Vertical Protocol Composition

Thomas Groß
IBM Research
Email: tgr@zurich.ibm.com

Sebastian Mödersheim
DTU Informatics
Email: samo@imm.dtu.dk

***Abstract*—The security of key exchange and secure channel protocols, such as TLS, has been studied intensively. However, only few works have considered what happens when the established keys are actually *used*—to run some protocol securely over the established "channel". We call this a vertical protocol composition, and it is truly commonplace in today's communication with the diversity of VPNs and secure browser sessions. In fact, it is normal that we have several layers of secure channels: For instance, on top of a VPN-connection, a browser may establish another secure channel (possibly with a different end point). Even using the same protocol several times in such a stack of channels is not unusual: An application may very well establish another TLS channel over an established one. We call this self-composition. In fact, there is nothing that tells us that all these compositions are sound, i.e., that the combination cannot introduce attacks that the individual protocols in isolation do not have.**

In this work, we prove a composability result in the symbolic model that allows for arbitrary vertical composition (including self-composition). It holds for protocols from any suite of channel and application protocols that fulfills a number of sufficient preconditions. These preconditions are satisfied for many practically relevant protocols such as TLS.

I. Introduction

Security protocols, such as key establishment, secure channels and VPNs, are a cornerstone of security on the Internet. Establishing the security of a protocol is challenging, not only because of its intrinsic complexity, but also because of its interactions with arbitrary, sometimes adversarial, environments. Even though composition of security protocols to higher-level protocols is common for most environments, its rigorous security analysis is still a major research problem.

Formal methods and automated deduction helped security protocol analysis. Both automated detection of flaws [1], [2] and proofs of security [3] were achieved. Automated analysis of security protocols is a hard problem in terms of undecidability of general insecurity [4] and NP-completeness of secrecy and authentication property checking for a bounded number of sessions [5]. Because of the computational complexity of the problems, researchers were often restricted to studying small problems. Still, several tools have been developed for security analysis [6], [2], [7] and successfully applied to the analysis of a plethora of security protocols.

However, in most cases the formal analyses only consider the protocol in isolation, while the actual deployment is within an environment of other protocols that might interfere with it. Directly analyzing such a system that is composed of many protocols is hardly feasible for several reasons. First, the verification methods often do not scale to the complexity of such a composed system. Second, there may be a large or even infinite number of ways to compose them, and this variety cannot directly be checked by automated verification. Third, if just one protocol of the composition is slightly changed, it is not sufficient to verify only this changed protocol again, but one has to verify the entire composition from scratch. For these reasons, it is desirable to have a modular *compositionality result* (also called *composability*) of the following form: given a suite of protocols that satisfy a certain sufficient condition and that are secure in isolation, then any way to compose them is a secure system, as well. Given that the sufficient conditions on the protocols are easy to check (statically), it is sufficient to verify protocols individually with formal methods.

Several works have considered the *parallel composition* of security protocols, i.e., when different protocols are deployed over the same communication medium (and possibly using the same key-infrastructures) [8], [9]. The potential problem here is that, although the protocols may be secure in isolation, their use in parallel may allow for attacks. This may happen in particular when they have similar message formats, so that an intruder can re-use a message (or message part) from one protocol in another protocol. To achieve compositionality along these ideas, protocols must have disjoint message format, even on (non-atomic) subterms. This condition can be checked statically.

In *sequential composition*, an output of one protocol is used as input for another, such as the session key output of a key exchange protocol being used as input for a secure channel protocol. Datta et. al. [8] have analyzed the sequential and parallel composition of protocols by composing processes, yet without considering keys. Ciobâcă and Cortier [10] analyze such compositions under consideration of shared data and employ disjointness principles introduced by Guttman and Thayer [11] to obtain a generic theorem for parallel and sequential composition.

Our focus here is *vertical composition*, which is a common phenomenon on the layered Internet. In this case, a client executes a protocol on top of another protocol, e.g., an application protocol over a secure channel established by TLS [12]. As in previous cases, such a composition can lead to attacks in general, even if the individual protocols are all secure in isolation. Mödersheim and Viganò [13] give

a first vertical composition result; however, it is limited to one transmitted message of the application protocol over the established channel and the sufficient condition is semantical (i.e., cannot be checked statically). Moreover, it does not allow for several compositions, e.g., establishing a secure channel on top of another channel.

The stacking of several protocols on top of each other is for instance introduced by the vast deployment of VPNs and secure channels established by browsers. It is common in such protocol stacks that several layers are established by same protocols, e.g., a TLS channel on top of another TLS channel. In fact, the interest of the authors in vertical (self-)composition was raised by a practical attack on a deployment of two TLS channels on top of another [14], [15]. An intruder establishes an unauthenticated channel with the server and forwards an honest client's channel establishment through his channel. When the client engages in a TLS Key Renegotiation to establish client authentication, the client's credentials are attributed to the intruder's channel. Therefore, the attack turned out to be a mis-association of the commands transmitted over the channels during the TLS Key Renegotiation (and would thus work with other channel protocols analogously). Even though the attack was not a fault of TLS itself or its stacking in this case, it raised the question whether vertical composition is secure, in general.

In fact, vertical composition is a composition class widely used in practice, for which we do not have sufficient compositionality results. To put it another way: we have a large system that is composed of many small parts, each of which has been analyzed extremely well with greatest possible rigor, but so far nothing allows us to conclude that a major way to put these parts together is actually secure. Our work supplies such a compositionality result for a large class of protocols.

Channel protocol dh :

$A \to B : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, \mathsf{exp}(g, X)])$
$B \to A : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_B}, [dh, A, \mathsf{exp}(g, Y)])$
$A \to B : \mathsf{scrypt}(h(\mathsf{exp}(\mathsf{exp}(g, X), Y), 0), keyack)$
$KA := h([\mathsf{exp}(\mathsf{exp}(g, X), Y), 0])$
$KB := h([\mathsf{exp}(\mathsf{exp}(g, X), Y), 1])$

Concrete application ca :

$A \to B : \mathsf{crypt}(\mathsf{pk}_B, [ca, M])$

Embed composition $ca\langle dh \rangle_{\mathsf{E}}$:

$A \to B : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, \mathsf{exp}(g, X)])$
$B \to A : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_B}, [dh, A, \mathsf{exp}(g, Y)])$
$KA := \ldots, KB := \ldots$
$A \to B : \mathsf{scrypt}(KA, keyack)$
$A \to B : \mathsf{scrypt}(KA, \mathsf{crypt}(\mathsf{pk}_B, [ca, M]))$

Abstract application aa :

$A \bullet\!\!\to\!\!\bullet\ B : [aa, M]$

Realize composition $aa[dh]_{\mathsf{R}}$:

$A \to B : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, \mathsf{exp}(g, X)])$
$B \to A : \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_B}, [dh, A, \mathsf{exp}(g, Y)])$
$KA := \ldots, KB := \ldots$
$A \to B : \mathsf{scrypt}(KA, keyack)$
$A \to B : \mathsf{scrypt}(KA, [aa, M])$

Figure 1. A collection of example protocols for composition. The notion of abstract application protocols and the corresponding realize composition type is discussed in the extended version of this paper [16]; here we focus only on concrete applications, channels, and embed composition.

A. The Ideas of the Paper

It is easy to craft an example of a secure channel protocol P that is secure in isolation, but will break when it is self-composed, i.e., when we run P on top of P itself. Along those lines, we will demonstrate in §II that vertical protocol composition can introduce attacks into a secure system.

This paper establishes reasonable compositionality preconditions on protocols (that P does not fulfill but, for instance, TLS does); under these preconditions it is indeed possible to safely compose arbitrary stacks of channels from compliant protocols. In contrast to previous compositionality results, we allow that several layers in such a stack may be established by the same protocol (e.g., several layers of TLS-channels) as well as an arbitrary depth of the stacking. Previous compositionality approaches usable for vertical composition, such as [10], [13], require disjointness between the protocols of the different layers and, thus, cannot allow the same channel protocol to be used more than once in a vertical protocol stack. Therefore, also the size of the stack is also bounded by the number of available channel protocols.

It may seem surprising at first that our result builds upon existing compositionality results that require disjointness. We start from a set of *atomic* protocols $\mathcal{P}$ (containing for instance TLS) and will assume that the protocols in $\mathcal{P}$ are indeed pairwise disjoint, so they can be safely deployed in parallel. The ideas to achieve compositionality results for the vertical composition, i.e., running possibly non-disjoint protocols on top of another, are laid out subsequently. First, we distinguish three kinds of atomic protocols: the *channel protocols*, the *concrete application protocols*, and the *abstract application protocols*. In this paper, we focus on the first two kinds and only mentions the third one; the extended version of this paper [16] contains a detailed treatment of *abstract application protocols* and their compositions, as well.

A channel protocol is a key exchange protocol between two parties A and B that establishes two symmetric keys,

one for protecting messages from A to B and one for messages from B to A.

Example 1. An example is protocol dh in Fig. 1. Here, crypt represents asymmetric encryption or signing (when the key used to prepare it is a private key, denoted by inv). scrypt denotes a symmetric encryption primitive that should provide not only confidentiality, but also integrity (e.g., by MAC-ing). Further, h is a cryptographic hash function and exp the modular exponentiation, for which we assume the algebraic property $\mathsf{exp}(\mathsf{exp}(g, X), Y) \approx \mathsf{exp}(\mathsf{exp}(g, Y), X)$. The variables X and Y are nonces freshly created by A and B, respectively. (We describe the details of the message model in §III) Both agents derive two symmetric keys: KA for messages from A to B and KB for the opposite direction.

In general, we require that the channel protocol is designed such that both parties contribute some fresh nonce to each of the keys and these nonces are not used anywhere else. This is a common method in robust key exchange design.

An application protocol is basically any protocol that may be run over a channel—including the channel protocols themselves. We distinguish two kinds of application protocols: *abstract* and *concrete* ones. The abstract ones focus on some high-level task and rely on the assumption that their messages are exchanged over a secure channel, but leave it abstract how that channel is implemented; SMTP is an example. This gives rise to the so-called *realize-composition* where the assumed channel of the abstract application is implemented by a concrete protocol. In contrast, a concrete application protocol already incorporates all the needed security mechanisms—i.e., we require that these protocols alone are already secure. For instance the protocol ca in Fig. 1 only insures confidentiality of message M but not integrity/authentication. The *embed composition* means running a concrete application protocol over a secure channel. This is what any form of VPN or secure layer protocol does: embedding the application protocols into a secure shell that the application is not aware of. In our concrete application ca this could be either to additionally ensure integrity of messages, or simply as a redundant layer of security that—hopefully—does not hurt the application. We focus here on this latter kind of embed composition and concrete application protocols, and leave the variant of the abstract application protocols and corresponding realize composition to the extended version of the paper [16]. We summarize embed and realize composition under the term *vertical composition*.

Given the set $\mathcal{P}$ of pairwise disjoint atomic protocols, $\mathcal{C}$ denotes the set of all their syntactically possible vertical and parallel compositions, including the parallel composition of applications over an arbitrary deep stack of channel protocols. The main result is that *every* composition in $\mathcal{C}$ is secure. The key is a requirement on the protocols of $\mathcal{P}$, introduced in Def. 5 in §VI-B similar to existing notions of *disjointness*. It is so central to our paper that we describe it precisely at this point already. For every protocol P we consider the set of message patterns that describe the sending and receiving of messages by honest agents, $MP(P)$. We then consider the set $EST(P)$ that contains for every message M also $\mathsf{scrypt}(K, M)$ for a new variable K (and everything modulo α-renaming); this represents the set of all message patterns with symmetric encryptions, when messages of P can be sent over an arbitrary deep stack of channels (i.e., one symmetric encryption layer per channel layer). Our disjointness notion of Def. 5 now requires (a) that every protocol $P \in \mathcal{P}$ is "disjoint from its own encryptions", i.e., messages of different encryption/channel layers cannot be confused, and (b) the message patterns with encryption $EST(P_1)$ and $EST(P_2)$ of two distinct protocols $P_1, P_2 \in \mathcal{P}$ are also disjoint. Roughly, this means for a given message M that can be sent or received by an honest agent in any composition $C \in \mathcal{C}$, we can tell a unique application protocol $P \in \mathcal{P}$ to which the message belongs, and derive the depth of the stack of channels, over which it was transmitted. By the fact that channel keys are composed in a certain way that is disjoint for distinct channel protocols (again by Def. 5), we can uniquely determine the channel protocols used. Thanks to this construction, we can then show the soundness of arbitrary stacks of protocols, including self-compositions like "TLS over TLS".

B. Contributions

The main result of this paper is a generic vertical composition theorem: for every pairwise disjoint protocol suite $\mathcal{P}$ of protocols that satisfies a number of conditions, if arbitrarily deep encryption produces no collisions, *every* vertical and parallel composition that can be formed with protocols of $\mathcal{P}$ is secure. This result includes arbitrary deep stacking of channels, even with the same protocol. Moreover, all these vertical compositions can be deployed in parallel. This provides both new theoretical insights and has many practical applications. On the theoretical side, we show that we can use existing results on parallel composition as a basis for the vertical compositionality result. While we exploit existing results that are based on disjointness of protocols, we can still allow for vertical self-composition, such as TLS over TLS. The reason that this is possible lies in the requirements on protocols suite $\mathcal{P}$—notably the disjointness of a protocol from its encryption and pairwise disjointness of $\mathcal{P}$ as well as a classification into channel, concrete and abstract application protocols—that allow us to attribute each message and message part to a unique context (i.e., atomic protocol and encryption layers).

On the practical side, the result can immediately be applied to the widely used protocols for establishing secure channels and VPNs such as TLS and IKE/IPSec, as well as the common (non-cryptographic) application protocols like

SMTP. Indeed, it is common practice to run applications over several layers of channels (and in fact the point of a virtual network is that such layering is indeed possible). We provide a soundness result given that the set of protocols considered satisfies a number of preconditions. These preconditions are realistic and good practice: Indeed, this work was carried out with the main examples TLS and IPSec/IKE in mind and we have entirely avoided any precondition that would not be satisfied by these protocols.[1]

Our composition theorem is also generic in the sense that it allows for arbitrary state-based safety properties over auxiliary predicates and positive intruder knowledge. The theorem therefore covers a large class of trace properties, where authentication is only one example.

C. Outline

§II gives an example of failing self-composition and motivates the restrictions and conditions we make in the following sections. §III introduces our protocol and intruder model, based on the Intermediate Format IF. §IV defines the classes of protocols that we work with in this paper. §V introduces the different kinds of compositions that we consider and the notion of protocol types. §VI contains key concepts for typing and disjointness; the preconditions in this section allow us to derive an important intermediate result, namely that in our compositions all messages can be attributed to a unique protocol type. §VII contains the main result that all compositions of the protocol suites we consider are secure, and how this can be derived from several theorems that are proved in the extended version of this paper [16].

II. Example Protocol Failing under Self-Composition

Before we proceed with the formal details of protocol model and composition, we illustrate in this section why vertical composition, and in particular self-composition, is tricky. To that end, we introduce an artificial toy protocol that is secure in isolation, but breaks under self-composition. This protocol violates several guidelines of good protocol design, and we can use it to motivate several of the preconditions of our compositionality result that rule out such bad protocols.

The protocol U (for "uncomposable") is a channel protocol that performs a two step handshake between parties A and B. It relies on pre-existing shared keys $\mathsf{sk}_{(A,B)}$ and $\mathsf{sk}_{(B,A)}$ (for the two directions) and in fact it does not establish new keys, but just acknowledges that these keys will be used and introduces a fresh nonce that is used as a session/channel identifier:

$$A \to B : \mathsf{crypt}(\mathsf{pk}_B, [A, B, \mathsf{sk}_{(A,B)}])$$
$$B \to A : \mathsf{scrypt}(\mathsf{sk}_{(B,A)}, \mathsf{crypt}(\mathsf{pk}_A, [B, A, N]))$$

After this handshake, every transmission of a message M from A to B has the form

$$A \to B : [N, \mathsf{scrypt}(\mathsf{sk}_{(A,B)}, M)]$$

where the fresh nonce N is used to identify the session/channel before decryption (transmissions from B to A are analogous). As goals, we assume the secrecy of the shared keys $\mathsf{sk}_{(A,B)}$ and $\mathsf{sk}_{(B,A)}$.

The handshake alone is a secure protocol, and even transmitting several "harmless" payloads in place of message M such as freshly generated nonces is fine. However, if we allow to run U itself as payload messages over the established channel, i.e., self-composition, then the following attack can happen:

$a \to b : \mathsf{crypt}(\mathsf{pk}_b, [a, b, \mathsf{sk}_{(a,b)}])$
Step 1 of plain U

$b \to a : \mathsf{scrypt}(\mathsf{sk}_{(b,a)}, \mathsf{crypt}(\mathsf{pk}_a, [b, a, n]))$
Step 2 of plain U, n some fresh new constant

$a \to b(i) : \mathsf{scrypt}(\mathsf{sk}_{(a,b)}, [n, \mathsf{crypt}(\mathsf{pk}_b, [a, b, \mathsf{sk}_{(a,b)}])])$
Step 1 of U over the established channel; intercepted by intruder, session dies.

$b \to a(i) : \mathsf{crypt}(\mathsf{pk}_a, b, a, \mathsf{sk}_{(b,a)})$
Step 1 of plain U, initiated by b; intercepted by intruder

$a(i) \to b : \mathsf{scrypt}(\mathsf{sk}_{(a,b)}, \mathsf{crypt}(\mathsf{pk}_b, [a, b, \mathsf{sk}_{(a,b)}]))$
replay of intercepted message from a, received by b as Step 2 of plain U, mistaking $\mathsf{sk}_{(a,b)}$ as session ID

$b \to a(i) : [\mathsf{sk}_{(a,b)}, \mathsf{scrypt}(\mathsf{sk}_{(a,b)}, \ldots)]$
b sends some payload, intruder learns the shared key.

Note that one can think of many other application protocols (besides U itself) that can break the security of U—basically any payload message that uses triples could be harmful here. In fact, this example shows that it is not trivial to give conditions that prevent interferences with an arbitrary payload protocol.

Here are some conditions that we require for channel protocols and that are violated by U. First, we require that a channel protocol must freshly generate keys, and both parties must contribute to these keys. The freshly established keys must only be used for a single channel and not be transmitted as payload part of a message (but only be used to encrypt messages).

Second, messages that have different meaning *must* have different format, so they cannot be confused. This is actually satisfied for the handshake alone. When we consider however the transmission over the channel, we see that

[1] Kerberos is the only major example that does not satisfy the preconditions (because each new session key is generated by the respective server alone without user interaction).

transmitting the first message of U over the channel produces a message that can be unified with the second message of U. This motivates our extension of the standard disjointness conditions: message formats should be additionally disjoint from their symmetric encryptions (consider Def. 5 for the details). In the given example, this could be achieved by inserting a tag into the body of the symmetric encryption of Step 2, which rules out the above attack already:

$$A \to B : \mathsf{crypt}(\mathsf{pk}_B, [A, B, \mathsf{sk}_{(A,B)}])$$
$$B \to A : \mathsf{scrypt}(\mathsf{sk}_{(B,A)}, [\mathbf{tag}, \mathsf{crypt}(\mathsf{pk}_A, [B, A, N])])$$

We conclude this discussion with the remark that we cannot show that any of the requirements we make are necessary, but they are sufficient and, in our opinion, part of good protocol design.

III. The Protocol Model

We introduce our model of protocols, messages, and the intruder behavior; this is the basis for a precise definition (a) of the classes of protocols that we reason about and (b) of the particular forms of protocol composition that we consider. We formalize protocols and their goals in the AVISPA Intermediate Format IF which we introduce along the way; for a detailed definition see [17].

A. Message Terms

As usual in the black-box cryptography models of security protocols, protocol *messages* (or *terms*) are represented in a term algebra over a signature Σ and variables $\mathcal{V}$. Both Σ and $\mathcal{V}$ are alphanumeric identifiers in IF, where symbols of Σ must begin with a lower-case letter, and those of $\mathcal{V}$ with an upper-case letter (so Σ and $\mathcal{V}$ are disjoint, and both are countable). We will set all IF identifiers in sans-serif, in particular to distinguish IF-variables from meta-variables of our argumentation (such as m or M). Σ consists of a denumerable set of constant symbols representing agent names (where i is the intruder), numbers, atomic keys etc.

Σ contains only a fixed set of function symbols representing operations on messages. In this paper, we work with the following symbols:

- $\mathsf{scrypt}(k, m)$ denotes the symmetric encryption of message m with symmetric key k. As it is standard in abstract term models, $\mathsf{scrypt}(\cdot, \cdot)$ does not denote a pure symmetric cipher (that only ensures confidentiality) but also includes mechanisms for ensuring integrity (such as a message authentication code). It is also possible to include other aspects of message transmission as part of this primitive, such as timestamps or sequence numbers (that our model abstracts from).
- $\mathsf{crypt}(pk, m)$ denotes the asymmetric encryption of message m with public key pk. $\mathsf{crypt}(\mathsf{inv}_{pk}, m)$ denotes the signature of message m with secret key inv_{pk}.
- a cryptographic hash function h,
- $\mathsf{exp}(B, E)$ as the modular exponentiation (omitting the modulus) for Diffie-Hellman, and
- the concatenation $[m_1, \ldots, m_n]$ of messages m_i.[2]

Σ also contains a fixed set of function symbols that do not represent operations on messages, but denote mappings of the model. For instance $\mathsf{pk}(a)$ may denote the public key of agent a, and $\mathsf{inv}_{\mathsf{pk}_a}$ the corresponding private key. These mappings of the model are indicated by writing their arguments as indices and we call terms built using these function symbols *dependent terms*. We consider all constants, variables, and dependent terms as *atomic terms*.

We assume a congruence relation $\approx$ over terms to model algebraic properties defined by a set of equations, such as $\mathsf{exp}(\mathsf{exp}(\mathsf{B}, \mathsf{X}), \mathsf{Y}) \approx \mathsf{exp}(\mathsf{exp}(\mathsf{B}, \mathsf{Y}), \mathsf{X})$ for Diffie-Hellman. We interpret terms in the *quotient algebra* $\mathcal{T}_\Sigma/_\approx$ (i.e., two terms are interpreted as being different iff this is a consequence of the algebraic properties). We use the standard notions of terms such as ground, substitution, unifier etc. (see [18]).

B. Message types

We will use a particular way of typing terms; note that the original IF allows also for untyped specifications. We discuss the exclusion of type-flaw attacks below. All constants in IF have a type from the set of *basic types*:

$$\{\mathsf{agent}, \mathsf{nonce}, \mathsf{symkey}, \mathsf{pubkey}, \mathsf{privkey}, \mathsf{tag}\}$$

We write $t : \tau$ to denote that term t has type τ. Functions induce *composed types*, namely if $(t_1 : \tau_1), \ldots, (t_n : \tau_n)$, then $f(t_1, \ldots, t_n) : f(\tau_1, \ldots, \tau_n)$. For instance, a Diffie-Hellman key has the type $\mathsf{exp}(\mathsf{exp}(\mathsf{nonce}, \mathsf{nonce}), \mathsf{nonce})$. We also require that all algebraic properties that we have are *type-correct*, i.e., left-hand side and right-hand side of every algebraic equation must have the same type. For instance we can have $\mathsf{exp}(\mathsf{exp}(\mathsf{B}, \mathsf{X}), \mathsf{Y}) = \mathsf{exp}(\mathsf{exp}(\mathsf{B}, \mathsf{Y}), \mathsf{X})$ but not $\mathsf{exp}(\mathsf{B}, 1) = \mathsf{B}$.

Variables can for now be either untyped, or have a basic or composed type. Variables of basic type can only be instantiated with constants or with dependent terms of the respective type. When variables have a type, then in any matching and unification, we only allow instantiations of variables with terms of the correct type as expected.

While thus all operations are injective functions on the type system, we consider the modeling functions as mapping into constants (e.g., $\mathsf{pk} : \mathsf{agent} \to \mathsf{pubkey}$ and $\mathsf{inv} : \mathsf{pubkey} \to \mathsf{privkey}$) so that ground dependent terms can be treated as constants (of a suitable type) and non-ground dependent terms can be treated as variables (of a suitable type).

[2] We use concatenation thus with different arity without further notice. Observe that with this notion of concatenation, we exclude a number of low-level type-flaw attacks where messages of different length are confused due to poor structuring of concatenated messages.

C. Facts, States and Rules

We use a fixed number of *fact symbols* (disjoint from Σ and $\mathcal{V}$) that each have an arity. A *fact* has the form $f(t_1, \ldots, t_n)$ where f is a fact symbol of arity n and the t_i are (message) terms. The most important facts that we will use are $\mathsf{ik}(m)$ to express that the intruder knows message m, $\mathsf{state}^{PID.TID}_{A:RID.PC}[M_1, \ldots, M_n]$ to represent the local state of an honest agent, and $\mathsf{net}_{(A,B)}(M)$ to denote that a message M has been sent on the insecure network; A is here the name of the (claimed) sender and B the intended recipient. Here PID represents the protocol name, RID the role of that protocol, PC is the "program counter", TID a thread identifier (to distinguish several parallel sessions that an agent takes place in), A is the name of the agent, and $M_1, \ldots, M_n$ are messages to represent the knowledge of the agent in that thread; note that the order of the messages does matter as this should be thought of being the fields of a record that is filled and modified during protocol execution. A further fact symbol is the nullary symbol attack that represents that we have an attack. We introduce further fact symbols below when we need them.

A state is a set of ground facts. An IF specification consists of an initial state and a finite set of *transition rules*, defining a transition relation on states. Transition rules have the form

$$PF.NF.C \; =[V]\Rightarrow \; RF$$

where

- PF and RF are sets of facts, NF is a set of *negated facts* of the form $not(f)$ where f is a fact,
- C is a set of conditions, that is, inequalities on terms,
- V is a set of variables.

It must hold that $vars(RF) \subseteq vars(PF) \cup V$. For such a transition rule, we distinguish the left-hand side (LHS) defining the preceding state and the right-hand side (RHS) defining the result state. For the LHS, PF and NF define the preceding state with positive and negative facts that are matched against. The conditions C restrict the matching with inequalities over terms. The variables V are existentially quantified in the rule. For the RHS, RF defines the resulting facts, where the variables of RF must be a subset of the variables of the positive facts of the LHS PF (excluding the variables only occurring in negative facts and conditions) and the existentially quantified variables V.

The transition relation $\rightarrow_R$ induced by the transition rules R is defined as follows: $S \rightarrow_R S'$ holds iff there is a rule $(PF.NF.C \; =[V]\Rightarrow \; RF) \in R$ and a grounding substitution σ with domain $vars(PF) \cup V$, such that

- $PF\sigma \subseteq S$.
- For all substitutions χ that substitute the remaining variables of the rule it holds that
  - for every $not(f) \in NF$, $f\sigma\chi \notin S$.
  - for every $s \neq t \in C$, $s\sigma\chi \not\approx t\sigma\chi$.
- For each $v \in V$, $v\sigma$ is a fresh constant (that does not occur in S).
- $S' = (S \setminus PF\sigma) \cup RF\sigma$.

Example 2. As an example, let us consider the first transition rule of role bob (played by some agent B) in the example dh of Fig. 1, p. 2; the full IF specification of the protocol is found in Fig. 2.

$$\mathsf{state}^{dh.TID}_{B:bob.1}[A].\mathsf{net}_{(A,B)}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, GX]))$$
$$=[Y]\Rightarrow \mathsf{state}^{dh.TID}_{B:bob.2}[h(\mathsf{exp}(GX, Y), 0), h(\mathsf{exp}(GX, Y), 1), A].$$
$$\mathsf{net}_{(A,B)}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_B}, [dh, A, \mathsf{exp}(g, Y)])).$$

This rule is read as follows. It can be applied to any state where an agent B playing bob is in the initial state of its protocol execution (expecting a message from agent A) and we have a message of the appropriate form on the network, apparently from A. This is done by pattern matching, i.e., all occurrences of the left-hand side variables must agree. In particular, the message must be signed with A's private key $\mathsf{inv}_{\mathsf{pk}_A}$. Observe that B cannot check that the last component of that message indeed has the format $\mathsf{exp}(g, X)$ for some X. Thus, we have here a variable GX that can be matched with any message. On the transition, B will create a fresh value Y (as indicated by the Y on the arrow) and send out an appropriate answer message. B will also store the newly derived keys KA and KB in his state, i.e., $h(\mathsf{exp}(GX, Y), 0)$ and $h(\mathsf{exp}(GX, Y), 1)$. We will later add some additional facts to the right-hand side, which will help us formulate *attack states* in a convenient way.

The following aspect is crucial: on the transition, the state-fact that had been matched on the LHS is being removed from the current state and the new state fact is introduced. In contrast, we define the net fact as being *persistent*, i.e., it remains on the state during all further transitions (so that messages can be received any number of times).

D. Intruder Deduction

We describe the intruder deduction by a protocol-independent set of rules that we assume to be present in all protocol descriptions (and will not list anymore later). These rules deal with *intruder knowledge* of messages, formalized by the persistent predicate ik. First, as it is standard, we model that the intruder controls the entire network:

$$\mathsf{ik(A).ik(B).ik(M)} \Rightarrow \mathsf{net_{(A,B)}(M)}$$
$$\mathsf{net_{(A,B)}(M)} \Rightarrow \mathsf{ik(A).ik(B).ik(M)}$$

The intruder deduction on messages is expressed by the following rules:

$$\begin{aligned}
&\mathsf{ik}(\mathsf{K}).\mathsf{ik}(\mathsf{M}) \to \mathsf{ik}(\mathsf{scrypt}(\mathsf{K},\mathsf{M}))\\
&\mathsf{ik}(\mathsf{scrypt}(\mathsf{K},\mathsf{M})).\mathsf{ik}(\mathsf{K}) \to \mathsf{ik}(\mathsf{M})\\
&\mathsf{ik}(\mathsf{K}).\mathsf{ik}(\mathsf{M}) \to \mathsf{ik}(\mathsf{crypt}(\mathsf{K},\mathsf{M}))\\
&\mathsf{ik}(\mathsf{B}).\mathsf{ik}(\mathsf{X}) \to \mathsf{ik}(\mathsf{exp}(\mathsf{B},\mathsf{X}))\\
&\mathsf{ik}(\mathsf{crypt}(\mathsf{K},\mathsf{M})).\mathsf{ik}(\mathsf{inv}_\mathsf{K}) \to \mathsf{ik}(\mathsf{M})\\
&\mathsf{ik}(\mathsf{M}_1).\ldots.\mathsf{ik}(\mathsf{M}_n) \to \mathsf{ik}([\mathsf{M}_1,\ldots,\mathsf{M}_n]) \quad \text{for } n \in \mathbb{N}\\
&\mathsf{ik}([\mathsf{M}_1,\ldots,\mathsf{M}_n]) \to \mathsf{ik}(\mathsf{M}_1).\ldots.\mathsf{ik}(\mathsf{M}_n) \quad \text{for } n \in \mathbb{N}\\
&\mathsf{ik}(\mathsf{M}) \Rightarrow\!\!\!\!\!\!\Rightarrow \mathsf{ik}(\mathsf{N})
\end{aligned}$$

which formalize that the intruder can encrypt and decrypt messages whenever he has the appropriate keys; he can build exponents for Diffie-Hellman; he can construct and deconstruct tuples and he can freshly create nonces. Note that we have here an infinite set of rules, but this can be restricted to those n-tuples that are used in the concrete protocol.

We forbid that ik facts can occur negatively in a rule; this ensures that attacks grow monotonically with the intruder knowledge (if the intruder learns something, this cannot decrease the number of attacks). For simplicity, we assume in this work that all agents besides the intruder are honest (but IF can express of course other intruder models). Further, we assume that the initial state will contain the persistent fact $\mathsf{agent}(a)$ and $\mathsf{ik}(a)$ for every constant a of type agent.

E. Attacks and Security

State-based safety properties can be formalized by transition rules that by their left-hand side describe states that qualify as an attack and the right-hand side being simply attack.

For example, secrecy properties will be specified by using a new fact $\mathsf{secret}(m, a)$ that expresses m is supposed to be a secret with agent a; this fact will be appropriately generated on the right-hand side of a transition rule of an honest agent. We then specify–independent of the details of the protocol– the rule

$$\mathsf{secret}(\mathsf{M},\mathsf{A}).\mathsf{ik}(\mathsf{M}).(\mathsf{A} \neq \mathsf{i}) \Rightarrow \mathsf{attack} \tag{1}$$

to express that it is considered as an attack if the intruder finds out a secret that is not meant for him.

An *attack state* is any state that contains the attack symbol. We say that an IF protocol is *secure* iff no attack state is reachable from the initial state using the transition relation.

IV. Protocol Classes

Based on the protocol model of the preceding section, we define now two classes of protocols, namely channel protocols and (concrete) application protocols. The extended version [16] introduces the additional class of abstract application protocols. We write $app(A)$ if A belongs to the class of concrete application protocols and $channel(C)$ if C belongs to the class of channel protocols. Our composition results are based on these classes, i.e., we allow $A\langle C\rangle_\mathsf{E}$—running A over C—only for protocols with $app(A)$ and $channel(C)$.

A. Concrete Application Protocols

IF allows for the specification of a huge class of protocols, and for many of them, our results may not hold. We thus define appropriate subclasses of protocols that have properties suitable for composition. The first class are concrete application protocols that may either be running directly over the insecure network or over the secure channels that we establish. These protocols shall already be protected sufficiently against the insecure network. The point is just that running them over a secure channel anyway should not hurt. An example would be a TLS-secured web-application that we may run directly over the network or over some VPN (that adds for this application a redundant layer of protection). Intuitively, we make the following restrictions on concrete application protocols:

- The initialization depends only on a number of agents that play the various other roles.
- Each thread of an honest agent is described by exactly one state fact and has an incoming and outgoing message in the intruder knowledge on each transition. This excludes multi-threaded applications and synchronous communication between agents.
- Except for the concrete message format sent or received, we ensure that the protocol description, including the formulation of its goals, is oblivious to whether it is running directly on an insecure channel or rather routed over some secure channel. In particular, we must prevent that goals are formulated using state facts, because they will be slightly changed in composed protocols.
- As said before, we have restricted ourselves to two party protocols and we want to assume that the names of the roles are alice and bob (i.e., the role IDs in transitions) and that A and B are the variables that hold the respective agent names in the transition rules of honest agents.[3]

An *initialization rule* has the form

$$\mathsf{agent}(A_1).\mathsf{agent}(A_2) =\!\![TID]\!\Rightarrow \mathsf{state}^{pid.TID}_{A_1:rid.pc}[A_2]$$

where either

- $A_1 = \mathsf{A}$, $rid = \mathsf{alice}$, $A_2 = \mathsf{B}$, or
- $A_1 = \mathsf{B}$, $rid = \mathsf{bob}$, $A_2 = \mathsf{A}$.

Recall that identifiers in *italics* are meta-variables and identifiers in sans-serif are IF variables. This rule creates a new thread for agent A_1 playing in role rid of protocol

[3]This does *not* a priori exclude that there can be confusions between agents about their communication partners.

pid (beginning at step pc), who would like to communicate with agent A_2 (to play role bob). This creates a fresh thread identifier TID and thereby induces an unbounded number of sessions between all combinations of agents. In particular, the intruder can be a normal participant of every role.

A *concrete application protocol transition rule* has the form

$$\mathsf{state}^{PID.TID}_{A:RID.PC}[Msgs].\mathsf{net}_{(B,A)}(inMsg)$$
$$=[V]\Rightarrow \mathsf{state}^{PID.TID}_{A:RID.PC'}[Msgs'].$$
$$\mathsf{net}_{(A,B')}(outMsg).gfacts$$

where we have either

- $A = \mathsf{A}$, $B = \mathsf{B}$, $RID = \mathsf{alice}$ or
- $A = \mathsf{B}$, $B = \mathsf{A}$, $RID = \mathsf{bob}$.

Moreover, B must be a variable of both lists $Msgs$ and $Msgs'$. Finally, we have the *goal facts* $gfacts$ such as the above introduced secret. In the extended version [16], we additionally allow that the rules can contain *set conditions* to model participants that maintain a database of items.

An *concrete application protocol* is now defined by a set of initialization rules, a set of protocol transition rules and a set of attack rules, for which we forbid the inclusion of state facts.

Example 3. As an example, the concrete application protocol ca from Fig. 1, p. 2, looks as follows in IF.

$$\mathsf{agent(A).agent(B)} =[\mathsf{TID}]\Rightarrow \mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{A:alice.1}}[\mathsf{B}]$$

$$\mathsf{agent(A).agent(B)} =[\mathsf{TID}]\Rightarrow \mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{B:bob.1}}[\mathsf{A}]$$

$$\mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{A:alice.1}}[\mathsf{B}] =[\mathsf{M}]\Rightarrow$$
$$\mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{A:alice.2}}[\mathsf{B},\mathsf{M}].\mathsf{net}_{(\mathsf{A},\mathsf{B})}(\mathsf{crypt}(\mathsf{pk_B},[\mathsf{ca},\mathsf{M}])).\mathsf{secret}(\mathsf{B},\mathsf{M})$$

$$\mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{B:bob.1}}[\mathsf{A}].\mathsf{net}_{(\mathsf{A},\mathsf{B})}(\mathsf{crypt}(\mathsf{pk_B},[\mathsf{ca},\mathsf{M}]))$$
$$\Rightarrow \mathsf{state}^{\mathsf{ca.TID}}_{\mathsf{B:bob.2}}[\mathsf{A},\mathsf{M}].\mathsf{secret}(\mathsf{A},\mathsf{M})$$

Here, we declare the transmitted nonce M as a secret, using the Rule (1) in §III-E.

B. Abstract Application Protocols

As already mentioned, in the extended version of this paper [16] we also consider another type of protocol that generalizes the concrete application protocols by allowing message transmissions over abstract secure channels—namely abstracting from the concrete implementation of these channels. The corresponding *realize* composition type $A[C]_{\mathsf{R}}$ means that this channel assumed by the abstract application A is realized by channel protocol C. This is similar to our embed composition, and, for simplicity, we focus in this paper on the embed-composition and concrete application protocols. We refer to [16] for a detailed treatment of abstract application protocols and realize composition.

C. Channel Protocols

We now define a third class that is a special case of concrete application protocols, namely channel protocols (or also called key-exchange protocols). The task of such a protocol is to establish two secure shared keys between two parties alice and bob, one for encrypting messages from alice to bob and the other for messages from bob to alice. Given that the keys are indeed authenticated, confidential, and fresh, this establishes a secure channel between the two parties over which other concrete application protocols can be run, namely encrypting messages with the appropriate symmetric key. Examples of such protocols are TLS or the various versions of IPSec/IKE. Informally, the restrictions we make are as follows:

- The protocols establish a pair of shared keys between two parties (one key for each communication direction). Honest parties must contribute something fresh to each of them (the intruder does of course whatever he likes, when playing any of the roles).
- This key is secure (i.e., authenticated and confidential).
- The key-exchange is isolated from other protocols, i.e., it does not depend on shared knowledge with other sessions (except for long-term keys, modeled as dependent terms) and cannot "leak" information to them (i.e., freshly created nonces here are not used elsewhere).

Formally, a concrete application protocol is a *key-exchange protocol*, iff the following additional conditions are met:

- The PC for each rule is increasing in each transition rule, so there is a largest PC, or final state of the key-exchange.
- In the final state of $Alice$, we require the following conditions:
  (1) The first two terms in the message list of the final local state of alice represent the established symmetric keys; let us call these terms KA and KB.
  (2) In some transition rule, Alice creates a fresh value V (and keeps it in one message field of her local state); V occurs in both KA and KB. We require that V cannot be deduced from KA or KB.
  (3) In the rule where Alice creates V, we have the secrecy fact $\mathsf{secret}(V,\mathsf{B})$ (for B being the name of the agent playing role bob) and thereby require that V cannot be known by the intruder unless $\mathsf{B} = i$.
  (4) In the first transition of alice where the value of the keys KA and KB is determined, she issues the event $\mathsf{candidate}(\mathsf{A},\mathsf{B},KA,KB)$
  (5) In the transition to the final state of alice, she issues the event $\mathsf{agreed}(\mathsf{A},\mathsf{B},KA,KB)$
  All these requirements similarly must hold for role bob, *mutatis mutandis*.
- The security goals are fixed to be the following secrecy and authentication goals of the exchanged keys defined

using the agreed and candidate facts: (1) The authentication of the keys is violated when there is an agreed event without a matching candidate event.

$$\mathsf{agreed(A,B,KA,KB).}$$
$$\mathsf{not(candidate(B,A,KB,KA)).A} \neq \mathsf{i.B} \neq \mathsf{i}$$
$$\Rightarrow \mathsf{attack}$$

We do not require anything about freshness here, because this is implicit in the construction by the requirement that both agent must contribute fresh random numbers to the key. (2) The secrecy of a key is violated if an honest A thinks to share a key with an honest B (or vice versa) and the intruder finds out the key:

$$\mathsf{agreed(A,B,KA,KB).ik(KA).A} \neq \mathsf{i.B} \neq \mathsf{i}$$
$$\Rightarrow \mathsf{attack}$$

Honest users issue the events as follows: Whenever a principal has determined the session keys in his view, it issues a candidate event with the keys' context (own agent name, partner agent name, own session key, partner session key). Once, a principal acknowledges session keys as validated in his respective view, it issues an agreed event with the same context. A (candidate, agreed)-pair with matching context constitutes a successful key agreement in the principal's view. This mechanism allows us to check for matching conversations in the analysis trace and is similar to the matching conversations paradigm by Bellare and Rogaway [19], [20] as well as the non-injective agreement by Lowe [21].

Example 4. In Figure 2, we outline the IF formalization of the Diffie-Hellman-based key exchange from Fig. 1, p. 2. The first two rules are the initialization rules. Rule 3 models alice' view on sending the first message, in which alice instantiates a fresh nonce X, computes $\mathsf{exp(g,X)}$ and sends a signed message with that payload to B. Rule 4 models bob's view on receiving the first message and replying with the second message. First note that bob cannot check the structure of the received value (i.e., that it is of the form $\mathsf{exp}(g, X)$ for some X). We thus have here a variable GX that can match every term. bob generates the fresh Y and can now derive the key terms KA and KB, where the Diffie-Hellman part is $\mathsf{exp}(GX, Y)$. In fact, the $KA := \ldots$ and $KB := \ldots$ here is an abbreviating notation for readability: all further occurrences of these two variables in the rest of this rule shall be replaced accordingly. In addition, bob generates the goal facts candidate() and secret() for the derived keys as required to formulate the authentication and secrecy goals of the key exchange. In Rule 5, alice similarly receives bob's half-key as GY and derives the keys analogously, issuing the candidate() and secret() facts for the goals. Since this is alice's last transition of the handshake, she also issues the agreed() fact that indicates

$$\mathsf{agent(A).agent(B)} \; =[\mathsf{TID}]\Rightarrow \; \mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice.1}}[\mathsf{B}]$$

$$\mathsf{agent(A).agent(B)} \; =[\mathsf{TID}]\Rightarrow \; \mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{B:bob.1}}[\mathsf{A}]$$

$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice.1}}[\mathsf{B}]$$
$$=[\mathsf{X}]\Rightarrow \mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice.2}}[\mathsf{B,X}].$$
$$\mathsf{net}_{(\mathsf{A,B})}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk_A}}, [\mathsf{dh,B,exp(g,X)}]))$$

$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{B:bob.1}}[\mathsf{A}].\mathsf{net}_{(\mathsf{A,B})}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk_A}}, [\mathsf{dh,B,GX}]))$$
$$=[\mathsf{Y}]\Rightarrow \mathsf{KA} := \mathsf{h}(\mathsf{exp}(\mathsf{GX,Y}),0), \mathsf{KB} := \mathsf{h}(\mathsf{exp}(\mathsf{GX,Y}),1)$$
$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{B:bob.2}}[\mathsf{KB,KA,A}].$$
$$\mathsf{net}_{(\mathsf{A,B})}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk_B}}, [\mathsf{dh,A,exp(g,Y)}])).$$
$$\mathsf{candidate(B,A,KB,KA).secret(A,KB).secret(A,KA)}$$

$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice.2}}[\mathsf{B,X}].\mathsf{net}_{(\mathsf{B,A})}(\mathsf{crypt}(\mathsf{inv}_{\mathsf{pk_B}}, [\mathsf{dh,A,GY}])).$$
$$\Rightarrow \mathsf{KA} := \mathsf{h}(\mathsf{exp}(\mathsf{GY,X}),0), \mathsf{KB} := \mathsf{h}(\mathsf{exp}(\mathsf{GY,X}),1)$$
$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice.3}}[\mathsf{KA,KB,B}].\mathsf{net}_{(\mathsf{A,B})}(\mathsf{scrypt}(\mathsf{KA,keyack})).$$
$$\mathsf{candidate(A,B,KA,KB).agreed(A,B,KA,KB).}$$
$$\mathsf{secret(B,KA).secret(B,KB)}$$

$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{B:bob.2}}[\mathsf{KA,KB,A}].\mathsf{net}_{(\mathsf{A,B})}(\mathsf{scrypt}(\mathsf{KA,keyack}))$$
$$\Rightarrow \mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{B:bob.3}}[\mathsf{KA,KB,A}].\mathsf{agreed(B,A,KB,KA)}$$

Figure 2. IF formalization of the DH key exchange from Figure 1.

that she now accepts the keys as agreed and can start sending and receiving messages encrypted with those keys. Finally in the last rule, bob receives the acknowledgment from alice and issues a corresponding agreed() fact as well.

V. Protocol Composition

Based on the classes of protocols introduced in Section IV, we now introduce the protocol composition operations that can be applied to these classes. Recall that we denote with $app(A)$ that A belongs to the class of (concrete) application protocols, $channel(C)$ that C belong to the class of channel protocols, and $sapp(A)$ that A belongs to the class of abstract application protocols (which we only discuss in [16]).

We use the following notation for the different kinds of composition:

A. $P_1 \parallel P_2$ for the *parallel composition* of P_1 and P_2, i.e., the two protocols are run independently over an insecure network. (This is also called horizontal composition as opposed to vertical composition.) The classes of abstract application protocols and of concrete application protocols are both closed under parallel composition.

B. $A\langle C\rangle_{\mathsf{E}}$, for $app(A)$ and $channel(C)$. We call this vertical composition *embed composition*. It means that we first establish shared keys using C and then run the concrete application protocol A, but where now every message is additionally encrypted with the symmetric

key for the respective direction. The result is again a concrete application protocol, i.e., $app(A\langle C\rangle_{\mathsf{E}})$.

C. $A[C]_{\mathsf{R}}$, for $sapp(A)$ and $channel(C)$. We call this vertical composition *realize composition*; we treat it in detail only in the extended version [16]. Here, one first runs the protocol C to establish a pair of symmetric keys for secure communication. Afterwards, one runs A, but replaces the abstract secure channels by symmetric encryption with the respective keys for the communication direction. The result of this composition is a concrete application protocol, i.e., $app(A[C]_{\mathsf{R}})$.

The closure properties of the composition types (i.e., that the results are concrete application protocols, or abstract application protocols, respectively) are important for our construction: together with the composability results for each type of composition, we can build arbitrary complex compositions of the protocols we start with (if they satisfy the conditions of our composition theorems); so for instance we may run an appropriate concrete application over any number of TLS layers. In general, we cannot run an abstract application protocol directly over a channel, i.e., $A\langle C\rangle_{\mathsf{E}}$ for $sapp(A)$, because A may contain abstract secure channels that cannot be "routed" over a real channel; we always have to first implement the abstract channel by some channel protocol, i.e., $A[C]_{\mathsf{R}}$, which can then be routed over other channels.

Precondition 1. *From now on, we expect that we are given a finite set $\mathcal{P}$, consisting of channel, concrete application, and abstract application protocols. We shall call protocols of $\mathcal{P}$* atomic protocols*, and we will consider only protocols of $\mathcal{P}$ and their compositions for the rest of this of this paper. Also we assume that every protocol $P \in \mathcal{P}$ is secure in isolation.*

Example 5. Our running example is $\mathcal{P} = \{dh, aa, ca\}$.

Definition 1. *Let $\mathcal{C}$ be the set of all "syntactically correct" compositions of $\mathcal{P}$, i.e., the least set that contains $\mathcal{P}$ and that is closed under the following rules:*

- *If $P_1, P_2 \in \mathcal{C}$ then also $P_1 \parallel P_2 \in \mathcal{C}$.*
- *If $A, C \in \mathcal{C}$ and $app(A)$ and $channel(C)$ then also $A\langle C\rangle_{\mathsf{E}} \in \mathcal{C}$.*
- *If $A, C \in \mathcal{C}$ and $sapp(A)$ and $channel(C)$ then also $A[C]_{\mathsf{R}} \in \mathcal{C}$.*

We prove that if all protocols of $\mathcal{P}$ are secure in isolation and satisfy several disjointness conditions introduced in the following sections, then all their compositions in $\mathcal{C}$ are also secure. This will be the main result of this paper. For our running example, we have for instance that $(aa[dh]_{\mathsf{R}} \parallel ca)\langle dh\rangle_{\mathsf{E}}$ is a syntactically correct composition, but $ca[dh]_{\mathsf{R}}$ is not.

Since we focus on vertical and not on (pure) parallel composition, we assume here that the parallel composition of the atomic protocol is already secure:[4]

[4]Note that parallel self-composition is trivial: $P \parallel P = P$.

Precondition 2. *We expect that all protocols of $\mathcal{P}$ are secure under* parallel composition*, i.e., $\parallel_{P\in\mathcal{P}} P$ is secure.*

Works like [9], [10] can be used to prove such a parallel compositionality result. The basic idea is that non-atomic parts of the deployed protocols should be sufficiently different so that messages cannot be confused. This will ensure that the intruder cannot reuse messages or message parts from one protocol in another protocol. We must, however, also ensure that the intruder cannot learn long-term constants from one protocol that are secrets in the other (e.g., private keys of honest agents). Moreover, we must ensure that neither the goals nor the databases of the agents can produce conflicts; a reasonable assumption here would be to assume they are also disjoint for the different protocols (i.e., the different atomic protocols do not communicate with each other over a database). However we note that our vertical composition works for every set $\mathcal{P}$ for which our preconditions hold (no matter how to achieve the parallel composition result).

A. Protocol Types

The preconditions that we introduce below will allow us to associate every message that honest agents can send or receive to a unique protocol $C \in \mathcal{C}$ where C does not contain parallel composition. This justifies to speak of a message being of "type τ" where τ identifies a protocol composition.

Recall that state facts have a field for a protocol name. In the definition of the protocol classes, we did not require that this field must hold the same constant in all rules of a protocol, i.e., we allow for "heterogeneous" protocols. We call a protocol *homogeneous* iff all state facts in its rule carry the same protocol identifier. We take for granted that each protocol of $\mathcal{P}$ is homogeneous, thus every $P \in \mathcal{P}$ has a unique protocol identifier and we call this identifier its type.

Definition 2. *Let T_C be the set of* channel types*, i.e., the protocol identifiers of those $P \in \mathcal{P}$ for which $channel(P)$ holds. T_A^0 is the set of* atomic concrete application types*, i.e., the protocol identifiers of those $P \in \mathcal{P}$ for which $app(P)$ but not $channel(P)$. Finally let T_S be the set of* atomic abstract application types*, i.e., the protocol identifiers of those $P \in \mathcal{P}$ for which $sapp(P)$ but not $app(P)$. T_A is the least set that contains T_A^0 and that is closed under the following two rules:*

- *If $\tau_A \in T_A$ and $\tau_C \in T_C$, then also $\tau_A\langle\tau_C\rangle_{\mathsf{E}} \in T_A$.*
- *If $\tau_S \in T_S$ and $\tau_C \in T_C$, then also $\tau_S[\tau_C]_{\mathsf{R}} \in T_A$.*

We define the set $T = T_C \cup T_S \cup T_A$ of all protocol types*. Moreover $T_0 = T_S \cup T_C \cup T_A^0$ is the set of all* atomic protocol types*.*

Let $MP(P)$ denote the message patterns *of a protocol P, that consist of all messages M that appear as $\mathsf{net}_{(A,B)}(M)$ or $\mathsf{secCh}_{(A,B)}(M)$ in a rule of P; we apply an α-renaming such that different elements of $MP(P)$ have disjoint variables; also, for vertically composed protocols $P = A\langle C\rangle_{\mathsf{E}}$ and $P = A[C]_{\mathsf{R}}$, we exclude those messages from $MP(P)$*

that appear only in the rules of C. With $ST(P)$ we denote all non-atomic subterms of $MP(P)$, again α-renamed so that elements have disjoint variables.

Let $P \in \mathcal{C}$ be any composed protocol that has a type $\tau \in T$. We say that a message m has protocol type *τ, and write $m : \tau$, iff there is a unifier between m and an element of $ST(P)$. (Theorem 2 shows that this protocol type is unique under the preconditions on $\mathcal{P}$.)*

Example 6. For our running example from Fig. 1, p. 2, we have $T_C = \{dh\}$, $T_A^0 = \{ca\}$, $T_S = \{aa\}$ and $T_A = \{ca, ca\langle dh\rangle_{\mathsf{E}}, aa[dh]_{\mathsf{R}}, ca\langle dh\rangle_{\mathsf{E}}\langle dh\rangle_{\mathsf{E}}, aa[dh]_{\mathsf{R}}\langle dh\rangle_{\mathsf{E}}, \ldots\}$

In this definition we have introduced composed protocol types using the "composition operators" $\cdot\langle\cdot\rangle_{\mathsf{E}}$ and $\cdot[\cdot]_{\mathsf{R}}$. We do not have composed types for parallel composition: this will always give heterogeneous protocols. This reflects that in a parallel composition, every action (sending or receiving) of an agent can be attributed to either protocol that had been composed in parallel, while in all vertical compositions (i.e., running over a channel) both the concrete application and the channel protocol are part of the message and are thus represented in the type expression of the message. Consider the example composition $(P_1 \parallel P_2)\langle C\rangle_{\mathsf{E}}$ for $P_1, P_2, C \in \mathcal{P}$ of types τ_1, τ_2, τ_c; this composition is heterogeneous, consisting of the two protocol types $P_1\langle C\rangle_{\mathsf{E}}$ and $P_2\langle C\rangle_{\mathsf{E}}$.

We take for granted that our construction of composed protocol types can be mapped in a collision-free way to IF protocol identifiers, so that we can use the composed types in state facts of the composition as protocol identifiers.

B. Parallel Composition

Definition 3. *Given two protocols P_1 and P_2 described by IF rules. Then their* parallel composition *$P_1 \parallel P_2$, is simply the union of the two sets of rules.*

It is immediate that if $app(P_1)$ and $app(P_2)$, then also $app(P_1 \parallel P_2)$; similarly, if $sapp(P_1)$ and $sapp(P_2)$ then also $sapp(P_1 \parallel P_2)$.

C. Embed: $A\langle C\rangle_{\mathsf{E}}$ Composition

Definition 4. *Given protocols $A, C \in \mathcal{P}$ where $app(A)$ and $channel(C)$, we define the composed protocol $A\langle C\rangle_{\mathsf{E}}$ as the following set of rules:*

1) *All initialization and transition rules of C without modification.*
2) *A modification (made precise below) of the transition rules of A that additionally contains two shared keys in the local states of roles* alice *and* bob*, and that are used to encrypt all messages from* alice *to* bob *and vice-versa.*
3) *A modification of the initialization rules of A that links the initial states of A with final states of C.*
4) *All attack rules of A and C.*

Ad 2. By our assumptions, both A and C have two protocol roles alice *and* bob *played by agents identified by variables* A *and* B *in their* state *facts. In all transition rules of A, we replace all facts of the form $\mathsf{state}^{\tau_A.TID}_{\mathsf{A:alice}.PC}[M_1, \ldots, M_n]$ by $\mathsf{state}^{\tau_A\langle\tau_C\rangle_{\mathsf{E}}.TID}_{\mathsf{A:alice}.PC}[\mathsf{KA}, \mathsf{KB}, M_1, \ldots, M_n]$ where τ_A and τ_C are the respective protocol types/names, and* KA *and* KB *are two new variables that do not occur in the specifications; analogously for* bob*. Every transmission of the form $\mathsf{net}_{(\mathsf{A},\mathsf{B})}(M)$ is replaced by $\mathsf{net}_{(\mathsf{A},\mathsf{B})}(\mathsf{scrypt}(\mathsf{KA}, M))$ and $\mathsf{net}_{(\mathsf{B},\mathsf{A})}(M)$ by $\mathsf{net}_{(\mathsf{A},\mathsf{B})}(\mathsf{scrypt}(\mathsf{KB}, M))$.*[5]

Ad 3. By assumption, the channel protocols are linear, and there is a highest PC for each role. We now consider for the highest PC PC_{alice} of alice*, and every transition rule of C that has on the RHS a local state fact of the form*

$$\mathsf{state}^{\tau_C.TID}_{\mathsf{A:alice}.PC_{\mathsf{alice}}}[KA, KB, M_1, \ldots, M_n]$$

where KA and KB will be composed terms. We consider further all initialization rules of the protocol A for role alice*, i.e., of the form*

$$\mathsf{agent}(\mathsf{A}).\mathsf{agent}(\mathsf{B}). \; =\![TID]\Rightarrow \; \mathsf{state}^{\tau_A.TID}_{\mathsf{A:alice}.pc}[\mathsf{B}]$$

We combine any such final state of C with any such initial state of A by a rule, basically replacing the initialization of the variables A *and* B *by their instances in the* state *fact of τ_C (where also* B *must occur within the messages by our assumptions):*

$$\mathsf{state}^{\tau_C.TID}_{\mathsf{A:alice}.PC_{\mathsf{alice}}}[\mathsf{KA}, \mathsf{KB}, M_1 \; \ldots, M_n].$$
$$\Rightarrow \mathsf{state}^{\tau_A\langle\tau_C\rangle_{\mathsf{E}}.TID}_{\mathsf{A:alice}.pc}[\mathsf{KA}, \mathsf{KB}, \mathsf{B}]$$

Here we have replaced the term KA of C by a variable KA *(i.e., the new protocol is oblivious to the precise structure of the key terms); also note that all temporary information of the channel protocol (the M_i except* B*) is purged in this transition.*

Example 7. For our running example, the $ca\langle dh\rangle_{\mathsf{E}}$ composition would give the following rules for role alice (we leave out role bob for brevity and the rules for dh which are identical):

$$\mathsf{state}^{\mathsf{dh.TID}}_{\mathsf{A:alice}.3}[\mathsf{KA}, \mathsf{KB}, \mathsf{B}] \Rightarrow \mathsf{state}^{\mathsf{ca}\langle\mathsf{dh}\rangle_{\mathsf{E}}.\mathsf{TID}}_{\mathsf{A:alice}.1}[\mathsf{KA}, \mathsf{KB}, \mathsf{B}]$$
$$\mathsf{state}^{\mathsf{ca}\langle\mathsf{dh}\rangle_{\mathsf{E}}.\mathsf{TID}}_{\mathsf{A:alice}.1}[\mathsf{KA}, \mathsf{KB}, \mathsf{B}] \; =\![\mathsf{M}]\Rightarrow$$
$$\mathsf{state}^{\mathsf{ca}\langle\mathsf{dh}\rangle_{\mathsf{E}}.\mathsf{TID}}_{\mathsf{A:alice}.2}[\mathsf{KA}, \mathsf{KB}, \mathsf{B}, \mathsf{M}].$$
$$\mathsf{net}_{(\mathsf{A},\mathsf{B})}(\mathsf{scrypt}(\mathsf{KA}, \mathsf{crypt}(\mathsf{pk}_{\mathsf{B}}, [\mathsf{ca}, \mathsf{M}]))).\mathsf{secret}(\mathsf{B}, \mathsf{M})$$

D. Realize: $A[C]_{\mathsf{R}}$ Composition

As mentioned above, the extended version of this paper [16] also considers the notion of abstract application protocols that run over abstract secure channels and a corresponding notion of *realize* composition, in which a concrete channel protocol is plugged in to realize the assumed secure channel. We omit it here, because it is similar to the embed composition.

[5] By construction, such net facts can only occur in the transition rules of alice and bob, so that the variables are always bound by their occurrence in state facts.

VI. Typing and Disjointness

In general, the kinds of composition of protocols we have introduced is not *sound*: if we compose secure protocols (that do not have an attack), we cannot be sure that their compositions are secure. The reason is that two protocols can have similar message formats (so that messages of one protocol can be confused for the other) with a different meaning. This is illustrated by the example in §II. There has been a lot of work on finding sufficient conditions for horizontal protocol composition, in particular the work on disjoint encryption, such as [11]. We point out that a different line of work is also very helpful here, namely on preventing type-flaw attacks, such as [22]. The idea here is that we could understand type-flaw attacks as something similar to the attacks against composition, because also they are basically a confusion of similar messages with a different meaning—only here it is within the same protocol rather than in different protocols. We sketch how to justify a typed model (and elaborate further on these ideas in the extended version [16]). We then consider in detail how to extend existing notions of disjointness so that they are sufficient for vertical composition.

A. Well-typed Messages

The basic idea of [22] is to require that all parts of a protocol are tagged with type information so that wherever the intruder cannot manipulate the value (e.g., inside an encryption) he cannot manipulate the type-interpretation either. As a consequence, whenever an attack exists, also a *well-typed attack* exists and thus it is sufficient to analyze protocols in a typed model. Still, we need to extend the result of [22] to include protocols that do not follow its tagging approach, but in which all non-atomic message parts are sufficiently disjoint. Basically, we require only that every two non-atomic parts of the message formats of a protocol will be different unless they have the same (intended) type. This extension is limited to free algebra. We leave the formal treatment of this idea to the extended version [16] and only state its theorem at this point:

Theorem 1 (Proof sketch in the extended version [16])**.** *If a format-type-safe protocol has an attack in the free algebra with the described intruder deduction rules, then it also has a well-typed attack, i.e., one where all sent and received messages have the intended types.*

Theorem 1 applies to a large class of protocols, namely all IF protocols (including composed ones) where the all variables can be consistently labeled with intended types. Still, because of its limitation to the free algebra, it does not apply to protocols based on algebraic properties, for instance Diffie-Hellman. From here on, we simply use a typed model as interface independent from its justification:

Precondition 3. *We expect that in transition rules of honest agents, all variables of incoming and outgoing messages are typed (basic or composed).*

In vertically composed protocols, the key terms of the channel encryption will have composed types. For instance, if a channel protocol establishes two keys $\mathsf{KA} = \mathsf{h}(\mathsf{NA}, \mathsf{NB})$ and $\mathsf{KB} = \mathsf{h}(\mathsf{NB}, \mathsf{NA})$, then the keys in transmissions have type $\mathsf{h}(\mathsf{nonce}, \mathsf{nonce})$ (while so far in our composition definition, these variables were simply untyped).

Note that variables of composed type can be replaced by terms with basic variables, e.g., $\mathsf{V} : [\mathsf{agent}, \mathsf{nonce}]$ can be replaced by $[\mathsf{V_A}, \mathsf{V_N}]$ where V_A and V_N are new variables of types agent and nonce, respectively. Thus we have now a model where honest agents accept only type-correct messages, even when they actually cannot check all parts.

B. Disjointness

Similar to our requirement that the different message formats within a single protocol are sufficiently disjoint, we now also require that for every pair of atomic protocols $\mathcal{P}$ the message formats must be disjoint. Note that this does *not* exclude the self-composition, such as $App\langle TLS\rangle_{\mathsf{E}}\langle TLS\rangle_{\mathsf{E}}$, but only prevent that the different atomic protocols (App and TLS in this example) have interferences with each other. Here we follow [9] and require disjoint message formats between the different messages of the protocols, quite similar to our previous property of format-type-safe in that we look at message formats of non-atomic subterms of message patterns. Recall that $MP(P)$ is specified in Def. 2 above to hold all the message patterns of protocol P, appropriately α-renamed. We now look at non-atomic sub-terms, in particular we will not require that dependent terms like pk_a (that we consider as atomic) must be disjoint between protocols. Additionally, because we later want to also look at vertical composition and not just at horizontal (parallel and sequential) composition like [9], we also introduce the notion that a protocol is *disjoint for its encryptions* meaning that adding several layers of symmetric encryption around a message pattern cannot lead to confusion with another message pattern. For instance, if $[\mathsf{A}, \mathsf{N}] : [\mathsf{agent}, \mathsf{nonce}] \in MP(P)$, then $\mathsf{scrypt}(\mathsf{k}, [\mathsf{A}', \mathsf{N}']) : \mathsf{scrypt}(\tau, [\mathsf{agent}, \mathsf{nonce}])$ cannot be allowed in $MP(P)$ as well (for any key term k of type τ), because that could lead to confusion with encryptions introduced by running the protocol over one of the secure channels; however simply an additional tag or just a change of format, e.g., $\mathsf{scrypt}(\mathsf{k}, [\mathsf{N}', \mathsf{A}']) : \mathsf{scrypt}(\tau, [\mathsf{nonce}, \mathsf{agent}])$ (for some $\tau : k$ that does not conflict with the rest), would be sufficient for our purposes.

Definition 5. *Let P be a protocol and $K_1, K_2, \ldots$ be variable symbols that do not occur in $MP(P)$. Then the*

message patterns with encryption of P *is defined as*

$$EMP(P) = \alpha(\bigcup_{n\in\mathbb{N}} EMP_n(P))$$
$$EMP_0(P) = MP(P)$$
$$EMP_{n+1}(P) = \alpha(\{\mathsf{scrypt}(K_{n+1}, m) \mid m \in EMP_n(P)\})$$

where $\alpha(M)$ indicates that the elements of M shall be α-renamed (without conflicts with the K_i) so that they have pairwise disjoint sets of variables.

P is called disjoint from its encryptions *iff there is no unifier between a members of $EMP_i(P)$ and $EMP_j(P)$ for $i \neq j$. Let $EST(P)$ be the non-atomic subterms of $EMP(P)$ again with α-renaming of variables as above. We require that all incoming and outgoing messages are non-atomic (because otherwise we cannot associate in general a unique protocol type to every message). We say that a set P_0 of IF protocols is* pairwise disjoint*, if every protocol $P \in P_0$ is disjoint from its encryptions and for every pair $P, P' \in P_0$ with $P \neq P'$, there is no unifier between any $t \in EST(P)$ and $t' \in EST(P')$.*

Example 8. For the ca example from Fig. 1, p. 2, for instance, we have (omitting the α renaming):

$$MP(ca) = \{\mathsf{crypt}(\mathsf{pk}_B, [ca, M])\}$$
$$EMP_1(ca) = \{\mathsf{scrypt}(K_1, \mathsf{crypt}(\mathsf{pk}_B, [ca, M]))\}$$
$$EMP_2(ca) = \{\mathsf{scrypt}(K_2, \mathsf{scrypt}(K_1, \mathsf{crypt}(\mathsf{pk}_B, [ca, M])))\}$$
$$EST(ca) = EMP(ca) \cup \{[ca, M]\}$$

ca is disjoint from its own encryptions, because there is no unifier between members of different $EMP_i(P)$ and $EMP_j(P)$. The derivation for dh is more complex (again we avoid the α renaming):

$$\begin{aligned} MP(dh) = \{\ & \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, \mathsf{exp}(g, X)]), \\ & \mathsf{crypt}(\mathsf{inv}_{\mathsf{pk}_A}, [dh, B, GX]), \ldots, \\ & \mathsf{scrypt}(h(\mathsf{exp}(GY, X), 0), keyack), \\ & \mathsf{scrypt}(h(\mathsf{exp}(GX, Y), 0), keyack) \ \} \end{aligned}$$
$$EMP_1(dh) = \{\mathsf{scrypt}(K_1, \mathsf{scrypt}(h(\ldots), keyack)), \ldots\}$$
$$\begin{aligned} EST(dh) = EMP(dh) \cup \\ \{[dh, B, \mathsf{exp}(g, X)], \mathsf{exp}(g, Y), [dh, B, GX], \ldots\} \end{aligned}$$

In addition, dh is disjoint from its own encryptions: here it is actually the tag $keyack$ and the fact that no other message of dh has symmetric encryption that prevent unification of messages of different EMP-levels. Also $EST(dh)$ and $EST(ca)$ do not have a unifier (even after proper α renaming) due to tagging and the fact that exp occurs in only one protocol.

Note that we do not exclude concatenation from the disjointness requirement as many other disjointness notions do. The reason for this is subtle. Consider a protocol in which the entire first message is a clear-text transmission, say $[A, B, N]$. Of course, the intruder can manipulate this message, and, consequently, the message can be left out of the disjointness notions for parallel composition: adding tags would be pointless since the intruder can change the tags anyway. In the vertical composition, however, where said protocol may be run over a channel, this message could suddenly be encrypted, and then it *is* important whether this term is disjoint from the formats of other protocols.

The inclusion of concatenation into the disjointness condition bears some subtleties, as well. For instance, consider two protocols where one contains the pair $[A, N]$ (as a subterm) and the other the pair $[B, M]$ (for variables A, B, N and M). These two protocols are then not disjoint. However, this is less restrictive than it seems: we consider concatenation of n-tuples as a family of operators ($n \in \mathbb{N}$) so that already concatenations of different length are considered as disjoint (e.g., $[A, N]$ is disjoint from $[B, C, M]$) and tags can be used—it is indeed a good idea in general if on each encryption layer there is a unique identifier of the protocol as part of which this message is meant.

Our disjointness definition is crucial for proving our compositionality results. Consider the following example: two protocols transmit each just one message from A to B, $h(N_1)$ and $h(N_2)$. These two are trivially not disjoint. However the variant $h([c_1, NA])$ and $h([c_2, NB])$ are for two constants $c_1 \not\approx c_2$.

Precondition 4. *From here on, we expect that the set of atomic protocols $\mathcal{P}$ is pairwise disjoint.*

This precondition is a key point of our paper: it allows us to prove a very helpful property, namely that every message that can be sent or received by an honest agent in any composition of $\mathcal{P}$-protocols has a *unique protocol type* as defined in §V-A. In fact, we named this concept "protocol types" because of its parallel with "message types", i.e., partitioning the space of messages.

Theorem 2 (Proved in the extended version [16])**.** *For composition $P \in \mathcal{C}$, every message pattern $m \in ST(P)$ has a* unique *protocol type $\tau \in T$. This implies that also all messages that an honest agent can send or receive as well as their non-atomic subterms all have a unique protocol type.*

The proof idea is that, if there were a message of two different types, then we could derive a violation of the disjointness condition. To carry over this result also to concrete messages (in an actual trace), we need Precondition 1 (all variables are well-typed), because otherwise agent variables can be replaced by arbitrary values (of any type).

VII. The Main Result

We are now able to put all the pieces together and give the main result of this paper:

Theorem 3. *Given a set $\mathcal{P}$ of atomic protocols that satisfies the four denoted preconditions, and all their syntactically correct compositions $\mathcal{C}$. Then every composed protocol in $\mathcal{C}$ is secure.*

The full proof is found in [16] and we sketch here only the key ideas. In fact, we first split the task into smaller theorems, basically one for each composition type. Part of the idea is the precise form of these theorems that does a great deal of the proof work, in particular by organizing a context where attack reductions can be given more easily.

To deal with this complicated form of the theorems more easily, we have organized them as a calculus. Each rule in the calculus has the form "if composition X is secure, then also composition Y is secure". The soundness of each rule is proved individually by one theorem (or by a simpler lemma). We then show that all compositions of $\mathcal{C}$ can be derived with the calculus to conclude the main result.

The most interesting rule of the calculus is perhaps the rule for embed composition:

$$\frac{A \parallel C \parallel \phi}{A\langle C\rangle_{\mathsf{E}} \parallel A \parallel C \parallel \phi}\ channel(C), app(A), A\langle C\rangle_{\mathsf{E}} \not\sqsubseteq \phi$$

This rule assumes that we have already proved the security of $A \parallel C \parallel \phi$ where A is a concrete application and C a channel protocol. ϕ can be any element of C, however ϕ must not already contain the type $A\langle C\rangle_{\mathsf{E}}$ that we want to compose now. If all this is true, then, the theorem tells us, adding $A\langle C\rangle_{\mathsf{E}}$ in parallel to what we already have is also secure.

Before we give a sketch why this reasoning is sound, let us first consider why this particular form to state the compositional reasoning result is helpful. First, suppose we had this theorem without the context $A \parallel C \parallel \phi$ as part of the conclusion; then, we would need extra rules that the composed protocol $A\langle C\rangle_{\mathsf{E}}$ can be used in parallel with A, C and other protocols (which is not implied by the security of $A\langle C\rangle_{\mathsf{E}}$). Thus, having this context as part of the conclusion is making this composition rule as general as possible. Second, on the premise side of the rule, the same context is also helpful (rather than requiring that A and C alone are secure): this allows to apply the theorem successively also to applications A that are themselves the result of a composition.

The idea of the proof of this composition theorem is indirect: if there is an attack against $A\langle C\rangle_{\mathsf{E}} \parallel A \parallel C \parallel \phi$, then there is one against $A \parallel C \parallel \phi$ alone. Again, here the context is very helpful in conducting the proof, because all the components of the composition are "available" and we can have them work similarly as in the composition. An essential point in the proof is that every message of the given attack can be uniquely attributed to a particular protocol type, thanks to Theorem 2. That theorem in turn is based on the fact that the atomic protocol of $\mathcal{P}$ are mutually disjoint and disjoint to their own encryptions. In particular, for a message of the form $\mathsf{scrypt}(K, M)$, either the entire message belongs to an atomic protocol or K is a key established by a unique channel protocol, and M's type can recursively determined again in a unique way. We can thus recognize all messages that belong to $A\langle C\rangle_{\mathsf{E}}$, either they belong to the key-exchange of the channel C or they are of the form $\mathsf{scrypt}(K, M)$ where M belongs to A and K was established with C. We show that the same attack works when replacing all such $\mathsf{scrypt}(K, M)$ messages with M itself—reducing it to an attack where only protocols A and C (and ϕ for other messages) are involved.

To conclude the exposition of our main result, observe that a great deal of the work lies in the setup and preparation we have made in the preceding sections, in particular allowing us to attribute every message to a unique application running over a uniquely identified stack of channels.

VIII. Related Work

Key exchange, secure channels and composition are both fundamental problems in formal methods as well as cryptography (KE [19], [20], [23], RSIM [24] and UC [25], UC KE and channels [26], [27]). And, there have been efforts to link both worlds with a relation to the protocol composition.

This work uses a symbolic representation of cryptographic primitives and, thus, focuses on formal methods literature on security analysis of key exchange and composition. Notably, Paulson established the security of the TLS key exchange by inductive analysis [3]. Composition problems have been researched in the symbolic model [11], [8], [28], [29], [9], [13], [10], [30], mostly analyzing parallel composition with limited or unlimited number of sessions.

We highlight the results of Cortier and Delaune [9], who show sufficient criteria for parallel composition. Their work relates to the result of Guttman and Thayer [11] that disjoint encryption can achieve protocol independence. Our work builds on both their insights: We use disjoint encryption to achieve secure protocol composition as well as adapt Cortier and Delaune's result on parallel composition to our setting. In addition, we benefit from the typing methods established by Heather et al. [22], which form a third ingredient to establish our composition results.

For sequential composition, Datta et. al. [8] have considered the construction of protocols from smaller sub-protocols and analyzed parallel and sequential composition of protocols, yet without considering keys. Escobar et al. [30] extend the Maude-NPA tool by sequential composition primitives, thus allow for an automated composition analysis by Maude-NPA's unification and backwards search method. Ciobâcă and Cortier [10] show that parallel and sequential composition of symbolic representations of protocols is secure if they use primitives with disjoint signatures and if the individual protocols are secure, even if they share data. Similarly to our method, Ciobâcă and Cortier obtain a composition theorem by reduction of attack traces against the composition to attack traces against the composed protocols. This work is in fact the closest to ours as they can consider handshake protocols establishing a key and using this key in another protocol. Note however that the use of this key (and thus the

channel) needs to be part of that application protocol (i.e., there is no vertical composition of an abstract application with a channel). The disjointness assumptions here exclude vertical self-composition: the composed protocols must be distinguishable and thus Ciobâcă's and Cortier's result cannot support several layers of the same protocol (like TLS) without some modification (like disjoint tags).

For vertical composition, we see that Gao et al. [31] considered the vertical stacking of protocols, yet did not establish a composition theorem. Mödersheim and Viganò [13] provide criteria vertical protocol composition as part of their research on secure pseudonymous channels. Even though they provide sufficient criteria and proof for general vertical composition, their theorems do not extend to self-composition as analyzed in this paper.

Guttman [32] considers a general concept of protocol transformations and argues that many kinds of protocol composition can be seen as instances of this transformation concept. As far as we can see, vertical protocol composition can indeed be seen as protocol transformation, as well. Due to the generality of Guttman's concept, however, the soundness requirements are (hard-to-check) semantical conditions. Guttman suggests that one may find easy-to-check conditions based on disjoint encryption. This would mean similar limitations as in the case of [10], i.e., self-composition of exactly the same protocol is excluded, but some distinction like different tags would be necessary.

Beyond these results, there have been research efforts to link the formal and the cryptographic world, where one reconciliatory impulse came from Abadi and Rogaway [33]. Cortier and Delaune [34] research formal methods for proving observational equivalence and conclude that observational equivalence implies computational indistinguishability in face of an active adversary. Cortier and Warinschi [35] pursue computationally sound automated proofs. Backes, Pfitzmann and Waidner [36] established a set of composable symbolic primitives that are computationally sound in the RSIM framework [37], for which Sprenger et al. [38] modelled a theorem proving theory in Isabelle/HOL.

IX. Conclusion

We showed for any suite of protocols satisfying our sufficient criteria that every vertical composition of its protocols is secure. This holds for vertical compositions of arbitrary depth as well as for self-compositions. The sufficient criteria that we employ are well-founded on earlier works in this field: We require disjointness of message formats as discussed by [11], [9], parallel composition of atomic protocols as established by [9], and a strong type model as introduced by [22]. From these foundations, we derive a composition theorem that holds for arbitrary goals on auxiliary predicates and positive intruder knowledge. Our preconditions can be statically checked and are liberal enough to cover a large class of protocols, such as the TLS and IPsec.

Acknowledgment

The authors are grateful for the dedicated support and guidance of Joshua Guttman. The authors thank Veronique Cortier and him for the insightful comments and discussions.

References

[1] G. Lowe, "Breaking and fixing the needham-schroeder public-key protocol using fdr," in *TACAS*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 1055. Springer, 1996, pp. 147–166.

[2] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The avispa tool for the automated validation of internet security protocols and applications," in *CAV*, ser. Lecture Notes in Computer Science, K. Etessami and S. K. Rajamani, Eds., vol. 3576. Springer, 2005, pp. 281–285.

[3] L. C. Paulson, "Inductive analysis of the internet protocol TLS," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 332–351, 1999.

[4] N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov, "Undecidability of bounded security protocols," in *Proc. of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, Jul. 1999.

[5] M. Rusinowitch and M. Turuani, "Protocol insecurity with a finite number of sessions, composed keys is np-complete," *Theor. Comput. Sci.*, vol. 1-3, no. 299, pp. 451–475, 2003.

[6] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. Cape Breton, Nova Scotia, Canada: IEEE Computer Society, Jun. 2001, pp. 82–96.

[7] S. Escobar, C. Meadows, and J. Meseguer, "Maude-npa: Cryptographic protocol analysis modulo equational properties," in *FOSAD*, ser. Lecture Notes in Computer Science, A. Aldini, G. Barthe, and R. Gorrieri, Eds., vol. 5705. Springer, 2007, pp. 1–50.

[8] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic, "Secure protocol composition," in *FMSE*, M. Backes and D. A. Basin, Eds. ACM, 2003, pp. 11–23.

[9] V. Cortier and S. Delaune, "Safely composing security protocols," *Formal Methods in System Design*, vol. 34, no. 1, pp. 1–36, 2009.

[10] S. Ciobâca and V. Cortier, "Protocol composition for arbitrary primitives," in *CSF*. IEEE Computer Society, 2010, pp. 322–336.

[11] J. D. Guttman and F. J. Thayer, "Protocol independence through disjoint encryption," in *CSFW*, 2000, pp. 24–34.

[12] T. Dierks and C. Allen, "RFC 2246: The TLS protocol," Jan. 1999, status: Standards Track. [Online]. Available: ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt

[13] S. Mödersheim and L. Viganò, "Secure pseudonymous channels," in *ESORICS*, ser. Lecture Notes in Computer Science, M. Backes and P. Ning, Eds., vol. 5789. Springer, 2009, pp. 337–354.

[14] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, "Transport layer security (TLS) renegotiation indication extension," http://tools.ietf.org/html/rfc5746, Feb. 2010, http://tools.ietf.org/html/rfc5746.

[15] M. Ray and S. Dispensa, "Renegotiating TLS," http://www.phonefactor.com/sslgapdocs/Renegotiating_TLS.pdf, Nov. 2009, http://www.phonefactor.com/sslgapdocs/Renegotiating_TLS.pdf.

[16] T. Groß and S. Mödersheim, "Vertical protocol composition (extended version)," IBM Research, IBM Research Report RZ3803, Apr. 2011, http://domino.research.ibm.com/library/cyberdig.nsf/index.html.

[17] AVISPA, "The Intermediate Format," Automated Validation of Internet Security Protocols and Applications (AVISPA), Deliverable D2.3, 2003, http://www.avispa-project.org/delivs/2.3/d2-3.pdf.

[18] F. Baader and T. Nipkow, *Term Rewriting and All That*. Cambridge University Press, 1998.

[19] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *93*, D. R. Stinson, Ed., vol. 773, 1994, pp. 232–249.

[20] ——, "Provably secure session key distribution — the three party case," in *Proceedings of the 27th Annual Symposium on Theory of Computing (STOC)*. ACM Press, May 1995, pp. 57–66.

[21] G. Lowe, "A hierarchy of authentication specifications." IEEE Computer Society Press, 1997, pp. 31–43.

[22] J. Heather, G. Lowe, and S. Schneider, "How to prevent type flaw attacks on security protocols," *Journal of Computer Security*, vol. 11, no. 2, pp. 217–244, 2003.

[23] V. Shoup, "On formal models for secure key exchange," IBM Research, Research Report RZ 3120 (#93166), Apr. 1999, version 4, November 1999, available from http://www.shoup.net/papers/.

[24] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission," Oakland, CA, May 2001, pp. 184–200.

[25] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," IACR Cryptology ePrint Archive, ePrint Report 2000/067, 2000, http://eprint.iacr.org/.

[26] R. Canetti and H. Krawczyk, "Universally composable notions of key exchange and secure channels," Report 2002/059, May 2002.

[27] S. Gajek, M. Manulis, O. Pereira, A.-R. Sadeghi, and J. Schwenk, "Universally composable security analysis of TLS—secure sessions with handshake and record layer protocols," IACR Cryptology ePrint Archive, ePrint Report 2008/251, 2008, http://eprint.iacr.org/.

[28] A. Datta, A. Derek, J. C. Mitchell, and A. Roy, "Protocol composition logic (pcl)," *Electr. Notes Theor. Comput. Sci.*, vol. 172, pp. 311–358, 2007.

[29] J. D. Guttman, "Cryptographic protocol composition via the authentication tests," in *FOSSACS*, ser. Lecture Notes in Computer Science, L. de Alfaro, Ed., vol. 5504. Springer, 2009, pp. 303–317.

[30] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, "Sequential protocol composition in maude-npa," in *ESORICS*, ser. Lecture Notes in Computer Science, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds., vol. 6345. Springer, 2010, pp. 303–318.

[31] H. Gao, F. Nielson, and H. R. Nielson, "Protocol stacks for services," in *Proc. of the Workshop on Foundations of Computer Security (FCS)*, Jul. 2009.

[32] J. D. Guttman, "Security goals and protocol transformations," in *Theory of Security and Applications (TOSCA'11)*, 2011, to appear.

[33] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (the computational soundness of formal encryption)," *J. Cryptology*, vol. 20, no. 3, p. 395, 2007.

[34] V. Cortier and S. Delaune, "A method for proving observational equivalence," in *CSF*. IEEE Computer Society, 2009, pp. 266–276.

[35] V. Cortier and B. Warinschi, "Computationally sound, automated proofs for security protocols," in *ESOP*, ser. Lecture Notes in Computer Science, S. Sagiv, Ed., vol. 3444. Springer, 2005, pp. 157–171.

[36] M. Backes, B. Pfitzmann, and M. Waidner, "A universally composable cryptographic library," IACR, Cryptology ePrint Archive Report 2003/015, Jan. 2003. [Online]. Available: http://eprint.iacr.org/2003/015

[37] ——, "The reactive simulatability (RSIM) framework for asynchronous systems," IACR, Cryptology ePrint Archive Report 2004/082, 2004, http://eprint.iacr.org/. [Online]. Available: http://eprint.iacr.org/2004/082

[38] C. Sprenger, M. Backes, D. A. Basin, B. Pfitzmann, and M. Waidner, "Cryptographically sound theorem proving," in *CSFW*. IEEE Computer Society, 2006, pp. 153–166.