  that should be transmitted over a channel provided by  $P_1$ .

Our conjecture is that there is no insurmountable obstacle to allowing the definition of a channel protocol for more than one message transmission. One obvious way to go is to generalize the channel protocol to the transmission of several payload messages  $\text{payload}_1(A_1, B_1), \dots, \text{payload}_k(A_k, B_k)$  for a fixed number  $k$  of transmissions (the endpoints of the channels may differ); these transmissions would be over  $k$  different channel types  $\kappa_i(A_i, B_i)$ ; they would be reflected by  $k$  disjoint contexts  $C_P^i[\cdot]$ , and the application protocol can then transmit  $k$  messages with associated concrete payload message sets  $\mathcal{M}_{A,B}^i$  (for  $1 \leq i \leq k$ ). These payload message sets would have to be disjoint unless  $\kappa_i(A_i, B_i) = \kappa_j(A_j, B_j)$ . The respective extensions of the definitions and proofs are notationally involved, but conceptually simple, so we avoided them here.

<sup>10</sup> Such as ProVerif [6] or the AVANTSSAR Platform [3], where for bounded-session tools compositionality only holds for those bounded sessions.

More generally, we also like to allow the transmission of an unbounded number of messages over a channel. The most prominent examples for this are, of course, secure channel protocols like TLS that establish a pair of symmetric keys (one for client-to-server transmissions, and one for server-to-client). We discuss an example based on TLS in more detail in Appendix B; this includes a suitable notation for the transmission protocol(s), i.e., how payload messages are handled. Note that we are here focussing only on the channel’s transmission properties for the single messages such as authentication and secrecy, not for their relationship such as their ordering, completeness or replay protection.

Again, there is no a fundamental problem in extending our static vertical composition result for arbitrary message transmissions as long as, again, the message spaces  $\mathcal{M}_{A_i, B_i}^i$  for the different used channel types are disjoint. In particular, observe that we require honest receivers in the channel protocol to accept any payload that is embedded into the proper context; thus, the abstraction of the payload in the pure  $P_i$  works, independent of whether there is just one concrete payload message per session or many of them.

We also conjecture that the principles of vertical protocol composition of [29] can also be extended to arbitrary payload transmissions. However, since this is beyond the scope of this paper, we leave it for future work.

## 6 Conclusions

We have formalized seven static conditions that are sufficient for vertical protocol composition for a large class of channels and applications. Our results tell us that we can check in isolation—with any protocol verification method—a channel protocol  $P_1$  with abstract payload, as well as an application protocol  $P_2$  that uses the respective channel type. If this channel type is part of the ones defined in [29] and the sufficient conditions of this paper are satisfied for  $P_1$  and  $P_2$ , then we can combine Theorems 1 and 2 to infer that  $P_2[P_1]$  is secure.

These conditions appear complex because they are formulated at a deep technical level of lazy intruder constraints, but they actually reflect realistic static properties satisfied by many protocols.

As we have already mentioned above, there are a number of interesting directions for future work, in particular, allowing for negative checks also on the channel protocol when considering finer abstractions and formalizing the extension of our sufficient conditions to the case of more messages for what concerns both the static aspect and the logical aspect of vertical protocol composition.

## References

1. S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. Mjølsnes, and S. Radomirović. A framework for compositional verification of security protocols. *Information and Computation*, 206:425–459, 2008.
2. M. Arapinis and M. Dufлот. Bounding messages for free in security protocols. In *Proceedings of FST TCS’07*, LNCS 4855, pages 376–387. Springer, 2007.

3. A. Armando, W. Arzac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, and L. Viganò. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In *Proceedings of TACAS*, LNCS 7214, pages 267–282, 2012.
4. The AVANTSSAR Project: Deliverable 2.3: ASLan (final version), 2010. Available at [www.avantssar.eu](http://www.avantssar.eu).
5. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
6. B. Blanchet. From secrecy to authenticity in security protocols. In *Proceedings of SAS'02*, LNCS 2477, pages 342–359. Springer, 2002.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS'01*, pages 136–145. IEEE Computer Society, 2001.
8. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. In *Proceedings of FST TCS 2003*, LNCS 2914, pages 124–135. Springer, 2003.
9. S. Ciobâca and V. Cortier. Protocol composition for arbitrary primitives. In *Proceedings of CSF'10*, pages 322–336. IEEE Computer Society, 2010.
10. V. Cortier and S. Delaune. Safely composing security protocols. *Formal Methods in System Design*, 34(1):1–36, 2009.
11. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *Proceedings of FMSE'03*, pages 11 – 23. ACM, 2003.
12. S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proceedings of CSF 21*, pages 239–251. IEEE Computer Society Press, 2008.
13. T. Groß and S. Mödersheim. Vertical protocol composition. In *Proceedings of CSF 2011*, pages 235–250. IEEE CS, 2011.
14. J. D. Guttman. Authentication tests and disjoint encryption: a design method for security protocols. *Journal of Computer Security*, 3–4(12):409–433, 2004.
15. J. D. Guttman. Cryptographic protocol composition via the authentication tests. In *Proceedings of FOSSACS'09*, LNCS 5504, pages 303–317. Springer-Verlag, 2009.
16. J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proceedings of CSFW 2000*, pages 24–34, 2000.
17. G. O. Hutchinson. *Aeschylus. Seven against Thebes*. Clarendon Press, Oxford, 1985.
18. R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In *Proceedings of CCS 18*, pages 41–50. ACM Press, 2011.
19. S. Malladi and P. Lafourcade. How to prevent type-flaw attacks under algebraic properties. In *Security and Rewriting Techniques*, 2009. <http://arxiv.org/abs/1003.5385>.
20. U. M. Maurer. Constructive cryptography – A new paradigm for security definitions and proofs. In *Proceedings of TOSCA 2011*, LNCS 6993, pages 33–56. Springer, 2011.
21. U. M. Maurer and R. Renner. Abstract Cryptography. In B. Chazelle, editor, *Proceedings of ICS 2011*, pages 1–21. Tsinghua University Press, 2011.
22. U. M. Maurer and P. E. Schmid. A calculus for security bootstrapping in distributed systems. *J. Comp. Sec.*, 4(1):55–80, 1996.

23. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.
24. S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD Thesis, ETH Zurich, 2007. ETH Dissertation No. 17013.
25. S. Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares'09*, 2009.
26. S. Mödersheim. Abstraction by Set-Membership: Verifying Security Protocols and Web Services with Databases. In *Proceedings of CCS 17*, pages 351–360. ACM Press, 2011.
27. S. Mödersheim. Diffie-Hellman without Difficulty. In *Proceedings of FAST 2011*. Springer, 2011. Extended version technical report IMM-TR-2011-13, DTU Informatics, 2011.
28. S. Mödersheim. Deciding Security for a Fragment of ASLan (Extended Version). Technical Report IMM-TR-2012-06, DTU Informatics, 2012. Available at [imm.dtu.dk/~samo](http://imm.dtu.dk/~samo).
29. S. Mödersheim and L. Viganò. Secure pseudonymous channels. In *Proceedings of ESORICS 14*, LNCS 5789, pages 337–354. Springer, 2009.
30. S. Mödersheim and L. Viganò. The Open-Source Fixed-Point Model Checker for Symbolic Analysis of Security Protocols. In *FOSAD 2008/2009*, LNCS 5705, pages 166–194. Springer-Verlag, 2009.
31. S. Mödersheim, L. Viganò, and D. A. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, 18(4):575–618, 2010.
32. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.

## A The Sufficiency of the Conditions

As we anticipated above, in the proof, we employ the constraint reduction technique called the lazy intruder. Before proving the conditions, we thus introduce the lazy intruder in detail.

### A.1 Intruder Deduction

We consider a Dolev-Yao-style intruder model, in which the intruder controls the insecure channels in the network, including that he can send messages under an arbitrary identity. Moreover, he may act, under his real name, as a normal agent in protocol runs. We generalize this slightly and allow the intruder to have more than one “real name”, i.e., he may have several names that he controls, in the sense that he has the necessary long-term keys to actually work under a particular name. This reflects a large number of situations, like an honest agent who has been compromised and whose long-term keys have been learned by the intruder, or when there are several dishonest agents who all collaborate. This worst case of a collaboration of all dishonest agents is simply modeled by one intruder who acts under different identities. To that end, we use the predicate

$$\frac{\phi\sigma}{\phi \wedge (M \vdash t)} \text{ Unify } (s, t \notin \mathcal{V}, s \in M, \sigma \in \text{mgu}(s, t)) \quad \frac{\phi \wedge (M \vdash t_1) \wedge \dots \wedge (M \vdash t_n)}{\phi \wedge (M \vdash f(t_1, \dots, t_n))} \text{ Generate } (f \in \Sigma_P^n)$$

$$\frac{(\bigwedge_{k \in K} M \vdash k) \wedge \text{Add}(P, M, \phi \wedge (M \vdash t))}{\phi \wedge (M \vdash t)} \text{ Analysis } ((K, P) \in \text{ana}(s), s \in M)$$

where

$$\text{Add}(P, M, \phi \wedge (M' \vdash t)) = \text{Add}(P, M, \phi) \wedge \begin{cases} M' \cup P \vdash t & \text{if } M' \supseteq M \\ M' \vdash t & \text{otherwise} \end{cases}$$

$$\text{Add}(P, M, \text{true}) = \text{true}$$

**Fig. 3.** The lazy intruder reduction rules.

$\text{dishonest}(A)$  that holds true for every dishonest agent  $A$  (from the initial state on). Dually, we use the predicate  $\text{honest}(\cdot)$  to identify honest agents.

The intruder can compose terms applying public functions of  $\Sigma_P$  to terms that he knows, and he can decompose terms when he knows the necessary keys. The latter is formalized by a function  $\text{ana}(\cdot)$  that takes as argument a message  $m$  and returns a set of potential ways to extract information from  $m$ . Each way to extract information has the form  $(K, P)$  where  $P$  (“plaintexts”) is a set of messages that can be extracted when the messages  $K$  (“keys”) are known. In this paper, we use:

$$\text{ana}(m) = \begin{cases} \{(\{\text{privk}(s)\}, \{p\})\} & m = \{p\}_{\text{pubk}(s)} \\ \{(\{k\}, \{p\})\} & m = \{p\}_k \\ \{(\emptyset, \{p_1, \dots, p_n\})\} & m = [p_1, \dots, p_n]_n \\ \{(\{\text{pubk}(s)\}, \{p\})\} & m = \text{sign}(\text{privk}(s), p) \\ \emptyset & \text{otherwise} \end{cases}$$

**Definition 9.** We write  $M \vdash m$  to denote that the intruder can derive the ground message  $m$  when knowing the set of ground messages  $M$ . We define  $\vdash$  as the least relation that satisfies the following rules:

- (D)  $M \vdash m$  for all  $m \in M$ ,
- (G) if  $M \vdash t_1, \dots, M \vdash t_n$ , then also  $M \vdash f(t_1, \dots, t_n)$  for all  $f \in \Sigma_P^n$ ,
- (A) if  $M \vdash m$  and  $(K, P) \in \text{ana}(m)$  and  $M \vdash k$  for all  $k \in K$ , then also  $M \vdash p$  for all  $p \in P$ .

All terms are interpreted in the free algebra.

Although this definition is given only for ground  $m$  and  $M$ , in the following we will use the symbol  $\vdash$  in *constraints* that contain variables.

## A.2 The Lazy Intruder

We now review the constraint reduction technique of [23,32,8,5] that we refer to as the *lazy intruder*. While this technique is originally a verification technique

(for a bounded number of sessions), we use it here for a proof argument for our compositionality result (without bounding the number of sessions), quite similar to [2,19,27].

The core idea behind the lazy intruder is to avoid the naive enumeration of the (large or even infinite) number of messages that the intruder can construct and send from a given set of known messages. Instead, the lazy intruder technique uses a symbolic, constraint-based approach and thereby significantly reduces the search space without excluding attacks and without introducing new ones.

Slightly abusing the original idea, we can employ the lazy intruder technique for proving compositionality results (and other relative soundness results) in a very convenient way. In our case, we consider a composed system (two vertically composed protocols) that satisfies our sufficient conditions and a symbolic attack trace against this system. We then can show that, thanks to the conditions, all constraint reduction steps would work on the atomic components of the system in isolation. In other words, the attack does not necessarily rely on the interaction between the component protocols, and, intuitively speaking, then the lazy intruder is too lazy to make use of such interactions. As a result we know that, if there is an attack, then one of the component protocols must have one; vice versa, secure component protocols that satisfy our sufficient conditions always yield secure compositions.

We emphasize that the results are independent both of the verification technique used for verifying the atomic components and of the formalism employed to model protocols such as rewriting, (symbolic) strands, or process calculi. To abstract from these, we now review the notion of a symbolic transition system.

**Symbolic Transition Systems** As defined in [31], we assume that a protocol can be represented by a *symbolic state transition system*. A state represents the local state of all the honest agents and the knowledge of the intruder. The message terms that occur in the state may contain variables that represent choices that the intruder made earlier. The values that these variables can take are governed by constraints of the form  $M \vdash m$ , where  $M$  and  $m$  may contain variables. These constraints express that only interpretations of the variables are allowed under which the message  $m$  can be derived from knowledge  $M$ .

We can derive a symbolic transition system from most protocol formalisms in a straightforward way. Whenever we have that an honest agent wants, as its next action, to send a message  $m$  on an insecure network, we simply add  $m$  to the intruder knowledge. Whenever an honest agent wants to receive a message of the form  $m$  on an insecure network, where  $m$  is a term with variables representing subterms where any value is acceptable, then we simply add the constraint  $M \vdash m$  where  $M$  is the current intruder knowledge, and let the agent proceed without instantiating the variables in  $m$ . This also works if the variables in  $m$  are related to other terms in the local state of the honest agent. Also, we do not forbid agent knowledge that spans several sessions, such as a data-base that a web-server may maintain. A complication that we want to leave out here, however, is negative checks on messages (i.e., checking that  $m \neq m'$  for some

other message  $m'$ ); we discuss later (cf. Section A.4) that our results still apply under certain conditions.

The lazy intruder requires one condition about the sent and received messages: variables originate only from the intruder, i.e., an agent description is *closed* in the sense that it only contains variables that are *bound* by a preceding receiving step. Thus, all “parameters” of an agent description (e.g., its name and the names of its peers, or freshly created values) must be instantiated initially.

Throughout the paper we will thus take for granted that every protocol trace can be represented by lazy intruder constraints. We will also take for granted that successful attacks can also be represented with the same machinery, i.e., as a satisfiability problem of lazy intruder constraints. This can be done, for instance, by means of special protocol roles whose completion represents an attack.

For simplicity, we thus work directly at the level of constraints, in particular formulating several of the sufficient conditions directly as properties of the lazy intruder constraints that the symbolic execution of the protocol produces. Again, we emphasize that this is a matter of convenience, being independent of a protocol specification formalism and without precluding any verification technique to analyze the individual protocols.

**Semantics of Constraints** We consider *constraints* that are conjunctions of  $M \vdash m$  statements where both the set of messages  $M$  and the message  $m$  may contain variables. An *interpretation*  $\mathcal{I}$  assigns a ground term to every variable; we write  $\mathcal{I}(v)$  to denote the interpretation of a variable  $v$  and extend this notation to messages, sets of messages, and constraints as expected. We inductively define the relation  $\mathcal{I} \models \phi$  to formalize that *interpretation*  $\mathcal{I}$  is a *model of constraint*  $\phi$ :

$$\mathcal{I} \models M \vdash m \quad \text{iff} \quad \mathcal{I}(M) \vdash \mathcal{I}(m) \qquad \mathcal{I} \models \phi \wedge \psi \quad \text{iff} \quad \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \psi$$

A constraint is *satisfiable* if it has at least one model.

**Constraint Reduction** The core of the lazy intruder is a set of reduction rules, shown in Fig. 3, based on which we can check in finitely many steps whether a given constraint is satisfiable. Before we discuss the rules, let us first review the idea of constraint reduction. The reduction rules work similar to the rules of a proof calculus in several regards. A rule of the form

$$\frac{\phi'}{\phi}$$

tells us that, in order to show the satisfiability of constraint  $\phi$  (the proof goal), it suffices to show the satisfiability of constraint  $\phi'$  (the sub goal). So we apply the rules in a backward fashion in the search for a satisfiability proof. This process succeeds once we find a *simple* constraint, which is one that consists only of conjuncts of the form  $M \vdash v$  where  $v$  is a variable. A simple constraint is obviously satisfiable: the intruder can choose for each variable an arbitrary message that he can construct. In fact, the laziness of the intruder manifests

itself exactly here in avoiding the exploration of choices that do not matter. That is the substitution of variables during search is postponed as long as possible, in a demand-driven (“lazy”) way.

Comparing to a proof calculus, one could call the simple constraints the “axioms” and we check whether for a given constraint  $\phi$  any proof can be constructed using the reduction rules that has  $\phi$  as a root and only simple constraints (“axioms”) as leaves. *Soundness* of such a calculus of reduction rules means that we never obtain a “proof” for an unsatisfiable constraint, and *completeness* means that every satisfiable constraint has a proof. There are further relevant properties such as finiteness of the set of reachable proof states, and the completeness of certain proof strategies. These play a minor role in this paper because we do not use the lazy intruder to implement an efficient model checker, but rather use the existence or non-existence of certain reduction as a proof argument in the proof of our main theorems.

Let us now consider the details of the rules in Fig. 3.

*Unify* The Unify rule says that one way for the intruder to produce a term  $t$  is to use any term  $s$  in his knowledge that can be unified with  $t$ . Here,  $mgu(s, t)$  means the set of most general unifiers between  $s$  and  $t$  (note that there can be several in unification modulo the property of exponentiation). In case  $\sigma$  is such a unifier, we have solved the constraint  $M \vdash t$  and apply  $\sigma$  to the remaining constraint  $\phi$  to be solved. We make here also an essential restriction: neither  $s$  nor  $t$  shall be variables. If  $t$  is a variable, then the constraint  $M \vdash t$  is already simple and should not be reduced to achieve the laziness. The case that  $s$  is a variable is more involved. Roughly speaking, such a variable will represent a value chosen by the intruder “earlier” and so whatever it is, he can also generate the same value from  $M$  already. This will be made precise below with the notion of well-formed constraints.

*Generate* The Generate rule tells us that the intruder can generate the term  $f(t_1, \dots, t_n)$  if  $f$  is a public symbol of  $\Sigma_P^b$  and if the intruder can generate all the subterms  $t_1, \dots, t_n$ . So this simply represents the intruder applying a public function (such as encryption) to a set of terms he already knows.

*Analysis* The Analysis rule represents the intruder trying to decompose messages in his knowledge such as decrypting with known keys. Given the intruder knows a message  $s$  from which he can learn  $P$  provided he knows  $K$ , we can go to a new constraint where the knowledge is augmented with the messages of  $P$  and where we have the additional constraints that the intruder can generate every  $k \in K$ . In fact, in actual implementations this rule must be carefully implemented to avoid non-termination of the search. For the same reason as in the case of the Unify rule, we do not analyze  $s$  if it is a variable, because then—the way we use it—it represents a message created earlier by the intruder.

It is not difficult to show that all rules of the calculus are sound (see [27] for a proof).

*Well-Formedness* We can define an order on the conjuncts of constraints, talking about earlier/later constraints. This order is essential for the constraint reduction. The idea is that the intruder does a sequence of actions during an attack and his knowledge monotonically grows with every message he learns. Moreover, variables that occur in messages sent by honest agents must have appeared in previous messages and thus represent values that depend on the choice of the intruder (though they might not be chosen by the intruder himself).

**Definition 10.** *We say that a constraint  $\phi$  is well-formed if it has the form (modulo reordering conjuncts)*

$$\phi = \bigwedge_{i=1}^n M_i \vdash t_i$$

*such that  $M_i \subseteq M_j$  for  $i \leq j$ , expressing that the intruder never forgets, and  $\text{vars}(M_i) \subseteq \bigcup_{j=1}^{i-1} \text{vars}(t_j)$ , expressing that all variables arise from intruder choices.*

As shown in, e.g., [5,23,24], the calculus is complete on well-formed constraints.

**Lazy Intruder Representation of Attack Traces** As explained already above, we want to use the lazy intruder technique as a proof argument of our main result, although it was originally designed as an automated verification technique. Note that while the technique can be a decision procedure only when the number of sessions is bounded, our proof argument works for any number of sessions—we just rely on the fact that attack traces are finite (because we use standard reachability notions).

The original lazy intruder technique could be formulated on a non-deterministic Turing machine: starting from the initial state we “guess” a sequence of rule applications that may lead to an attack (this sequence may apply the same rule many times). The last step of the sequence is an attack rule. Rather than working on ground states however—which would require to match every  $\text{iknows}(m)$  fact on the left-hand side of an IF rule with any of the infinitely many messages the intruder can generate from his knowledge  $M$  in that state, we do not instantiate any variables in  $m$  and simply require the constraint  $M \vdash m$ . (Thus rule matching must be replaced with rule unification throughout the trace.) Finally, the machine checks for the guessed solution whether the conjunction of  $M \vdash m$  constraints is satisfiable—using the constraint reduction above. If so, we have found an attack. The machine answers “safe” if no guess of a sequence of rules leads to an attack.

We now exploit the fact that (even for an unbounded number of sessions), any attack means there is such a finite sequence of symbolic rule application, producing a satisfiable conjunction of constraints. We show that, when applying the lazy intruder to these constraints, certain invariants will hold on every reduction step, e.g. that no  $P_1$  variables are instantiated with  $P_2$  terms. Since the

lazy intruder is complete, we then know: if there is an attack, then there is an attack that satisfies certain properties (such as never confusing messages from different protocols).

The formulation of most of our conditions carries over verbatim to the lazy intruder constraints produced by the symbolic attack trace; it is an invariant that they will continue to hold over all reduction steps, with few exceptions: we only need to reformulate execution independence and (parts of) (Καπανεύς), (Έτέοκλος) and (Άμφιάραος). The changes and further notions for the constraints are as follows:

- Execution independence: We formulate this using symbolic transitions and intruder constraints as follows. We say that  $P_1$  and  $P_2$  are *execution independent* if the following property holds. Given a symbolic trace for the parallel composition of two protocols  $P_1$  and  $P_2$ , if we project on the events belonging to one of the two protocols, say  $P_1$ , and accordingly filter the constraints (and the knowledge part of the constraints), then we obtain a symbolic trace of  $P_1$ . And if the symbolic trace was an attack against a goal of  $P_1$ , then so is the projection.

Observe that this definition says nothing about the *satisfiability* of the intruder constraints. In general, it may well be that an attack trace against the parallel composition of two execution-independent protocols  $P_1$  and  $P_2$  has satisfiable constraints, while the constraints of the projection to  $P_1$  may be unsatisfiable, so the intruder actually needs to know messages from  $P_2$  to perform the attack. Therefore, this property of execution independence alone does not yet give parallel composability; it only requires that the executability of the steps of  $P_1$  is independent of what steps of  $P_2$  have been executed (and vice versa).

- (Καπανεύς):
  - Every non-atomic subterm  $m$  of the constraints is labeled either  $P_1$  or  $P_2$ , in symbols  $m : P_i$ . There is only one unique such labeling because the spaces of non-atomic subterms of the  $P_i$  must be disjoint.
  - We initially label constraints  $(M \vdash m) : P_i$  if  $m : P_i$ .
  - During constraint reduction, when an analysis step introduces a new constraint for a key-derivation, then this constraint is labeled by the protocol from which the analyzed message stems. Note that, for instance, in a  $P_1$  constraint  $M \vdash m$ , a message  $t \in M$  may be of type  $P_2$ ; thus analyzing  $t$  would produce a  $P_2$ -type constraint. Note also that we may reach a key derivation constraint  $M \vdash k$  where  $k$  is a secret constant; such a constraint cannot be satisfiable since we already assumed secret long-term constants are never transported, so the intruder cannot obtain them.
  - We will also assign a third kind of label *special* to some constraints during the reduction: when we apply the generate rule to a  $P_2$  term, we obtain constraints for its immediate subterms; these subterms can be of type  $P_1$ ; the resulting constraints are then labeled *special*. As we will see below, distinguishing these special constraints from other  $P_2$  constraints makes the formulation of the invariants in the sufficiency proof easier.

- (Ἐτέοκλος):
  - Payload type messages in  $P_1$  constraints can also occur toplevel now.
  - (Labeling when unifying messages.) If we unify a term  $m_1$  that is labeled  $\text{spayload}(A, B)$  with another term  $m_2$  during constraint reduction, then after unification both terms are labeled  $\text{spayload}(A, B)$ . Note that this labeling is always uniquely determined because the terms labeled  $\text{spayload}(A, B)$  must always be ground and if two terms are labeled  $\text{spayload}(A, B)$  and  $\text{spayload}(A', B')$ , respectively, then they can only be unifiable if  $A = A'$  and  $B = B'$ , by the properties we assume (cf. (Πολυνεΐξης)). Further, we define that the labeling  $\text{rpayload}(A, B)$  is dominated by  $\text{spayload}(A, B)$ , i.e., when unifying terms labeled for sending and receiving payloads, then the sending payload label “wins”. We can then reformulate the remark at the end of the condition as follows. This assumption ensures that every subterm  $m$  of a  $P_1$ -term (in the initial constraints) that is of type **Payload** is either a variable or ground, and if ground, then  $m \in \mathcal{M}_{A,B}$  for some uniquely determined  $A$  and  $B$  and this is indicated by the appropriate label  $\text{spayload}(A, B)$ . In the proof, we show that the constraint reduction would equally work by replacing the concrete  $m$  with the abstract  $\text{payload}(A, B)$ .
- (Ἀμφιάρσος): in terms of constraints, we define payload secrecy as follows.
  - Consider a lazy intruder trace for  $P_1$  and let  $\phi_0$  be the corresponding constraints and  $M$  the intruder knowledge after  $\phi_0$ . If the constraint  $\phi_0 \wedge M \vdash \text{payload}(A, B)$  is satisfiable, then this trace must be an attack trace against secrecy in  $P_1$ .  
Thus, if the intruder finds out a secret payload, this fact alone counts as an attack; we will use this in the proof in an indirect way: we may assume that in a given attack, the intruder discovers no secret payload at an intermediate step, because then a shorter attack exists (cutting off all steps after discovering the secret payload).
  - We make a similar requirement for  $P_2$ : let again  $\phi_0$  be the constraint of a lazy intruder trace, this time of  $P_2$ , and let  $M$  be the resulting intruder knowledge. Let  $m \in \mathcal{M}_{A,B}$  be a concrete payload that needs to be secret. If  $\phi_0 \wedge M \vdash m$  is satisfiable, then this trace must be an attack trace against secrecy in  $P_2$ .

### A.3 The proof

We now show the main result of this paper, namely that the conditions are sufficient for static vertical composability.

**Theorem 2.** *Consider two protocols  $P_1$  and  $P_2$  that satisfy all the seven conditions. If there is an attack against  $P_1^{P_2} \parallel P_2$ , then there is an attack against  $P_1$  or against  $P_2$ .*

*Proof.* Consider an attack against  $P_1^{P_2} \parallel P_2$  in form of a satisfiable lazy intruder constraint  $\phi$ . The proof is structured as follows:

- (Shortest Attack) If the intruder discovers a secret payload at an intermediate step of the attack, then there is a simpler attack (cutting off all steps after discovering the secret payload). We thus show that we can assume without loss of generality that no secret payload is derivable in any step of the attack but the final one.
- (Invariants) We show several invariants, including that the seven properties are preserved over all reduction steps.
- (Split into  $P_1^{P_2}$  part and  $P_2$  part) We show that the attack can be split into a pure  $P_1^{P_2}$  part and a pure  $P_2$  part so that both constraints are still satisfiable. At least one of the two is an attack.
- (Abstraction from  $P_1^{P_2}$  to  $P_1$ ) In case the attack is on  $P_1^{P_2}$ , we show how that a corresponding attack exists on the abstract  $P_1$ .

### Shortest Attack

The requirement ( $\text{\AA}\mu\varphi\acute{\alpha}\varphi\alpha\sigma\zeta$ ) allows us to simplify an attack, if the intruder is able to derive a secret payload at an intermediate stage of the attack (since at that point we have an attack already). This simplification is necessary in our proof, because the steps that follow using a particular concrete secret payload cannot be reflected on the abstract payload level of  $P_1$ .

Formally, we will assume that the constraint  $\phi$  can be written as  $\phi = \phi_0 \wedge M \vdash m$  such that from no intruder knowledge of  $\phi_0$ , in no satisfying interpretation of  $\phi$ , a secret payload can be derived. Moreover, if a secret payload can be derived from  $M$ , then  $m$  is one such secret payload. We can also assume that in the derivation of  $m$  from  $M$  we do not use secret payloads as an intermediate reduction step.

This assumption is not a restriction for the following reason. If  $\phi$  does not satisfy this property, then we can find a suitable  $\phi'$  that does: cutting off from  $\phi$  all conjuncts after reaching an intruder knowledge  $M$  from which a secret payload  $m$  can be derived (and so that this derivation does not require to derive another secret payload first). Except for the  $M \vdash m$  conjunct, this is a valid symbolic trace of  $P_1^{P_2} \parallel P_2$  (and  $M$  the reached intruder knowledge). Following the further parts of the proof, we obtain that  $\phi'$  can be split into a pure  $P_1$  or a pure  $P_2$  part. By ( $\text{\AA}\mu\varphi\acute{\alpha}\varphi\alpha\sigma\zeta$ ), therefore the  $M \vdash m$  conjunct is an attack against secrecy either in  $P_1$  or in  $P_2$  (depending on which part is used to derive  $m$  here).

By the soundness of the lazy intruder, the property that no secret payload enters the intruder knowledge remains preserved over all constraint reductions: suppose  $\phi_0$  has no interpretation  $\mathcal{I}$  such that  $\mathcal{I} \models \max(\phi_0) \vdash m'$  for any secret payload  $m'$  and for  $\max(\phi_0)$  the union of all intruder knowledges in  $\phi_0$ , then this holds over all reductions on  $\phi_0$ .

### Invariants

We consider now a reduction of  $\phi$  to a simple constraint, which must be possible because  $\phi$  is satisfiable. As said before, whenever we have a generate step to

a term  $C_P[m]$ , then the resulting constraint is labeled *special*. Then, by our previous assumption,  $m$  cannot be a secret payload. Therefore, either  $m \in M_0$  or  $m \in \mathcal{V}$  by ( $\text{\textcircled{E}}\tau\acute{\epsilon}\omicron\kappa\lambda\omicron\varsigma$ ) and ( $\text{\textcircled{A}}\mu\phi\iota\acute{\alpha}\rho\omicron\varsigma$ ). By the invariants shown below, if this  $m$  is a variable, it can only be instantiated to a ground term in  $M_0$ . Therefore, as soon we have a ground term in a special constraint (as defined in ( $\text{\textcircled{K}}\alpha\pi\alpha\nu\epsilon\acute{\upsilon}\varsigma$ )), we know that it is an element of  $M_0$ . We therefore do not change the meaning of  $\phi$  by just removing that special constraint and forgetting all steps in the reduction that are performed in the original reduction of that constraint. We call this the *deletion of redundant constraints*.

1. (Instantiation of variables and the unify rule)  $P_1$ -typed variables are never instantiated with  $P_2$ -typed terms and vice versa. Moreover, the application of a Unify on constraint  $M \vdash m$  where some  $m' \in M$  is unified with  $m$ , can only occur when  $m$  and  $m'$  are both of type  $P_1$  or both of type  $P_2$  or both public constants.
2. (Purity of the subterms)  $P_2$ -typed terms never have  $P_1$ -typed subterms;  $P_1$ -typed terms can have  $P_2$ -typed subterms only under a  $C_P[\cdot]$  context.
3. (Payload invariants) The properties of ( $\text{\textcircled{E}}\tau\acute{\epsilon}\omicron\kappa\lambda\omicron\varsigma$ ) are preserved, especially points (1) and (2)—the other ones are structural properties or apply to terms of the abstract  $P_1$ .
4. (Derivation) If constraint  $M \vdash m$  is of type  $P_i$  then  $m$  is either of type  $P_i$  or a public constant. If constraint  $M \vdash m$  is of type *special*, then  $m$  is of type  $P_2$  and either ground or a variable. If it is ground, then it is labeled with some  $\text{spayload}(A, B)$  and is a member of  $\mathcal{M}_{A, B}$  getting deleted immediately (by the deletion of redundant constraints). If it is a variable, then it is labeled  $\text{rpayload}(A, B)$ .
5. (Payload Label) In all terms of type  $P_1$ , every occurrence of a ground subterm  $t$  of type Payload is labeled with  $\text{spayload}(A, B)$  for some  $A$  and  $B$ , such that  $t \in \mathcal{M}_{A, B}$ .

We now show that the lazy intruder reduction rules preserve all these invariants.

*Unify Rule* We first show that an application of the unify rule preserves all invariants, which is the most interesting (and difficult) case. We use the variables  $s, t, M, \sigma, \phi$  as in the rule.

- 1 (Instantiation of variables and the unify rule)
  - Note that neither  $s$  nor  $t$  can be variables (by the form of Unify).
  - If  $t$  is a constant, then it can only be a public constant (because the secrecy of secret constants has been ensured statically) by ( $\text{\textcircled{I}}\pi\pi\omicron\mu\acute{\epsilon}\delta\omega\nu$ ). If  $t$  is a public constant, then so must be  $s$ , and the invariant holds.
  - If  $t$  is not a public constant, then it can be only a composed term (and so must be  $s$ ). Both  $s$  and  $t$  are thus labeled as  $P_1$  or  $P_2$ . Now  $s$  and  $t$  must then be labeled either both  $P_1$  or both  $P_2$  as their unifiability would otherwise contradict ( $\text{\textcircled{K}}\alpha\pi\alpha\nu\epsilon\acute{\upsilon}\varsigma$ ).

- If both  $s$  and  $t$  are labeled  $P_2$ , then no subterm is labeled  $P_1$ , so the unification preserves the invariant.
  - If both  $s$  and  $t$  are labeled  $P_1$ , then  $P_2$ -labeled subterms can occur only under  $C_P[\cdot]$ , and no subterms under  $C_P[\cdot]$  are labeled  $P_1$ . Since  $C_P$  cannot be unified with any other non-variable subterm of  $P_1$  (by ( $\text{\textit{E}\textit{t}\textit{e}\textit{o}\textit{k}\textit{l}\textit{o}\textit{s}}$ )), the unifier  $\sigma$  in question cannot relate a  $P_1$ -variable with a  $P_2$ -term or vice versa.
- 2 (Purity of the subterms) This follows from the previous invariant, as  $P_1$  variables are instantiated only with  $P_1$  terms (and similar for  $P_2$ ).
- 3 (Payload invariants) For what concerns the identification of the payload, if  $s$  and  $t$  are  $P_1$ -typed messages, then the previous invariant already preserves the identification property for  $C_P[\cdot]$ . The unification of public constants or of  $P_2$ -messages cannot destroy the property either.
- For what concerns the property of blank-or-concrete payloads, again, if  $s$  and  $t$  are  $P_1$ -typed messages, then the unifier can only unify either payload-typed variables with ground terms of type payload or with other payload-typed variables, so the invariant is preserved.
- The blank-or-concrete payload property only applies to terms under  $C_P[\cdot]$ ; and if it is a variable, then by ( $\text{\textit{E}\textit{t}\textit{e}\textit{o}\textit{k}\textit{l}\textit{o}\textit{s}}$ ), this variable can only occur under the  $C_P[\cdot]$  context or toplevel. Therefore, applying the Unify rule to  $P_2$ -terms can never destroy the blank-or-concrete property since it explicitly forbids unifying terms  $s$  and  $t$  if one of them is a variable.
- 4 (Derivation) and 5 (Payload Label) are immediate.

*Generate Rule* It is almost immediate to see that all applications of the generate rule preserve the invariants. We only need to discuss the case of applying a generate rule to a payload term (which cannot be a variable for the generate rule to be applicable). In this case, we either have a normal  $P_2$ -constraint, and then the generate rule does not create problems, or a constraint of type *special*, in which case the message  $m$  to derive is ground (due to clause 2 of ( $\text{\textit{E}\textit{t}\textit{e}\textit{o}\textit{k}\textit{l}\textit{o}\textit{s}}$ ) and the fact that it cannot be a variable) and moreover  $m \in M_0$  because  $m$  cannot be a secret payload: if it were a secret payload, then a secret payload would be an intermediate step (since special constraints only arise from opening a  $C_P[\cdot]$  context using a generate rule) and this contradicts our assumption that a secret payload cannot be derived at an intermediate point already.

*Analyze Rule* Also in this case we have that almost all the invariants are immediate, and it only remains to be mentioned that when analyzing a context-term, since  $C_P[\cdot]$  must be a concatenation at the top level, there is not going to be a key-derivation constraint in this case. This concludes the proof of the invariants, and we can resume the main thread of the proof.

### Split into $P_1^{P_2}$ part and $P_2$ part

We now show that we can split a reduction of the constraint  $\phi$  into a  $P_1$  part and a  $P_2$  part, and still have valid reductions of the respective part of the constraints.

More formally, let  $\phi_1$  be the  $P_1$  constraints of  $\phi$  and  $\phi_2$  be the  $P_2$  constraints of  $\phi$ , where the constraints may contain both  $P_1$  and  $P_2$  terms on the knowledge side. The case of the “last” conjunct  $M \vdash m$  of  $\phi$  deserves a special treatment: if this  $m$  is a secret payload and the reduction of  $M \vdash m$  uses the analysis of a  $P_1$  term, then  $M \vdash m$  shall be considered as part of  $\phi_1$ ; otherwise it is considered as part of  $\phi_2$ .

Considering a sequence of reduction steps that turns  $\phi$  into a simple (and satisfiable) constraint, we can split this reduction into a  $\phi_1$ -part and a  $\phi_2$ -part as expected. We thus have a reduction for  $\phi_1$  to a simple constraint and one for  $\phi_2$ , and they work independently of each other. For the full independence of the two parts, it now remains to show that in the reduction of  $\phi_1$  we never need  $P_2$  messages of the intruder knowledge and vice versa.

Consider a conjunct  $M_i \vdash m_i$  of  $\phi_1$  where  $m_i$  is not the secret payload  $m$ . A unify step at this point can only be with a public value of  $M_0 \subseteq M_i$  or a message of  $P_1^{P_2}$ . A generate step, however, may yield a  $P_2$  term to generate and we need to show that we do not need  $P_2$  messages for the remaining constraint reduction. Either the resulting term is a ground term labeled by  $\text{spayload}(A, B)$ , then this must be public and thus in  $M_0$  (because otherwise we have another secrecy violation within the reduction, which we excluded before), or it is a variable of type `Payload`. Again by the payload invariant, this variable can only be instantiated by another payload variable, or by a concrete public payload. Therefore, all those analysis steps in the knowledge  $M_i$  of  $P_2$  messages that can only yield public knowledge or other  $P_2$  messages can safely be omitted, and we can thus do with the  $P_1^{P_2}$  and  $M_0$  part of  $M_i$ .

The case that the reduction in  $\phi_1$  contains an analysis step of a  $P_1^{P_2}$  message yielding the secret payload  $m$  cannot require any  $P_2$  messages either and must be the pre-last step of the reduction anyway, where the last step is a unify.

An even simpler argument holds vice versa, unless of course the goal is the derivation of the secret payload. In this case, analysis steps of  $P_1^{P_2}$  messages may yield a  $P_2$  message, but then it is either the secret payload  $m$ , so this must be part of  $\phi_1$  instead, or it is a public payload that is contained in  $M_0$ . Either way, we thus do not need the  $P_1^{P_2}$  messages.

Therefore, by the property of execution independence ( $\text{T}\delta\epsilon\zeta$ ), there are valid traces of  $P_1^{P_2}$  and  $P_2$  and the corresponding intruder constraints are satisfiable. Moreover, one of the two is an attack trace since either we have a violation of one of the original goals or one of the two is the derivation of a secret payload.

### Abstraction from $P_1^{P_2}$ to $P_1$

In case we have found an attack against  $P_1^{P_2}$ , then by the invariant ( $\text{E}\tau\epsilon\alpha\lambda\omicron\varsigma$ ), we know that every occurrence of a subterm of type `Payload` is either a ground term  $t \in \mathcal{M}_{A,B}$  that is labeled  $\text{spayload}(A, B)$  or a variable labeled  $\text{rpayload}(A, B)$ . Note that having split the attack into  $\phi_1$  and  $\phi_2$ , the entire reduction for  $\phi_1$  works without considering any  $P_2$  terms, except for those public payloads in  $M_0$  and the final secret payload  $m$  (if it exists) that comes out of the last constraint directly as an analysis step. Therefore, all reductions work if we replace all ground

$t$  labeled  $\text{spayload}(A, B)$  by  $\text{payload}(A, B)$  and then form a valid trace of  $P_1$  since the initial knowledge in  $P_1$  contains the abstract label for all public payloads by ( $\text{Παρθενοπαίος}$ ).

**Theorem 3.** *The statement of Theorem 2 also holds when the symbolic traces induced by the application protocol  $P_2$  include negated substitutions.*

*Proof.* We consider here only the changes in the proof of Theorem 2 that are required by this extension.

We extend the invariant 2 to cover also the inequalities: they are all induced by  $P_2$  and thus have initially pure  $P_2$ -terms, and this holds true over all reduction steps, because all unifications that ever occur are those of the Unify rule, which preserves the purity of terms. (The satisfiability check for the inequalities also includes a unification, but this is only performed for checking, we do not compute with the unified inequalities.)

Thus, when the reduction is split into a  $P_1^{P_2}$  and  $P_2$  part, we can associate all the negative substitutions to the  $P_2$  part. Therefore, in the last section of the proof, where we abstract the concrete payloads of  $P_1^{P_2}$  to the abstract ones of  $P_1$ , no negated substitutions need to be considered anymore.

#### A.4 Extensions

Expanding on the discussion in Section 5, where we formalized the extension of our sufficient conditions to the case of more messages for what concerns both the static aspect and the logical aspect of vertical protocol composition, we now discuss in more detail how we can extend our composability result to allow for negative checks also on the channel protocol when considering finer abstractions.

**Negation** Above, we only considered positive constraints; as argued by [5], we can express a much larger class of protocols and goals if we just additionally allow constraints that represent the negation of a substitution, i.e., that are of the form

$$\forall x_1, \dots, x_n. y_1 \neq t_1 \vee \dots \vee y_m \neq t_m, \quad (2)$$

where the variables  $x_i$  are the “free” variables of the  $y_i$  and  $t_i$ , i.e., those that do not occur in any  $M \vdash m$  constraint. Intuitively, this excludes the substitution  $\sigma = [y_1 \mapsto t_1, \dots, y_m \mapsto t_m]$  for any value of the free variables  $x_i$ .

Allowing such negative conditions is essential for more complex protocols; examples include participants who maintain a database of items they have processed, may use negative conditions (e.g. a certain item does not yet occur in the database), or more advanced security goals like in [29]. As shown in [28], the support for negative substitution in the constraint-based approach allows for a large fragment of ASLan—basically everything except the Horn clauses of

ASLan. (To support also the Horn clauses in the symbolic model one additionally needs to allow deduction constraints, which we do not here; we plan to show in a future version that we can support them as well.)

The simple idea to support negative substitutions with the lazy intruder is: once all the  $M \vdash m$  constraints have been simplified (i.e., all such  $m$  are variables) check that the inequalities are still satisfiable. To check that the condition (2) is satisfiable, check whether the unification problem  $\{(y_1, t_1), \dots, (y_m, t_m)\}\sigma$  has a solution if  $\sigma$  substitutes every variable of the  $y_i$  and  $t_i$  *except* the  $x_i$  with a fresh constant. This reflects that the intruder can generate arbitrarily many fresh constants himself and use them for the choices he has. If a unifier for said unification problem exists, then no matter what the intruder chooses for his variables, we can find a substitution of the  $x_i$  so that all the inequalities are violated. Otherwise, the inequalities are satisfiable.

Inspecting the proof of Theorem 2, we see that almost everything is compatible with the addition of negated substitutions. The only problem is in the last part. When we show how a  $P_1^{P_2}$  attack can be reduced to an attack of the pure  $P_1$  protocol, we use abstract interpretation but inequalities on terms are not compatible with this: naturally, we cannot represent all payloads that  $A$  may ever send to  $B$  by one single constant  $\text{payload}(A, B)$  if  $B$  can for instance check that in every session he gets a different payload. But if the channel protocol  $P_1$  does not contain any negative check on payloads, then the abstraction is actually fine, as expressed by the following theorem.

**Theorem 3.** *The statement of Theorem 2 also holds when the symbolic traces induced by the application protocol  $P_2$  include negated substitutions.*

This extension allows us to use at least negative conditions and facts in the application protocol, though not in the channel protocol. In fact, the application protocol may even make negative checks on the payloads (e.g., a replay protection) as long as this does not affect the channel protocol (and so the abstraction of the payload there is sound). As the work on ProVerif and other tools shows (e.g., [6,26]), there is hope for also allowing for negative checks on the channel protocol when considering finer abstractions. We leave this investigation for future work.

## B An Extended Example

We now discuss a larger example that already makes use of the extension of Section 5. As protocol  $P_1$  we use (a slightly simplified version of) unauthenticated TLS composed with a password login to authenticate the client (we later discuss how TLS and login could be decomposed)

$$\begin{aligned}
A &\rightarrow B : [\text{clientHello}, A, NA]_3 \\
B &\rightarrow A : [\text{serverHello}, NB]_2 \\
B &\rightarrow A : \text{sign}(\text{privk}(\text{pk}(ca)), [\text{cert}, B, \text{pubk}(\text{pk}(B))]_3) \\
A &\rightarrow B : \{PMS\}_{\text{pubk}(\text{pk}(B))} \\
K &:= \text{prf}(PMS, NA, NB) \\
K_A &:= \text{clientk}(K) \\
K_B &:= \text{serverk}(K) \\
A &\rightarrow B : \{h(\text{allMsg})\}_{K_A} \\
B &\rightarrow A : \{h(\text{allMsg})\}_{K_B} \\
A &\rightarrow B : \{[\text{login}, A, B, \text{pw}(A, B)]_4\}_{K_A}
\end{aligned}$$

Here, `clientHello`, `serverHello`, `cert`, and `login` are public constants used as tags to identify the meaning of the message. `ca` is the (trusted) certificate authority that issued `B`'s certificate. `prf`, `h`, `clientk` and `serverk` are cryptographic one-way functions used to produce key-material or digests from the exchanged random nonces `PMS` (pre-master secret), `NA` and `NB`. `allMsg` represents the concatenation of all previously exchanged messages.

$K_A$  and  $K_B$  are shared keys used for the different communication directions (from `A` to `B` and from `B` to `A`, respectively). From now on, the client can transmit over the channel payloads to the server as follows:

$$\frac{A \rightarrow B : \{\text{payload}(A, B)\}_{K_A}}{A \bullet \rightarrow \bullet B : \text{payload}(A, B)}$$

Symmetrically, the server uses:

$$\frac{B \rightarrow A : \{\text{payload}(B, A)\}_{K_B}}{B \bullet \rightarrow \bullet A : \text{payload}(A, B)}$$

*An Application.* As a simple application protocol  $P_2$ , we consider the online movie stream purchase

$$\begin{aligned}
A &\bullet \rightarrow \bullet B : [\text{orderMovie}, \text{MovieID}]_2 \\
B &\bullet \rightarrow \bullet A : [\text{askConfirm}, \text{MovieID}, \text{Price}, \text{Date}, \text{StreamID}]_5 \\
A &\bullet \rightarrow \bullet B : [\text{confirm}, \text{MovieID}, \text{Price}, \text{Date}, \text{StreamID}]_5
\end{aligned}$$

One problem with the given  $P_1$  and  $P_2$  is that TLS does not have the required form of contexts for the message transmission, since we have either  $\{\cdot\}_{\text{clientk}(K_0)}$  or  $\{\cdot\}_{\text{serverk}(K_0)}$  (recall that for different channel types like  $A \bullet \rightarrow \bullet B$  vs.  $B \bullet \rightarrow \bullet A$  we need two different contexts). In contrast, our paper required a context that is a concatenation. A slight modification of the protocol would satisfy this assumption, namely  $\{[\text{clientTag}, \cdot]_2\}_{K_0}$  for the client sending messages and  $\{[\text{serverTag}, \cdot]_2\}_{K_0}$  for the server sending messages. (Here  $C_P^C[\cdot] = [\text{clientTag}, \cdot]_2$  and  $C_P^S[\cdot] = [\text{serverTag}, \cdot]_2$ .)

In other words, our compositionality result requires that the channel-identifying context needs to be separated from protection mechanisms, whereas TLS merges these two aspects by using different keys for the direction. Although we could have formulated our compositionality result in a different way and thereby supported exactly the form of TLS, we chose not to do so to keep the arguments

reasonably simple: otherwise we would have had to deal in the proofs with the question of whether the intruder can compose/decompose the context of the message. We think that it is therefore more economical (i) to keep the context simply a concatenation in our compositionality result, (ii) to show that a small variant of TLS would satisfy the conditions, and (iii) to then see in a separate step if this slight modification makes a difference. For TLS, it is obviously not the case, because  $\{\{clientTag, \cdot\}_2\}_{K_0}$  can be composed or decomposed by the intruder, if and only if this is the case for  $\{\cdot\}_{clientk(K_0)}$  (and similarly for the serverkey).

It is straightforward that the message formats of  $P_1$  and  $P_2$  satisfy the condition (Κατανεύς), thanks to the tagging. (And it is not difficult at all to compute  $MP(P_i)$ ,  $SMP(P_i)$ , and  $\mathcal{M}_{A,B}$ .)

*Payload Messages of the Movie Purchase.* In contrast to the running example, we have here the more difficult case of an application protocol where payloads can contain “foreign” input:  $A$  sends a *MovieID* of a movie it wants to see to  $B$ , and  $B$ ’s answer contains that *MovieID*. Modeling this as an arbitrary nonce, we get into the problem described after Definition 6: we cannot bound the set of possible messages here (depending on the other protocols the intruder may be learning messages from) and taking  $\mathcal{T}_\Sigma$  as the domain for the nonces is not compatible with our sufficient conditions. In this case, the solution is straightforward: it is realistic to assume that the set of movie IDs is a fixed set of public identifiers (the ones that the movie portal is offering for purchase on its website) and an honest  $A$  would only order a movie from that set; even for a request from a dishonest  $A$ , an honest server  $B$  would only answer with the second step, if the received movie ID is a legal one. So, for the first and second payload message of the application protocol, we can give very clear bounds that are realistic in our scenario.

The third message is more difficult, because here the user replies with whatever he or she has received from the server as *Price*, *Date*, and *StreamID*. At least for *Price* and *Date* we can again make the restriction to well-defined a priori fixed sets of public constants (because the user  $A$  can check the “well-formedness” of price and date and not answer to an unreasonable input).

It is however more tricky to deal with *StreamID* here, because a dishonest server  $B$  may indeed have sent any term (not necessarily a pure  $P_2$ -term). We again have two possibilities. The first is to assume that the server  $B$  is honest in all runs and chooses the stream IDs from a dedicated set  $\mathcal{I}_{B,A}$  of constants again (that are only used in  $P_2$  as stream IDs from  $B$  for  $A$ ). This distinction by name is necessary since the intruder will get all payloads that are meant for him as a recipient into the initial intruder knowledge by (Ἀμφιῶραος), and he should not know the streamIDs for other honest agents to prevent mistakes here.

A way to also allow at this point a dishonest  $B$  without the final payload message from  $A$  containing arbitrary input as *StreamID* is to go for a typed model where the intruder behavior is already restricted to well-formed messages (and this typed model can be justified again by disjointness conditions [28] so

that the intruder can never produce at this point a *StreamID* that is not a  $P_2$ -constant). We leave a general investigation of such concepts for future work.

*Separating the password step.* As described in [29], we can regard the pure unilaterally authenticated TLS *without* the login as a protocol that provides a *secure pseudonymous channel*, i.e., a secure channel where the identity of the client  $A$  is not authenticated (yet), while no other agent can read or modify messages on that channel. We write  $[A] \bullet \rightarrow \bullet B$  and  $B \bullet \rightarrow \bullet [A]$  to denote this kind of channel. In principle, we could thus see the login protocol as a separate protocol that builds on such channels, turning secure pseudonymous channels into standard secure ones. The login  $P'_1$  would thus be:

$$[A] \bullet \rightarrow \bullet B : [\text{login}, A, B, pw(A, B)]_4$$

and from there on the client can send messages as follows:

$$\frac{[A] \bullet \rightarrow \bullet B : \text{payload}(A, B)}{A \bullet \rightarrow \bullet B : \text{payload}(A, B)}$$

and similarly for the server in the other direction. Hence, let  $P_0$  be TLS without client authentication and without login; this protocol gives us the channel pair  $[A] \bullet \rightarrow \bullet B$  and  $B \bullet \rightarrow \bullet [A]$ . The login protocol  $P'_1$  can turn such channels into standard secure channels, i.e.,  $A \bullet \rightarrow \bullet B$  and  $B \bullet \rightarrow \bullet A$ . The vertical composition of  $P'_1[P_0]$  (i.e., TLS plus login) then gives exactly the protocol  $P_1$ .

There is, however, a slight problem with this again when we think of the abstract payload in  $P_0$ : it cannot be simply  $\text{payload}(A, B)$  because  $A$  is not authenticated—if we tried to verify  $P_0$  like that we would get trivial authentication flaws. As argued in [29], we should in this case use a pseudonym as the endpoint of the channel. In this case, it could be the pre-master secret  $PMS$  that  $A$  has created, and then we require that  $\text{payload}(PMS, B)$  is known to the intruder iff  $PMS$  is *dishonest*—defining  $PMS$  to be *honest* iff it was created by an honest client  $A$  for use with an honest server  $B$ . The fact that now the payload abstraction depends on a freshly created nonce may seem counter-intuitive, but as far as we can see there are no formal problems attached to this. In fact, similar abstraction techniques have been used in ProVerif and other tools [6,26].