Sufficient Conditions for Vertical Composition of Security Protocols (Extended Version)*

IMM-Technical Report-2011-18

Sebastian Mödersheim
DTU Compute
Lyngby, Denmark
smo@imm.dtu.dk

Luca Viganò
Department of Informatics
King's College London, UK
luca.vigano@kcl.ac.uk

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 well known that protocol composition can lead to attacks even when the individual protocols are all secure in isolation. In this paper, we formalize seven easy-to-check static conditions that support a large class of channels and applications and that we prove to be sufficient for vertical security 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.
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[1] 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, 13, 17, 18]), 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

*The work presented in this paper was partially supported by the EU FP7 Projects no. 318424, "FutureID: Shaping the Future of Electronic Identity" (futureid.eu), and no. 257876, "SPaCIoS: Secure Provision and Consumption in the Internet of Services", and by the PRIN 2010-2011 Project "Security Horizons". Much of this work was carried out while Luca Viganò was at the Dipartimento di Informatica, Università di Verona, Italy.

[1]In this paper, we use the terms *compositionality* and *composability* interchangeably.

parallel composition that we refer to as *horizontal* compositions. Maurer et al. [23] (but see also the more recent [21, 22]) 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. Another example is Diffie-Hellman that allows two agents A and B to establish secure channels with each other, provided that they already have authentic channels; how exactly these channels are realized does not matter for Diffie-Hellman.

Mödersheim and Viganò [30] use the concepts that Maurer et al. had presented informally in [23] and give a formal definition in a transition-system model for security protocols. In particular, channels can be both *assumptions* of a protocol (e.g., the authentic channels in Diffie-Hellman) and *goals* of a protocol (e.g., that Diffie-Hellman establishes secure channels).[2] 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), is it then possible to derive that their vertical composition $P_2[P_1]$ (e.g., a banking service over TLS) is secure?

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

1. Mödersheim and Viganò [30] considered what we call the *logical aspect:* a mismatch between the behavior of a channel as an assumption and as a goal.

2. But there is also a *static aspect:* an interference between the message formats (when message parts of P_1 could be confused with message parts of P_2).

Mödersheim and Viganò [30] prove only the logical aspect of the compositionality question for their notion of channels, while for the static aspect they simply *assume* that the composed protocols do not interfere with each other. In fact, this assumption is a *semantical* condition that involves the set of all concrete runs of the composed protocols—as opposed to a simple syntactic check on the protocols. They suggest that this could maybe be solved similar to existing disjointness notions in parallel protocol composition, but leave this complex problem open.

Contributions.

This open problem of how to deal with the static aspect of vertical protocol composition turns 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.[3] 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" all possible payloads of a protocol P_2 are inserted into P_1 and this serves as an interface to the results of [30].

- 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 [30] 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 [30] 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, 15, 16] 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 [14], which also considers vertical protocol composition. The difference is that [14] supports only one particular kind of channel protocol,

[2] In fact, later papers [21, 22] have more formal notions of channels as well, but very remote from transition systems.

[3] 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).

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 [30]) 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, [14] 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 [21, 22, 23] 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 [19]. A detailed comparison is difficult here because the UC-works, and similarly the other cryptographic approaches, are rooted in the cryptographic world based on the indistinguishability of an ideal and a real system, while our approach is based on trace-based protocol semantics and properties (and treats cryptography as black boxes). Works are emerging that bridge the gap between the cryptographic and the symbolic world, but this is beyond the scope of our paper. Rather, we are interested in obtaining results that can be immediately applied with the established protocol verification approaches and we formalize sufficient conditions that are both easy to check and satisfied by many protocols in practice.

Organization.

Section 2 contains some formal preliminaries. To ease the reading, we first (Section 3) discuss our compositionality theorem that shows that the seven conditions that we formalize in this paper are sufficient for static vertical composition and then (Section 4) describe the conditions and illustrate the role that they play in the theorem. In Section 5, we discuss extensions of our compositionality result, and we then conclude in Section 6. In the appendix, we give the full proof of our compositionality result and consider a concrete and large 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 Σ be a countable signature and $\mathcal{V}$ be a countable set of variable symbols disjoint from Σ. The signature is partitioned into the set Σ^0 of constants and the set Σ_p of "public" operations. As a convention, we denote variables with upper-case letters and constants with lower-case letters. We use standard notions about terms such as *ground* (without variables), *atomic*, etc. 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 Σ_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 a public or private key k; $\{\!|m|\!\}_k$ represents the symmetric encryption of m with symmetric key k (we assume that this primitive includes also integrity protection such as a MAC); and $[m_1, \ldots, m_n]_n$, for every $n \geq 2$, represents the concatenation of n messages $m_1, \ldots, m_n$ (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 Σ_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, e.g., $\mathsf{pubk}(s)$ and $\mathsf{privk}(s)$ represent the public and private keys resulting from a seed s. Moreover, $\mathsf{pk}(A)$ may represent the seed for the public key of agent A to model a fixed public-key infrastructure. Most importantly, we use the mapping $\mathsf{payload}(A, B)$ to denote an *abstract* payload message that A wants to send to B (we will make precise the details of payload messages below). Formally, these mappings are injective functions on Σ^0 (rather than function symbols of the term algebra). As a consequence, the expression $\mathsf{payload}(A, B)$ represents a constant of Σ^0, and is thus regarded as atomic.

2.2 Transition system (ASLan)

For concreteness, as a formal protocol specification language, we use here the *AVANTSSAR Specification Language ASLan* [4, 3] but all our results carry over to other protocol formalisms such as strands, the applied π 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 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 *state-transition system*, where states are sets of *facts* (i.e., predicates that express something that holds true in a give state) separated by dots (“.”). These facts model the state of honest agents, the knowledge of the intruder, communication channels, and goal-relevant information. 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 Figure 1 below)

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

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. 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:

$$\begin{array}{l}\mathsf{state}_{\mathcal{A},P_1}(A, \mathit{step1}, \mathit{SID}, B) \;\Rightarrow\\ \mathsf{state}_{\mathcal{A},P_1}(A, \mathit{step2}, \mathit{SID}, B).\\ \mathsf{iknows}(\{\{[\mathsf{p}, A, B, \mathsf{payload}(A, B)]_4\}_{\mathsf{privk}(\mathsf{pk}(A))}\}_{\mathsf{pubk}(\mathsf{pk}(B))})\end{array}$$

where $\mathsf{state}_{\mathcal{A},P_1}(\ldots)$ formalizes the local state of an honest agent in role $\mathcal{A}$ of protocol P_1. 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. 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, as formalized by the fact $\mathsf{iknows}(\cdot)$. The local state of an honest agent does not necessarily carry all the knowledge of the agent (like $\mathsf{payload}(A, B)$) 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 $\mathsf{iknows}(\cdot)$ 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 $\mathsf{dishonest}(\cdot)$ that holds true for every agent under the intruder's control (from the initial state on), and the predicate $\mathsf{honest}(\cdot)$ that holds for all other 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 $\mathsf{secret}(M, \{A, B\})$ 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 $\mathsf{secret}(M, \{A, B\})$ holds for two honest agents A and B, and the intruder has learned the message, then we can derive the fact attack:

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

Definition 1. (Secure protocol) We say that a protocol is *secure* when no attack state is reachable.

3. CHANNELS AND COMPOSITION

The most common type of security 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. Similarly, in ASLan we will simply consider the union of the rules of the two protocols (as well as the unions of initial states and goal rules).

There is, however, a subtlety about this in ASLan due to its expressiveness. Recall that in the previous example of a transition rule in ASLan we have noted explicitly the protocol in the name of the state fact. If we did instead use the same fact $\mathsf{state}_{\mathcal{A}}$ in several protocols and build the union, then we might obtain executions that do not make much sense. So, in general, we assume that the state facts of different protocols are disjoint to avoid this kind of collisions. For other facts it can, however, make sense to use the same predicate in several protocols. Obviously, $\mathsf{iknows}(\cdot)$ and attack shall be shared by protocols, but one may also formalize a database of an agent A as the set of messages *msg* for which a fact $\mathsf{db}(A, msg)$ holds. This database can then be "shared" across different protocols that A participates in. As this makes composition much more difficult, we will exclude this by assuming the following notion of protocol independence:

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

Execution independence is neither necessary nor sufficient for parallel compositionality (or other composition types), but it only simplifies the problem: we reduce ourselves to protocols that can interfere with each other only in terms of exchanged messages.

Three further remarks are in order. First, note that execution independence does not exclude protocols where agents for instance maintain protocol-specific databases over several protocol sessions, it only excludes that a database can be shared over several protocols. Second, note that for security goals we also have protocol-specific facts, e.g., $\mathsf{secret}_{\mathsf{P}_1}$, but that is not a restriction and even helps to identify in which protocol the goals were violated. Finally, note that the definition of execution independence is trivial for strands and the applied π calculus: they cannot express dependent protocols in this sense.

Definition 3. (Parallel Composition $P_1 \parallel P_2$ and Composability) For two execution independent protocols P_1 and P_2 in ASLan, we define their *parallel composition* as the union of the initial states, transition rules, and goal rules, respectively. We denote the resulting ASLan specification with $P_1 \parallel P_2$.

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.

A similar definition can be given for the composition of Alice-and-Bob-style protocols modulo their translation to ASLan.

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 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.

3.1 Channels as Assumptions, Channels as Goals

Channels may be used 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:

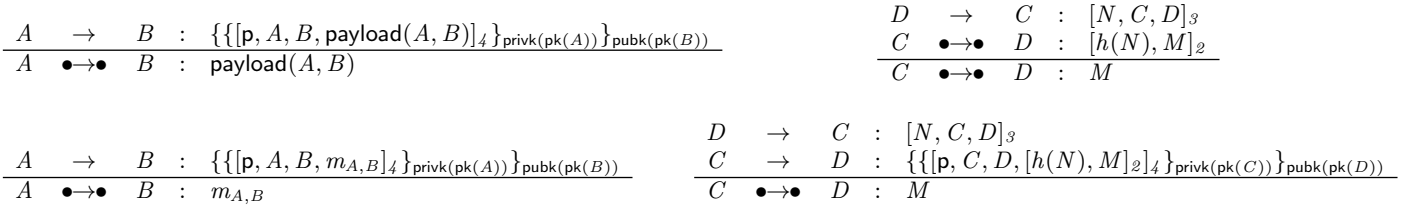

Figure 1: Abstract channel protocol P_1 (top left) and application protocol P_2 (top right), their static vertical composition $P_1^{P_2}$ (bottom left) and their vertical composition $P_2[P_1]$ (bottom right).

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

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.

In [30], two definitions are given 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).

[30] also gives 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: 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 [30] tackles what we call the logical aspect of this question, we look in this paper at what we call the static aspect, i.e., the potential 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 [26, 30], which can be automatically translated to ASLan so that we can rejoice with our ASLan formalization of protocols. It is possible to give 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 4. (Abstract Channel Protocol) Let $\kappa(A, B)$ range over a set of defined channel types (e.g., $A \bullet\!\!\to B : M$) and let $\mathsf{payload}(A, B)$ be a mapping from pairs of agents to (abstract) payloads. An *abstract channel protocol* for $\kappa(A, B)$ is an AnB specification that has $\kappa(A, B) : \mathsf{payload}(A, B)$ as a goal. Moreover, we require that every agent A initially knows $\mathsf{payload}(A, B)$ for every communication partner B (recall the notion of knowledge given in Section 2).

An example of an abstract channel protocol is P_1 in Figure 1. The message exchange was already explained in Section 2, and we declare the goal to be the transmission of the abstract payload over a secure channel.

Definition 5. (Application Protocol) An *application protocol* for channel $\kappa(A, B)$ is an AnB specification that contains as part of its message exchange one step $\kappa(A, B) : t$ with some message term t.

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

3.2 Vertical Protocol Composition

Definition 6. (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 $\kappa(C, D)$.[4] The *vertical composition* $P_2[P_1]$ is defined by replacing in P_2 the step $\kappa(C, D) : t$ with the entire protocol P_1 under the replacement $[A \mapsto C, B \mapsto D, \mathsf{payload}(A, B) \mapsto t]$.

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

As already mentioned, we separate the vertical composition question into a logical aspect (that is already handled in [30]) 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 7. Let P_2 be an application protocol and let $\kappa(C, D) : t$ be the step that uses an abstract channel $\kappa(C, D)$

[4]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.

to transmit message t. Consider the set of all ground terms t_0 that are instance of t in any run of P_2 for a fixed pair (C, D) of agents. We define $\mathcal{M}_{C,D}$ to be an *arbitrary* superset of this set of payload messages.

For the example protocol P_2 from Figure 1, the payload messages sent by an honest agent C has always the form $[h(N), M]_2$, where N is any message that C has received in the first step (supposedly from D) and M is a fresh nonce. We can bound the values that are possible for M since they are freshly created by C; let us say we have a distinguished subset $M_{C,D}$ of all constants (for each pair C and D of agents), from which these are taken. Then the set of all payload messages that can ever occur here are a subset of $\mathcal{M}_{C,D} = \{[h(N), M]_2 \mid N \in \mathcal{T}_\Sigma \wedge M \in M_{C,D}\}$. Note that it is difficult to bound the set of values that variable N can take: it is (potentially) under the control of the intruder and basically depends on what he knows at the time, so we take the largest possible choice, namely, the set $\mathcal{T}_\Sigma$ of all ground terms. (In fact, for dishonest C we usually have to set $\mathcal{M}_{C,D} = \mathcal{T}_\Sigma$ as the intruder can send any term from his knowledge; the case of dishonest C is however uncritical for the rest.)

This example shows why we do not require $\mathcal{M}_{C,D}$ to be *exactly* the set of all payload messages that can occur, but allow *any* superset. This over-approximation is typical of static analysis and, as a rule of thumb, very large, coarse over-approximations make computations easier, but also increase the risk of false positives, i.e., protocols that do not satisfy our sufficient conditions even though they are composable. In our case, a course over-approximation can lead to false positives, since we will later assume that the intruder gets all payloads $\mathcal{M}_{C,D}$ whenever the receiver D is dishonest or secrecy of the payload is not a goal (and C is honest). Suppose we had in the example protocol instead of the hash $h(N)$ directly the nonce N; this would mean that the intruder knows $[N, \cdot]_2$ for every $N \in \mathcal{T}_\Sigma$, i.e., then with $\mathcal{M}_{C,i}$ he initially knows *every* term, trivially giving us false positives. In such a case, we would need to resort to a full-fledged static-analysis approach—in this example, intuitively, using the fact that the intruder learns only nonces N that he himself sent earlier. As the entire approach is already quite complex, we chose not to consider this complication in this paper.

Definition 8. (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 $\mathsf{payload}(A, B)$ when it is first sent by A with a non-deterministically chosen element of $\mathcal{M}_{C,D}$.

An example is given in Figure 1, where we write $m_{A,B}$ to denote an arbitrary message, non-deterministically chosen, from the set $\mathcal{M}_{A,B}$.

The protocol $P_1^{P_2}$ represents a certain "concretization" of P_1 with "random" payload messages from P_2.[5] This notion is valuable because it indeed allows us to divide the composition problem into two aspects, a logical and a static one:

[5] [30] instead uses the notion P_1^*, which, however, may be confusing here as it does not denote the protocol from which the payloads come.

Definition 9. (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.

These notions are used in [30] to show the following compositionality result:

THEOREM 1 ([30]). *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.*

In this paper, 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 9.

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 [30] and the sufficient conditions of this paper are satisfied for P_1 and P_2, then we can combine Theorem 1 with our Theorem 2 (given below) to infer that $P_2[P_1]$ is secure.

3.3 The Static Aspect of Vertical Protocol Composition

We are now ready for our main result, namely that the seven conditions that we will define in Section 4 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 defined in Section 4. 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, and then illustrate the proof by means of a detailed example, which also provides further motivation for the conditions themselves, which we will formalize in the next section.

PROOF SKETCH. In the proof, we employ the constraint reduction technique that we refer to as the *lazy intruder* (see, e.g., [5, 8, 24, 33]). 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 uninstantiated. We use intruder *constraints* of the form

$$IK \vdash t$$

where t is a (symbolic) term that an agent is able to receive and the set IK of messages 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. This is because the lazy intruder technique never instantiates variables whose concrete value is irrelevant for the attack (this is why 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. □

3.4 Illustration

We illustrate the proof at hand of a concrete example. Let us turn again to the example protocols P_1, P_2, $P_1^{P_2}$ and $P_2[P_1]$ from Figure 1, and let us now consider a slight variant of the protocol P_1 in which we deliberately insert an authentication vulnerability. This allows us to illustrate the different steps in the construction of the proof—that an attack against $P_2[P_1]$ can be reduced to an attack against either P_1 or P_2. Moreover, it also helps to illustrate and motivate the conditions that we introduce below.

Note that the message in P_1 mentions both the sender and the intended receiver as part of the signature. Let us now consider a variant without these two names:

$$\begin{array}{llll} A & \to & B & : \{\{[\mathsf{p}, \mathsf{payload}(A,B)]_2\}_{\mathsf{privk}(\mathsf{pk}(A))}\}_{\mathsf{pubk}(\mathsf{pk}(B))} \\ \hline A & \bullet\!\!\to\!\!\bullet & B & : \mathsf{payload}(A,B) \end{array}$$

that also gives accordingly the following variants of the compositions $P_1^{P_2}$ and $P_2[P_1]$:

$$\begin{array}{llll} A & \to & B & : \{\{[\mathsf{p}, m_{A,B}]_2\}_{\mathsf{privk}(\mathsf{pk}(A))}\}_{\mathsf{pubk}(\mathsf{pk}(B))} \\ \hline A & \bullet\!\!\to\!\!\bullet & B & : m_{A,B} \end{array}$$

$$\begin{array}{llll} D & \to & C & : [N, C, D]_3 \\ C & \to & D & : \{\{[\mathsf{p}, [h(N), M]_2]_2\}_{\mathsf{privk}(\mathsf{pk}(C))}\}_{\mathsf{pubk}(\mathsf{pk}(D))} \\ \hline C & \bullet\!\!\to\!\!\bullet & D & : M \end{array}$$

As we already remarked above, it turns out that, for our proof, a symbolic representation of traces is very useful to make the arguments about intruder actions concise. This representation is often used in model checking, sometimes briefly referred to as the lazy intruder technique, but our results are of course independent of any such technology. A *symbolic trace* of a protocol consists of a sequence of send and receive actions of the honest agents, in which we have variables for each subterm of a received message where the agent is willing to accept an arbitrary value; this variable can then occur in subsequent sending actions. Although we do not note it explicitly here, every sent message is directly added to the intruder knowledge, and every received message must be constructed by the intruder. For instance, the following is one symbolic trace for the protocol $P_2[P_1]$:

d sends $m_1 = [n, c, d]_3$
c receives $m_1' = [N, c, D]_3$
c sends $m_2' = \{\{[\mathsf{p}, [h(N), m]_2]_2\}_{\mathsf{privk}(\mathsf{pk}(c))}\}_{\mathsf{pubk}(\mathsf{pk}(D))}$
d receives $m_2 = \{\{[\mathsf{p}, [h(n), M]_2]_2\}_{\mathsf{privk}(\mathsf{pk}(c))}\}_{\mathsf{pubk}(\mathsf{pk}(d))}$

where the constants n and m are the concrete nonces that d and c created for N and M, respectively. When c receives m_1', every value for nonce N and for the claimed sender D is possible, and the answer m_2' that c sends depends on these two variables. In contrast, when d receives m_2, it must be encrypted for d and contain the nonce n sent earlier. A requirement for such a trace to be feasible is that the intruder can indeed construct all the messages that are received, using what he learned. For this symbolic trace, we thus have the constraints

$$IK_0 \cup \{m_1\} \vdash m_1' \quad \wedge \quad IK_0 \cup \{m_1, m_2'\} \vdash m_2\,,$$

where IK_0 is the set of initially known messages, say, the names of all agents, their public keys, and the intruder's private key $\mathsf{privk}(\mathsf{pk}(i))$. Note that the symbolic representation with the constraints corresponds exactly to the set of *ground* traces that are possible.

The core of the lazy intruder is a constraint reduction technique to find (a finite representation of) all solutions of the constraints. In this case, we have, for instance, the solution $D = d$, $N = n$, $M = m$ that corresponds to one normal execution of the protocol between two honest agents c and d with the intruder as a "network node" simply forwarding all messages. We can, however, also express that the symbolic trace violates authentication if $M \neq m$, i.e., convincing d that c has sent a message M that c never sent, or if $D \neq d$, i.e., convincing d that c has sent a message while c actually meant to send it to somebody else.[6] There is indeed such a solution: $D = i$, $M = m$, and $N = n$. This means that the intruder started a session with c, playing under his real name in role D and using the nonce n from the other session with the honest d. The intruder can then decrypt the message m_2' (since it is encrypted with his public key) and re-encrypt it with the public key of d, completing the attack.

The logical part of the composition problem (i.e., what is proved in [30]) shows that such an attack on $P_2[P_1]$ can be reduced to one on $P_1^{P_2} \parallel P_2$. For our example, the transformed symbolic attack would look like this (annotating for each step which of the protocols it belongs to):

P_2 : d sends $m_1 = [n, c, d]_3$
P_2 : c receives $m_1' = [N, c, D]_3$
P_2 : c sends on a secure channel $c \bullet\!\!\to\!\!\bullet\, D$: $[h(N), m]_2$
$P_1^{P_2}$: c sends $m_2' = \{\{[\mathsf{p}, [h(N), m]_2]_2\}_{\mathsf{privk}(\mathsf{pk}(c))}\}_{\mathsf{pubk}(\mathsf{pk}(D))}$
$P_1^{P_2}$: d receives $m_2 = \{\{[\mathsf{p}, X]_2\}_{\mathsf{privk}(\mathsf{pk}(c))}\}_{\mathsf{pubk}(\mathsf{pk}(d))}$
P_2 : D receives on a secure channel $c \bullet\!\!\to\!\!\bullet\, D$: $[h(N), m]_2$

Here, the abstract channel of P_2 runs in parallel with a corresponding step from $P_1^{P_2}$ with exactly the same payload message (which is possible since in $P_1^{P_2}$ the agent c nondeterministically picks a message from the set of all payload messages $\mathcal{M}_{c,D}$). Also note that d receiving m_2 in $P_1^{P_2}$ does not require a particular form of payload anymore (in contrast to the message m_2 in the $P_2[P_1]$ trace).

[6] One may argue that this is not really an authentication problem, but we do not want to debate authentication goals here and use the definition from [30].

The intruder deduction constraints for the received messages are again the same (modulo the said change of m_2). The requirement for the attack is also similar: $D \neq d$ or $X \neq [h(N), m]_2$, and we have again the attack for the solution $D = i$ (and $X = [h(N), m]_2$ with N arbitrary).

Now the goal of this paper to show that such a $P_1^{P_2} \parallel P_2$ attack can indeed be reduced to an attack against P_1 in isolation or against P_2 in isolation.

To that end, we look at how the lazy intruder technique would check the satisfiability of the constraints

$$IK_0 \cup \{m_1\} \vdash m_1' \ \wedge \ IK_0 \cup \{m_1, m_2'\} \vdash m_2 \ \wedge \ (D \neq i \vee M \neq m)\,.$$

The point for using the lazy intruder here is that the technique is complete, i.e., if the constraints have a solution, then the reduction rules find a solution, and that the *laziness* precludes making any instantiations of variables that are not required for solving the constraints.

One reduction rule is *unification*: for constraint $IK \vdash t_1$, if there is a term $t_2 \in IK$ that is unifiable with t_1 under the most general unifier σ, then we can solve this constraint and apply σ to all remaining constraints. The key insight is now that unification cannot be applied to two terms t_1 and t_2 that are variables—because we are lazy: we do not make a choice when any value t_1 is fine, and do not analyze any value t_2 that the intruder himself created earlier. One of the conditions below (namely, (Condition 3)) in fact says that the structure of all messages and non-atomic submessages in the protocols P_1 and P_2 must be sufficiently disjoint; roughly speaking, P_1 stuff cannot be unified with P_2 stuff. The exclusion of atomic submessages (i.e., variables and constants) is necessary because a random nonce does not indicate whether it "belongs" to P_1 or P_2. Thus, in any unification step of the lazy intruder, t_1 and t_2 can only be unifiable if they belong to the same protocol. In the example, neither m_1 nor m_1' can be unified with m_2 or m_2'.

Other intruder operations are *analysis* of messages in the knowledge (e.g., in the example the intruder can decrypt m_2'), as well as *generating* terms on the sender side (in the example, the intruder can generate m_2 by encrypting with $\mathsf{pubk}(\mathsf{pk}(d))$ the message obtained from decrypting m_2'). More generally, this allows us to show that each constraint for generating a $P_1^{P_2}$ message can be solved without using any P_2 messages in the knowledge and vice versa. Thus, we can reduce it to a problem of "pure" constraints that contain only messages from one protocol each. That, however, by the notion of protocol independence (cf. Definition 8, required below in (Condition 1)) means that we can simply split the constraints into a $P_1^{P_2}$ part and a P_2 part, and they represent an execution in isolation of $P_1^{P_2}$ and of P_2, respectively. Moreover, one of the two executions is then an attack against the respective protocol. In our case, the two $P_1^{P_2}$ steps of the trace together with the constraint $IK_0 \cup \{m_2'\} \vdash m_2$ alone entail an attack against $P_1^{P_2}$: the attack is the solution $D = i$ and $X = [h(n), m]_2$.

What remains to show is that this implies also an attack on P_1. This follows merely by replacing the payload $[h(n), m]_2$ in m_2' with the abstract payload $\mathsf{payload}(c, D)$. In fact, as part of the conditions below we will label every payload sent by an honest agent with this abstract payload (reflecting the intentions of the agent). Similarly, the receiver-side payload X is labeled with d's expectations of a payload from C for d. The solution $C = c$ and $D = i$ is the authentication attack. This concludes the illustration of the proof.

We thus solved the static vertical composition question as it was left open in [30]. 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.

4. THE CONDITIONS

We now finally present our seven conditions on a pair of protocols that are sufficient for the vertical composition result (cf. Theorem 2); actually, in some cases, the conditions are sets of related sub-conditions. For each condition, we also highlight the specific role it plays in the proof.

Structural Properties (Condition 1)

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 4 and 5, so that the compositions $P_2[P_1]$ and $P_1^{P_2}$ (Definitions 6 and 8) are actually defined. Let also $\mathcal{M}_{A,B}$ be defined with respect to P_2 according to Definition 8. We further assume that P_1 and P_2 are execution independent (Definition 2).

The execution independence is used in the "splitting step" of the proof, where we have an execution of $P_1^{P_2}$ and P_2 in parallel and where we already know that the intruder does not need to use messages from either protocol to attack the other. Execution independence then allows us to conclude that the sub-traces of the respective protocols are valid traces.

Constants (Condition 2)

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.
- $\mathcal{F}$ 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 $\mathsf{payload}(A, B)$). These may only occur in 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.

The partitioning of the constants plays a role in proving that for each message that the intruder has to produce for protocol P_2, he needs only (composed) P_2 messages, public constants, and fresh constants from $\mathcal{F}_2$. (And a similar property holds for $P_1^{P_2}$.) If, however, the considered attack were to use a secret constant from $\mathcal{S}$, then there would be

a simpler attack already (and we would not need to worry about the construction of further messages with the exposed secret). The abstract payloads $\mathcal{L}$ only come back into the picture in the final step of the proof, when we reduce the $P_1^{P_2}$ attack to a P_1 attack.

Disjointness (Condition 3)

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) since constants (a) may be shared between protocols (e.g., agent names and keys) and (b) by 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 ASLan 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 Figure 1, *MP* and *SMP* are as shown in Figure 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:[7] $MP(P_i) \cap (\mathcal{V} \cup \Sigma^0) = \emptyset$ for $i \in \{1, 2\}$; moreover, non-atomic subterms must be disjoint: $SMP(P_1) \sqcap SMP(P_2) = \emptyset$, 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'd 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 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.

Forbidding atomic messages in *MP* may seem like a restriction, e.g., we cannot send simply a single nonce N as message. However, observe that we only require to put that nonce into a bit of context e.g., $[tag, N]_2$, where *tag* could be a constant identifying the protocol, as it is in practice often done with port numbers.

This condition is used in the proof when we consider how the lazy intruder constraints of a given attack can be solved. The message to construct is an instance of a subterm of *MP*. It is either non-atomic or atomic. If it is non-atomic (i.e., is an instance of a term in *SMP*), then it belongs to a unique protocol P_1 or P_2 and unification is only possible with other *SMP* messages of that protocol that are in the knowledge of the intruder. If it is atomic, then it is either a constant (and handled by arguments provided by the previous condition) or a variable. Note that the variable is thus labeled by the last surrounding context. This is in fact the key where the lazy intruder comes into play: we leave constraints where the term to generate is just a variable, i.e., any term the intruder knows will do. However, during other reduction steps, the variable may get instantiated. In this case, again by this condition, the instantiated term will have the same label as the variable (i.e., whether it belongs to P_1 or P_2). Thus, at the end of the day, we obtain that all P_2 terms can be constructed using only P_2 knowledge and public constants (and similar for $P_1^{P_2}$ terms).

Disjointness of Concrete Payloads (Condition 4)

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 senders, i.e., $\mathcal{M}_{A,B} \cap \mathcal{M}_{A',B'} = \emptyset$ whenever A and A' are honest and ($A \neq A'$ or $B \neq B'$).[8] This does not imply that honest agents or the intruder can recognize from a payload who is (the claimed) A and B.

This condition allows us to unambiguously label every concrete payload with its original sender (even when the message has been forwarded and manipulated by the intruder) and the intended recipient. This labeling makes several constructions in the proof easier as we explain below.

Payload Type and Context (Condition 5)

We now make some provisions about the handling of payload messages in the protocols. We begin with an overview and then give the precise conditions. First, we label those subterms in the message patterns that represent payloads sent or received by honest agents. This is, of course, no restriction but requires that during the translation from AnB to ASLan we need to keep track of which subterms of messages represent payloads, and we thus simply speak of messages of type Payload (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 signalizes 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 $[m_1, \ldots, m_n]_n$ 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 Figure 1 uses the context $C_P(A, B)[\cdot] = [\mathsf{p}, A, B, \cdot]_4$. Note that we here actually allow that contexts may be parameterized over additional information such as the involved agents, but for simplicity of notation we 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

[7] Recall that mappings like $\mathsf{pk}(a)$ map from atoms to atoms and thus $\mathsf{pk}(a)$ also counts as atomic in the sense of this definition.

[8] 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 $\mathcal{X}_{A,B}$.

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

Figure 2: *MP* and *SMP* for the example protocols of Figure 1.

from P_2 and is 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 thus additionally label it $\mathsf{spayload}(A,B)$. The variable case means that an agent receives a payload, and we thus label it $\mathsf{rpayload}(A,B)$ (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 require 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 $\mathsf{Payload}$ 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 $\mathsf{spayload}(A,B)$, or it is a symbolic term (not necessarily a variable) and labeled $\mathsf{rpayload}(A,B)$.

4. ($C_P[\cdot]$ cannot be confused in P_1.) $\{t\} \sqcap \{C_P[s] \mid s \in \mathcal{T}_\Sigma(\mathcal{V})\} = \emptyset$ for any $t \in SMP(P_1)$ where $t \neq C_P[t']$ for any term t'.[9] By (Condition 3), 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 $P_1^{P_2}$ term that is of type $\mathsf{Payload}$ is either a variable or ground. If it is a variable it is labeled $\mathsf{rpayload}(A,B)$, whereas if it is ground, then $m \in \mathcal{M}_{A,B}$ for some uniquely determined A and B and this is indicated by the appropriate label $\mathsf{spayload}(A,B)$. This property is preserved over all constraint reductions in the proof. In particular, we cannot unify payload with non-payload subterms. Note that when we unify a concrete payload with a variable (as respective subterms of a message), the variable will be replaced with the concrete ground payload and will be again labeled with an $\mathsf{spayload}(A,B)$ term. This is even true if the variable was labeled with $\mathsf{rpayload}(A',B')$; this may then be the source of the attack, but observe that when the goal did not include authentication, it may be not be an attack in itself. Finally, the labels give us the argument for the final step of the proof, when we have isolated a $P_1^{P_2}$ attack and want to transform it to a P_1 attack: then we can replace every concrete payload m labeled $\mathsf{spayload}(A,B)$ simply with the abstract $\mathsf{payload}(A,B)$ and obtain a valid P_1 attack.

[9]This is not a consequence of the fact that $t \neq C_P[t']$ for any t' as t may contain variables.

Abstract Payloads (Condition 6)

In the channel protocol P_1 with abstract payload $\mathsf{payload}(A,B)$, we require that the intruder knows initially

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

if the channel-type $\kappa(A,B)$ includes secrecy (i.e., if it is $A \rightarrow\!\bullet\, B$ or $A \bullet\!\rightarrow\!\bullet\, B$, which we can denote by $\mathit{secrecy} \in \kappa(A,B)$); otherwise, the intruder initially knows all payloads $\{\mathsf{payload}(A,B)\}$. 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 $P_1^{P_2}$). This is essential for the soundness of the payload abstraction in the sense that when a concrete payload is known (and this fact is not yet a violation of a secrecy goal), then the intruder knows also the corresponding abstract payload.

This condition is thus used in the last step of the proof when we transform a $P_1^{P_2}$ attack into a P_1 attack and thus replace concrete payloads with abstract payloads: if the attack includes that the intruder can produce a concrete payload m labeled $\mathsf{spayload}(A,B)$, then either we already had earlier a secrecy violation (thus there exists a shorter attack) or it is one of the public payloads, and the intruder thus knows $\mathsf{payload}(A,B)$.

Properties of Concrete Payloads (Condition 7)

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:

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

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 $\mathsf{secret}(M, \{A,B\})$ in every transition where an honest agent sends a payload M labeled $\mathsf{spayload}(A,B)$ or receives a payload M labeled $\mathsf{rpayload}(A,B)$ and using the general attack rule (1).[10]

This condition is, of course, similar to the previous one on abstract payloads. We use it in the proof in two ways: as already explained before, whenever in the attack the intruder uses a payload that is supposed to be secret, we already have a simpler attack to reduce to. Conversely, whenever the intruder uses a payload that is not supposed to be secret, then this is already part of the initial knowledge. This is the closing stone in the proof, because the payloads are the only non-atomic submessages shared between the two

[10]Pedantically, to fulfill the condition of protocol independence, we should use two distinct facts $\mathsf{secret}_{\mathsf{P}_1}$ and $\mathsf{secret}_{\mathsf{P}_2}$ for secrecy in the respective subprotocols.

protocols $P_1^{P_2}$ and P_2, and having these always in the initial knowledge (when not secret) allows us thus to solve $P_1^{P_2}$ constraints using only $P_1^{P_2}$ messages and P_2 constraints only using P_2 messages.

5. EXTENSION TO MORE MESSAGES

We thus solved the static vertical composition question as it was posed in [30]. 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 [30] 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., $P_2[P_2[P_1]]$ 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 $\mathsf{payload}_1(A_1, B_1), \ldots, \mathsf{payload}_k(A_k, B_k)$ 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.

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; see also [13]). 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 [30] 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[11] — 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 [30] 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.

7. 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. Duflot. Bounding messages for free in security protocols. In *Proceedings of FST TCS*, LNCS 4855, pages 376–387. Springer, 2007.

[3] A. Armando, W. Arsac, 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`.

[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.

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

[6] B. Blanchet. From secrecy to authenticity in security protocols. In *Proceedings of SAS*, LNCS 2477, pages 342–359. Springer, 2002.

[7] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS*, 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*, LNCS 2914, pages 124–135. Springer, 2003.

[9] S. Ciobâca and V. Cortier. Protocol composition for arbitrary primitives. In *Proceedings of CSF*, 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*, pages 11 – 23. ACM, 2003.

[12] S. Delaune, S. Kremer, and M. D. Ryan. Composition of password-based protocols. In *Proceedings of CSF*, pages 239–251. IEEE Computer Society Press, 2008.

[13] T. Gibson-Robinson and G. Lowe. Analysing Applications Layered on Unilaterally Authenticating Secure Transport Protocols. In *Proceedings of FAST*, LNCS 7140, pp. 164–181, Springer, 2012.

[14] T. Groß and S. Mödersheim. Vertical protocol composition. In *Proceedings of CSF*, pages 235–250. IEEE CS, 2011.

[15] J. D. Guttman. Authentication tests and disjoint encryption: a design method for security protocols. *Journal of Computer Security*, 3–4(12):409–433, 2004.

[16] J. D. Guttman. Cryptographic protocol composition via the authentication tests. In *Proceedings of FOSSACS*, LNCS 5504, pages 303–317. Springer-Verlag, 2009.

[17] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *Proceedings of CSFW*, pages 24–34, 2000.

[18] A. Kamil and G. Lowe. Understanding Abstractions of Secure Channels. In *Proceedings of FAST*, LNCS 6561, pp. 50–64, Springer, 2011.

[19] R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In *Proceedings of CCS*, pages 41–50. ACM Press, 2011.

[20] 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`.

[21] U. M. Maurer. Constructive cryptography – A new paradigm for security definitions and proofs. In *Proceedings of TOSCA*, LNCS 6993, pages 33–56. Springer, 2011.

[22] U. M. Maurer and R. Renner. Abstract Cryptography. In *Proceedings of ICS*, pages 1–21. Tsinghua University Press, 2011.

[23] U. M. Maurer and P. E. Schmid. A calculus for security bootstrapping in distributed systems. *J. Comp. Sec.*, 4(1):55–80, 1996.

[24] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS*, pages 166–175. ACM Press, 2001.

[25] S. Mödersheim. *Models and Methods for the Automated Analysis of Security Protocols*. PhD Thesis, ETH Zurich, 2007. ETH Dissertation No. 17013.

[26] S. Mödersheim. Algebraic Properties in Alice and Bob Notation. In *Proceedings of Ares*, 2009.

[27] S. Mödersheim. Abstraction by Set-Membership: Verifying Security Protocols and Web Services with Databases. In *Proceedings of CCS*, pages 351–360. ACM Press, 2011.

[28] S. Mödersheim. Diffie-Hellman without Difficulty. In *Proceedings of FAST*. Springer, 2011. Extended version technical report IMM-TR-2011-13, DTU Informatics, 2011.

[29] S. Mödersheim. Deciding Security for a Fragment of ASLan (Extended Version). Technical Report IMM-TR-2012-06, DTU Informatics, 2012.

[30] S. Mödersheim and L. Viganò. Secure pseudonymous channels. In *Proceedings of ESORICS*, LNCS 5789, pages 337–354. Springer, 2009.

[31] 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.

[32] 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.

[33] 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.

APPENDIX

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 $\mathsf{dishonest}(\cdot)$ that holds true for every agent under the intruder's control (from the initial state on), and the predicate $\mathsf{honest}(\cdot)$ that holds for

all other agents.

The intruder can compose terms applying public functions of Σ_p to terms that he knows, and he can decompose terms when he knows the necessary keys. The latter is formalized by a function $ana(\cdot)$ 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:

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

Definition 10. We write $IK \vdash m$ to denote that the intruder can derive the ground message m when knowing the set of ground messages IK. We define $\vdash$ as the least relation that satisfies the following rules:

(D) $IK \vdash m$ for all $m \in IK$,

(G) if $IK \vdash t_1, \ldots, IK \vdash t_n$, then also $IK \vdash f(t_1, \ldots, t_n)$ for all $f \in \Sigma_p^n$,

(A) if $IK \vdash m$ and $(K, P) \in ana(m)$ and $IK \vdash k$ for all $k \in K$, then also $IK \vdash p$ for all $p \in P$.

All terms are interpreted in the free algebra.

Although this definition is given only for ground m and IK, 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 [24, 33, 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, 20, 28].

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.

A.2.1 Symbolic Transition Systems

As defined in [32], 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 $IK \vdash m$, where IK 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 IK.

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 $IK \vdash m$ where IK 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.

A.2.2 Semantics of Constraints

We consider *constraints* that are conjunctions of $IK \vdash m$ statements where both the set of messages IK and the mes-

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

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

where

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

Figure 3: The lazy intruder reduction rules.

sage 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* ϕ:

$$\begin{aligned} \mathcal{I} \models IK \vdash m &\ \text{ iff }\ \mathcal{I}(IK) \vdash \mathcal{I}(m) \\ \mathcal{I} \models \phi \wedge \psi &\ \text{ iff }\ \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \psi \end{aligned}$$

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

A.2.3 Constraint Reduction

The core of the lazy intruder is a set of reduction rules, shown in Figure 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 ϕ (the proof goal), it suffices to show the satisfiability of constraint ϕ' (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 $IK \vdash v$ 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 ϕ any proof can be constructed using the reduction rules that has ϕ 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 Figure 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 σ is such a unifier, we have solved the constraint $IK \vdash t$ and apply σ to the remaining constraint ϕ 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 $IK \vdash t$ 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 IK 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 $f(t_1, \ldots, t_n)$ if f is a public symbol of Σ_P^n and if the intruder can generate all the subterms $t_1, \ldots, t_n$. 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 [28] 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 in-

truder (though they might not be chosen by the intruder himself).

Definition 11. We say that a constraint ϕ is *well-formed* if it has the form (modulo reordering conjuncts)

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

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

As shown in, e.g., [5, 24, 25], the calculus is complete on well-formed constraints.

A.2.4 Lazy Intruder Representation of Attack Traces

As already explained 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 $\mathsf{iknows}(m)$ fact on the left-hand side of an IF rule with any of the infinitely many messages the intruder can generate from his knowledge IK in that state, we do not instantiate any variables in m and simply require the constraint $IK \vdash m$. (Thus rule matching must be replaced with rule unification throughout the trace.) Finally, the machine checks for the guessed solution whether the conjunction of $IK \vdash m$ 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) (Condition 3), (Condition 5) and (Condition 7). 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).
- (Condition 3):
  - 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 $(IK \vdash m) : P_i$ 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 $IK \vdash m$, 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 $IK \vdash k$ 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.
- (Condition 5):
  - 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 $\mathsf{spayload}(A, B)$ with another term m_2 during constraint reduction, then after unification both terms are labeled $\mathsf{spayload}(A, B)$. This labeling is always uniquely determined as the terms labeled $\mathsf{spayload}(A, B)$ must always be ground and if two terms are labeled $\mathsf{spayload}(A, B)$ and $\mathsf{spayload}(A', B')$, respectively, then they can only be unifiable if $A = A'$ and $B = B'$, by the properties we assume (cf. (Condition 4)). Further, we define that the labeling $\mathsf{rpayload}(A, B)$ is dominated

by $\mathsf{spayload}(A, B)$, 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 $\mathsf{Payload}$ 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 $\mathsf{spayload}(A, B)$. In the proof, we show that the constraint reduction would equally work by replacing the concrete m with the abstract $\mathsf{payload}(A, B)$.

- (Condition 7): in terms of constraints, we define payload secrecy as follows.
  - Consider a lazy intruder trace for P_1 and let ϕ_0 be the corresponding constraints and IK the intruder knowledge after ϕ_0. If the constraint $\phi_0 \wedge IK \vdash \mathsf{payload}(A, B)$ 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 ϕ_0 be the constraint of a lazy intruder trace, this time of P_2, and let IK be the resulting intruder knowledge. Let $m \in \mathcal{M}_{A,B}$ be a concrete payload that needs to be secret. If $\phi_0 \wedge IK \vdash m$ 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 defined in Section 4. 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 ϕ. 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 (Condition 7) 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 ϕ can be written as $\phi = \phi_0 \wedge IK \vdash m$ such that from no intruder knowledge of ϕ_0, in no satisfying interpretation of ϕ, a secret payload can be derived. Moreover, if a secret payload can be derived from IK, then m is one such secret payload. We can also assume that in the derivation of m from IK we do not use secret payloads as an intermediate reduction step.

This assumption is not a restriction for the following reason. If ϕ does not satisfy this property, then we can find a suitable ϕ' that does: cutting off from ϕ all conjuncts after reaching an intruder knowledge IK 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 $IK \vdash m$ conjunct, this is a valid symbolic trace of $P_1^{P_2} \parallel P_2$ (and IK the reached intruder knowledge). Following the further parts of the proof, we obtain that ϕ' can be split into a pure P_1 or a pure P_2 part. By (Condition 7), therefore the $IK \vdash m$ 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 ϕ_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 ϕ_0, then this holds over all reductions on ϕ_0.

Invariants

We consider now a reduction of ϕ to a simple constraint, which must be possible because ϕ 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 $m \in IK_0$ or $m \in \mathcal{V}$ by (Condition 5) and (Condition 7). By the invariants shown below, if this m is a variable, it can only be instantiated to a ground term in IK_0. Therefore, as soon we have a ground term in a special constraint (as defined in (Condition 3)), we know that it is an element of IK_0. We therefore do not change the meaning of ϕ 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 rule on constraint $IK \vdash m$ where some $m' \in IK$ 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 (Condition 5) 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 $IK \vdash m$ is of type P_i then m is either of type P_i or a public constant. If constraint $IK \vdash m$ 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 $\mathsf{spayload}(A, B)$ 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 $\mathsf{rpayload}(A, B)$.
5. (Payload Label) In all terms of type P_1, every occurrence of a ground subterm t of type $\mathsf{Payload}$ is labeled with $\mathsf{spayload}(A, B)$ 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, IK, σ, ϕ 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 (Condition 2). 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 (Condition 3).
- 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 (Condition 5)), the unifier σ 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 black-or-concrete payload property only applies to terms under $C_P[\cdot]$; and if it is a variable, then by (Condition 5), 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 (Condition 5) and the fact that it cannot be a variable) and moreover $m \in IK_0$ 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 ϕ 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 ϕ_1 be the P_1 constraints of ϕ and ϕ_2 be the P_2 constraints of ϕ, where the constraints may contain both P_1 and P_2 terms on the knowledge side. The case of the "last" conjunct $IK \vdash m$ of ϕ deserves a special treatment: if this m is a secret payload and the reduction of $IK \vdash m$ uses the analysis of a P_1 term, then $IK \vdash m$ shall be considered as part of ϕ_1; otherwise it is considered as part of ϕ_2.

Considering a sequence of reduction steps that turns ϕ into a simple (and satisfiable) constraint, we can split this reduction into a ϕ_1-part and a ϕ_2-part as expected. We thus have a reduction for ϕ_1 to a simple constraint and one for ϕ_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 ϕ_1 we never need P_2 messages of the intruder knowledge and vice versa.

Consider a conjunct $IK_i \vdash m_i$ of ϕ_1 where m_i is not the secret payload m. A unify step at this point can only be with a public value of $IK_0 \subseteq IK_i$ 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 $\mathsf{spayload}(A, B)$, then this must be public and thus in IK_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 IK_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 IK_0 part of IK_i.

The case that the reduction in ϕ_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 ϕ_1 instead, or it is a public payload that is contained in IK_0. Either way, we thus do not need the $P_1^{P_2}$ messages.

Therefore, by the property of execution independence (Condition 1), 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 (Condition 5), 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 $\mathsf{spayload}(A, B)$ or a variable labeled $\mathsf{rpayload}(A, B)$. Note that having split the attack into ϕ_1 and ϕ_2, the entire reduction for ϕ_1 works without considering any P_2 terms, except for those public payloads in IK_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 $\mathsf{spayload}(A, B)$ by $\mathsf{payload}(A, B)$ 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 (Condition 6). □

A.4 Extension: negation

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.

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

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

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 $IK \vdash m$ constraint. Intuitively, this excludes the substitution $\sigma = [y_1 \mapsto t_1, \ldots, y_m \mapsto t_m]$ 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 [30]. As shown in [29], 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 $IK \vdash m$ 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 $\{(y_1, t_1), \ldots, (y_m, t_m)\}\sigma$ has a solution if σ 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 $\mathsf{payload}(A, B)$ 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.*

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. □

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, 27]), 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. A LARGER 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)

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

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:

$$
\begin{array}{lcll}
A & \to & B & : \{|\mathsf{payload}(A,B)|\}_{K_A} \\
\hline
A & \bullet\!\!\to\!\!\bullet & B & : \mathsf{payload}(A,B)
\end{array}
$$

Symmetrically, the server uses:

$$
\begin{array}{lcll}
B & \to & A & : \{|\mathsf{payload}(B,A)|\}_{K_B} \\
\hline
B & \bullet\!\!\to\!\!\bullet & A & : \mathsf{payload}(A,B)
\end{array}
$$

An Application: Online Movie Stream Purchase.

As a simple application protocol P_2, we consider the online movie stream purchase

$$
\begin{array}{lcll}
A & \bullet\!\!\to\!\!\bullet & B & : [orderMovie, MovieID]_2 \\
B & \bullet\!\!\to\!\!\bullet & A & : [askConfirm, MovieID, Price, Date, StreamID]_5 \\
A & \bullet\!\!\to\!\!\bullet & B & : [confirm, MovieID, Price, Date, StreamID]_5
\end{array}
$$

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 $\{|\cdot|\}_{clientk(K_0)}$ or $\{|\cdot|\}_{serverk(K_0)}$ (recall that for different channel types like $A \bullet\!\!\to\!\!\bullet B$ vs. $B \bullet\!\!\to\!\!\bullet A$ 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 $\{|[clientTag, \cdot]_2|\}_{K_0}$ for the client sending messages and $\{|[serverTag, \cdot]_2|\}_{K_0}$ for the server sending messages. (Here $C_P^C[\cdot] = [clientTag, \cdot]_2$ and $C_P^S[\cdot] = [serverTag, \cdot]_2$.)

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 (Condition 3), 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 7: 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 (Condition 7), 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 [29] 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 [30], we can regard the pure unilaterally authenticated TLS *without* the login as a protocol that pro-

vides 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:

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

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

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

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 $\mathsf{payload}(A, B)$ because A is not authenticated—if we tried to verify P_0 like that, then we would get trivial authentication flaws. We should in this case use a pseudonym as the endpoint of the channel. In this case, it could be the premaster secret PMS that A has created, and then we require that $\mathsf{payload}(PMS, B)$ 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, 27].