Composition Theorems Without Pre-Established Session Identifiers

Ralf Küsters
University of Trier
Trier, Germany
kuesters@uni-trier.de

Max Tuengerthal
University of Trier
Trier, Germany
tuengerthal@uni-trier.de

ABSTRACT

Canetti's universal composition theorem and the joint state composition theorems by Canetti and Rabin are useful and widely employed tools for the modular design and analysis of cryptographic protocols. However, these theorems assume that parties participating in a protocol session have pre-established a unique session ID (SID). While the use of such SIDs is a good design principle, existing protocols, in particular real-world security protocols, typically do not use pre-established SIDs, at least not explicitly and not in the particular way stipulated by the theorems. As a result, the composition theorems cannot be applied for analyzing such protocols in a modular and faithful way.

In this paper, we therefore present universal and joint state composition theorems which do not assume pre-established SIDs. In our joint state composition theorem, the joint state is an ideal functionality which supports several cryptographic operations, including public-key encryption, (authenticated and unauthenticated) symmetric encryption, MACs, digital signatures, and key derivation. This functionality has recently been proposed by Küsters and Tuengerthal and has been shown to be realizable under standard cryptographic assumptions and for a reasonable class of environments. We demonstrate the usefulness of our composition theorems by several case studies on real-world security protocols, including IEEE 802.11i, SSL/TLS, SSH, IPsec, and EAP-PSK. While our applications focus on real-world security protocols, our theorems, models, and techniques should be useful beyond this domain.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*

General Terms

Security, Verification

1. INTRODUCTION

Universal composition theorems, such as Canetti's composition theorem in the UC model [7] and Küsters' composition theorem in the IITM model [19], allow to obtain security for multiple sessions of a protocol by analyzing just a single protocol session. These theorems assume that different protocol sessions have disjoint state; in particular, each session has to use fresh randomness. This can lead to inefficient and impractical protocols, since, for example, every session has to use fresh long-term symmetric and public/private keys. Canetti and Rabin [11] therefore proposed to combine universal composition theorems with what they called composition theorems with joint state. As the name suggests, such theorems yield systems in which different sessions may use some joint state, e.g., the same long-term and public/private keys.

However, these theorems, both for universal and joint state composition, assume that parties participating in a protocol session have pre-established a unique session ID (SID), and as a result (see Section 3), make heavy use of this SID in a specific way stipulated by the universal and joint state composition theorems. On the one hand, the use of such SIDs is a good design principle and as discussed by Canetti [8] and Barak et al. [2] establishing such SIDs is simple. On the other hand, many existing protocols, including most real-world security protocols, do not make use of such pre-established SIDs, at least not explicitly and not in the particular way stipulated by the theorems. As a result, these theorems cannot be used for the faithful modular analysis of such protocols; at most for analyzing idealized variants of the original protocols, which is unsatisfactory and risky, in the sense that attacks on the original protocols might be missed (see Section 4). The problems resulting from pre-established SIDs in the existing composition theorems do not seem to have been brought out in previous work.

The goal of this paper is therefore to obtain general universal composition and joint state composition theorems that do not assume pre-established SIDs and their use in cryptographic protocols, and hence, to enable modular, yet faithful analysis of protocols, without the need to modify/idealize these protocols. A main motivation for our work comes from the analysis of real-world security protocols. While many attacks on such protocols have been uncovered (see, e.g., [12, 15, 1, 27, 25] for recent examples), their comprehensive analysis still poses a big challenge, as often pointed out in the literature (see, e.g., [26, 18, 10]). A central problem is the complexity of these protocols. In order to tame the complexity, *modular* analysis of such protocols should be pushed as far as possible, but without giving up on accurate modeling. Our composition theorems are useful tools for this kind of modular and faithful analysis. They should be of interest also beyond the analysis of real-world security protocols. More precisely, the main contributions of our work are as follows:

Contribution of this Paper. Our universal composition theorem without pre-established SIDs states that if a protocol realizes an ideal functionality for a single session, then it also realizes the ideal

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

functionality for multiple sessions, subject to mild restrictions on the single-session simulator. The important point is that a user invokes a protocol instance simply with a *local* SID, locally chosen and managed by the user herself, rather than with an SID pre-established with other users for that session. This not only provides the user with a more common and convenient interface, where the user addresses her protocol instances with the corresponding local SIDs, but, more importantly, as explained in Section 3, frees the real protocol from the need to use pre-established SIDs and allows for realizations that faithfully model existing (real-world) protocols.

In our joint state composition theorem without pre-established SIDs we consider protocols that use an ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ proposed in [23]. The functionality $\mathcal{F}_{\text{crypto}}$ allows its users to perform several cryptographic operations in an ideal way, including public-key encryption, authenticated and unauthenticated symmetric encryption, MACs, digital signatures, key derivation, and establishing pre-shared keys. As shown in [23], $\mathcal{F}_{\text{crypto}}$ can be realized under standard cryptographic assumptions, subject to reasonable restrictions on the environment. Now, our joint state composition theorem states that under a certain condition on the protocol, which we call *implicit (session) disjointness*, it suffices to show that the protocol (which may use $\mathcal{F}_{\text{crypto}}$) realizes an ideal functionality for a single session of the protocol to obtain security for multiple sessions of the protocol, where all sessions may use the *same* ideal crypto functionality $\mathcal{F}_{\text{crypto}}$; again we put mild restrictions on the single-session simulator. So, $\mathcal{F}_{\text{crypto}}$ (or its realization), with the keys stored in it, constitutes the joint state across sessions. As in the case of the universal composition theorem, users again invoke protocol instances with locally chosen and managed SIDs. Unlike joint state composition theorems with pre-established SIDs, our joint state composition theorem does not modify/idealize the original protocol.

Given our theorems, (real-world) security protocols can be analyzed with a high degree of modularity and without giving up on precision: Once implicit disjointness is established for a protocol—first proof step—, it suffices to carry out single-session analysis for the protocols—second proof step—in order to obtain multi-session security with joint state for the original protocol, not just an idealized version with pre-established SIDs added in various places, as explained in Sections 3 and 4. We emphasize that, due to the use of $\mathcal{F}_{\text{crypto}}$, in all proof steps often merely information-theoretic or purely syntactical reasoning, without reasoning about probabilities and without reduction proofs, suffices.

We demonstrate the usefulness of our theorems and approach by several case studies on real-world security protocols, namely (subprotocols of) IEEE 802.11i, SSL/TLS, SSH, IPsec, and EAP-PSK. More precisely, we show that all these protocols satisfy implicit disjointness, confirming our believe that this property is satisfied by many (maybe most) real-world security protocols. While proving implicit disjointness requires to reason about multiple sessions of a protocol, this step is nevertheless relatively easy. In fact, as demonstrated by our case studies, to prove implicit disjointness, typically the security properties of only a fraction of the primitives used in a protocol need to be considered. For example, to prove that the SSH key exchange protocol satisfies implicit disjointness, only collision resistance of the hash function is needed, but not the security of the encryption scheme, the MAC, or the Diffie-Hellman key exchange used in SSH. Now since the above mentioned protocols satisfy implicit disjointness, to show that these protocols are secure key exchange or secure channel protocols, single-session analysis suffices. Performing this single-session analysis for all these protocols is out of the scope of this paper. (The main point of this paper is to provide the machinery for faithful and highly modular analysis, not to provide a full-fledged analysis of these protocols.) However, for some of the mentioned protocols single-session analysis has been carried out in other works (see Section 5). For example, this has been done for SSL/TLS by Gajek et al. in [16]. However, they used the original joint state theorem to lift their security result to the multi-session case, resulting in an idealized version of SSL/TLS (see Section 5.2). With our theorems and since SSL/TLS satisfies implicit disjointness, multi-session security follows for the original, unmodified protocol.

Structure of this Paper. In Section 2, we recall basics on simulation-based security and introduce some notation. Our universal composition and joint state composition theorems without pre-established SIDs are then presented in Sections 3 and 4, respectively, with applications discussed in Section 5. Full details and proofs are provided in our technical report [22].

2. SIMULATION-BASED SECURITY

We briefly recall the framework of simulation-based security, following [19] (see also [22]). We provide a quite model-independent account as the details of the model are not important to be able to follow the rest of this paper.

The General Computational Model. The general computational model is defined in terms of systems of interactive Turing machines. An interactive Turing machine (shortly, machine) is a probabilistic polynomial-time Turing machine with named input and output tapes. The names determine how different machines are connected in a system of machines. A *system* $\mathcal{S}$ of machines is of the form $\mathcal{S} = M_1 \mid \cdots \mid M_k \mid !M'_1 \mid \cdots \mid !M'_{k'}$ where the M_i and M'_j are machines such that the names of input tapes of different machines in the system are disjoint. We say that the machines M'_j are in the scope of a bang operator. This operator indicates that in a run of a system an unbounded number of (fresh) copies of a machine may be generated. Conversely, machines which are not in the scope of a bang operator may not be copied. Systems in which multiple copies of a machine may be generated are often needed, e.g., in case of multi-party protocols or in case a system describes the concurrent execution of multiple instances of a protocol. In a run of a system $\mathcal{S}$ at any time only one machine is active and all other machines wait for new input. A (copy of a) machine M can trigger another (copy of a) machine M' by sending a message on an output tape corresponding to an input tape of M'. Identifiers, e.g., session or party identifiers, contained in the message can be used to address a specific copy of M'. If a new identifier is used, a fresh copy of M' would be generated. The first machine to be triggered is the so-called master machine. This machine is also triggered if a machine does not produce output. A run stops if the master machine does not produce output or a machine outputs a message on a tape named decision. Such a message is considered to be the overall output of the system. Systems will always have polynomial runtime in the security parameter (and possibly the length of auxiliary input).

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called *indistinguishable* ($\mathcal{P} \equiv \mathcal{Q}$) iff the difference between the probability that $\mathcal{P}$ outputs 1 (on the decision tape) and the probability that $\mathcal{Q}$ outputs 1 is negligible in the security parameter.

Notions of Simulation-Based Security. We need the following terminology. For a system $\mathcal{S}$, the input/output tapes of machines in $\mathcal{S}$ that do not have a matching output/input tape are called *external*. These tapes are grouped into *I/O* and *network tapes*. We consider three different types of systems, modeling i) real and ideal protocols/functionalities, ii) adversaries and simulators, and iii) en-

vironments: *Protocol systems* and *environmental systems* are systems which have an I/O and network interface, i.e., they may have I/O and network tapes. *Adversarial systems* only have a network interface. Environmental systems may contain a master machine and may produce output on the decision tape. We can now define strong simulatability; other equivalent security notions, such as (dummy) UC, can be defined in a similar way [19].

DEFINITION 1 ([19]). *Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems with the same I/O interface, the real and the ideal protocol, respectively. Then, $\mathcal{P}$* realizes *$\mathcal{F}$ ($\mathcal{P} \leq \mathcal{F}$) iff there exists an adversarial system $\mathcal{S}$ (a simulator or an ideal adversary) such that the systems $\mathcal{P}$ and $\mathcal{S}|\mathcal{F}$ have the same external interface and for all environmental systems $\mathcal{E}$, connecting only to the external interface of $\mathcal{P}$ (and hence, $\mathcal{S}|\mathcal{F}$), it holds that $\mathcal{E}|\mathcal{P} \equiv \mathcal{E}|\mathcal{S}|\mathcal{F}$.*

Composition Theorems. We restate the composition theorems from [19], in a slightly simplified way. The first composition theorem handles concurrent composition of a fixed number of protocol systems. The second one guarantees secure composition of an unbounded number of copies of a protocol system.

THEOREM 1 ([19]). *Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{F}_1, \mathcal{F}_2$ be protocol systems such that $\mathcal{P}_1$ and $\mathcal{P}_2$ as well as $\mathcal{F}_1$ and $\mathcal{F}_2$ only connect via their I/O interfaces and $\mathcal{P}_i \leq \mathcal{F}_i$, for $i \in \{1, 2\}$. Then, $\mathcal{P}_1|\mathcal{P}_2 \leq \mathcal{F}_1|\mathcal{F}_2$.*

Let $\mathcal{F}$ and $\mathcal{P}$ be protocol systems, which, for example, describe one session of an ideal/real protocol. By $\underline{\mathcal{F}}$ and $\underline{\mathcal{P}}$ we denote the so-called session versions of $\mathcal{F}$ and $\mathcal{P}$, which allow an environment to address different sessions of $\mathcal{F}$ and $\mathcal{P}$, respectively, in the multi-session versions $!\underline{\mathcal{F}}$ and $!\underline{\mathcal{P}}$ of $\mathcal{F}$ and $\mathcal{P}$ by prefixing messages with session identifiers (SIDs); an instance of $\underline{\mathcal{P}}/\underline{\mathcal{F}}$ is accessed via a unique SID. Conversely, messages output by $\underline{\mathcal{F}}$ and $\underline{\mathcal{P}}$ will be prefixed by their respective SID.

THEOREM 2 ([19]). *Let $\mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{P} \leq \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq !\underline{\mathcal{F}}$.*

These theorems can be applied iteratively to construct more and more complex systems. For example, using that $\leq$ is reflexive, we obtain, as a corollary of the above theorems, that for any protocol system $\mathcal{Q}$: $\mathcal{P} \leq \mathcal{F}$ implies $\mathcal{Q}|\,!\underline{\mathcal{P}} \leq \mathcal{Q}|\,!\underline{\mathcal{F}}$, i.e., $\mathcal{Q}$ using an unbounded number of copies of $\mathcal{P}$ realizes $\mathcal{Q}$ using an unbounded number of copies of $\mathcal{F}$. This corollary is in the spirit of Canetti's universal composition theorem [7].

3. UNIVERSAL COMPOSITION WITHOUT PRE-ESTABLISHED SIDS

Universal composition theorems, such as Theorem 2 and the composition theorem by Canetti, allow to obtain security for multiple sessions of a protocol by analyzing just a single session. Such theorems can therefore greatly simplify protocol analysis. However, these theorems rely on the setup assumption that the parties participating in a protocol session agree upon a unique SID and that they invoke their instance of the protocol with that SID. This is due to the way multi-session versions of ideal functionalities are defined in these composition theorems: A multi-session version of an ideal functionality F is such that parties which want to access an instance of F have to agree on a unique SID in order to be able to all invoke the same instance of F with that SID. As a consequence, the composition theorems implicitly require that a session of a real protocol with SID s realizes a session of the ideal functionality with SID s. (For example, if a session of the real protocol consists of two instances, e.g., an initiator instance and a responder instance, then the initiator with SID s and the responder with SID s together have to realize the ideal functionality with SID s.) This, in turn, implies that the real protocol has to use the SID s in some way, since otherwise there is nothing that prevents grouping instances with different SIDs (e.g., an initiator with SID s and a responder with SID s') into one session. One usage of the SID s is, for example, to access a resource for the specific session, e.g., a functionality (with SID s) that provides the parties with fresh keys or certain communication channels for that specific session. In realizations with joint state, s is typically used in *all* messages exchanged between parties in order to prevent interference with other sessions (see also Section 4).

Canetti [8] discusses three methods of how such unique SIDs could be established, including a method proposed by Barak et al. [2], where parties simply exchange nonces in clear and then form a unique SID by concatenating these nonces and the party names. We will refer to such uniquely established SIDs (using whatever method) by *pre-established* SIDs. The use of pre-established SIDs is certainly a good design principle. However, assuming pre-established SIDs and, as a result, forcing their use in the protocols greatly limits the scope of the composition theorems for the analysis of existing protocols. In particular, they cannot be used for the modular analysis of real-world security protocols since such protocols typically do not make use of SIDs in this explicit and specific way. In other words, the composition theorems could only be used to analyze idealized/modified versions of such protocols. However, this is dangerous: While the idealized/modified version of a protocol might be secure, its original version may not be secure (see Section 4). We note that, alternatively, protocols could of course directly be analyzed in the multi-session settings, instead of first analyzing the single-session setting and then lifting the analysis to the multi-session setting by a composition theorem. But this kind of analysis would be more involved and would not exploit the potential of modular analysis, which for the comprehensive analysis of complex protocols, such as real-world security protocols, is essential.

In this section, we therefore present a general universal composition theorem that does not assume pre-established SIDs (and their use in protocols). For this purpose, we first provide a new definition of the multi-session version of an ideal functionality. Our new multi-session version models the more realistic scenario that a party accesses an instance of an ideal functionality F simply by a *local* SID, which is locally chosen and managed by the party itself. It is then left to an adversary (simulator) to determine which group of local sessions may use one instance of F, where the grouping into what we call a (global) session is subject to certain restrictions (see below). This seemingly harmless modification not only provides a more realistic and common interface to the functionality (and its realization), but, as explained above, more importantly frees the realization from the need to use pre-established SIDs and allows for realizations that faithfully model existing (real-world) protocols. Before presenting the new multi-session versions of ideal functionalities and our composition theorem, we fix some notation and terminology for modeling arbitrary real protocols.

3.1 Multi-Session Real Protocols

A multi-session real protocol is an arbitrary real protocol with n roles, for some $n \geq 2$, which may use arbitrary subprotocols/functionalities to perform its tasks. More precisely, a *multi-session (real) protocol* $\mathcal{P}$ is a protocol system of the form $\mathcal{P} = !M_1 | \cdots |\, !M_n$ for some $n \geq 2$ and machines $M_1, \ldots, M_n$. Each machine M_r represents one role in the protocol and, since these machines are under the scope of a bang operator, there can be multiple instances of each machine in a run of the system (see below). Every machine M_r has

i) an I/O input and output tape for communication with the environment (users), ii) a network input and output tape for communication with the adversary (modeling the network), and iii) an I/O input and output tape for communication with a subprotocol/ideal functionality $\mathcal{F}$. We require that the I/O interface of $\mathcal{F}$ consists of n pairs of I/O input and output tapes, one for each role. Note that $\mathcal{F}$ may include several subprotocols/functionalities. We say that $\mathcal{P}$ *uses* $\mathcal{F}$.

A machine M_r expects inputs to be prefixed with a tuple of the form $(lsid, p)$ from the environment (user) and the adversary, and it prefixes all messages it outputs with $(lsid, p)$. Intuitively, p is a party identifier (PID) and $lsid$ a *local* SID (LSID), locally chosen and managed by party p. In a run of $\mathcal{P}$ there will be at most one instance of M_r with ID $(lsid, p)$, representing the local session $lsid$ of party p in role r.

To model corruption, we assume that every (instance of) M_r stores a flag **corrupted** $\in \{\mathsf{false}, \mathsf{true}\}$ in its state, which initially is false. At some point, M_r might set it to true in which case we call M_r *corrupted*. We require that once M_r sets the flag to true, it stays true. Furthermore, whenever the environment sends the message $(lsid, p, \mathsf{Corrupted?})$ to M_r (on the I/O tape), M_r replies with $(lsid, p, \mathsf{Corrupted}, \textbf{corrupted})$. This allows the environment to know which instances are corrupted. (As usual, this is necessary in simulation-based settings.) However, we do not fix how M_r behaves when corrupted; we leave this entirely up to the definition of M_r. One possible behavior could be, for example, that when corrupted, M_r gives complete control to the adversary by forwarding all messages between the environment and the adversary. We note that the possibility of corrupting single instances of M_r is quite fine-grained and allows several forms corruption, including complete corruption of a party: the adversary can simply corrupt all instances of that party.

3.2 A New Multi-Session Version of Ideal Functionalities

Let F be any machine, modeling an ideal functionality, with n pairs of input and output I/O tapes, one for each role, and one pair of input and output network tapes. We now define the new multi-session version of F, informally explained above. We call this new multi-session version *multi-session local-SID (ideal) functionality* and denote it by $\mathcal{F}[F]$ or simply $\mathcal{F}$:

A *user* of $\mathcal{F}$ is identified within $\mathcal{F}$ by the tuple $(p, lsid, r)$, where p is a party identifier (PID), $r \leq n$ a role, and $lsid$ a *local* SID (LSID), which can be chosen and managed by the party itself. In particular, on the tape for role r, $\mathcal{F}$ expects requests to be prefixed by tuples of the form $(lsid, p)$, and conversely, $\mathcal{F}$ prefixes answers sent on that tape with a tuple of the form $(lsid, p)$.

A user of $\mathcal{F}$, say $(p, lsid, r)$, can initiate a session by sending a *session-start* message of the form $(lsid, p, \mathsf{Start}, m)$ where m is an optional bit string, which can be used to set parameters of the session, e.g., the desired partners or the name of a key distribution server. For example, in the case of two-party key exchange, a user with PID p who wants to exchange a key with party p' could set $m = (p, p')$. We emphasize that the interpretation of m is left to F. Upon such a *session-start* request, $\mathcal{F}$ records this request (if it is the first such request from $(p, lsid, r)$) as a *local session* for user $(p, lsid, r)$ and forwards this information to the adversary.

The adversary (simulator) determines to which *global* session local sessions belong, by sending a *session-create* message of the form (Create, sid) to $\mathcal{F}$ where $sid = (p_1, lsid_1, 1), \ldots, (p_n, lsid_n, n)$ contains one local session for every role. (We note that $\mathcal{F}$ could easily be extended to deal with multiple local sessions per role.) The machine $\mathcal{F}$ only accepts such a message if it is consistent with the local sessions: The mentioned local sessions all exist, are not corrupted (see below), and are not already part of another global session. If $\mathcal{F}$ accepts such a *session-create* message, $\mathcal{F}$ internally creates a new instance of F identifying it by sid. Then, $\mathcal{F}$ sends the message $(\mathsf{Create}, m_1, \ldots, m_n)$ to this instance of F where, for all $r \leq n$, m_r is the optional bit string contained in the *session-start* message of user $(p_r, lsid_r, r)$. The adversary can address this instance of F (via the network interface) by prefixing messages with sid; conversely, messages output by F on its network interface are prefixed with sid.

For a message m of a user $(p, lsid, r)$, which is not a *session-start* message or a message of the form Corrupted? (see below), $\mathcal{F}$ checks whether $(p, lsid, r)$ is part of a global session. If not, $\mathcal{F}$ drops m, i.e., this message is ignored. Otherwise, $\mathcal{F}$ forwards m to the corresponding instance of F. Conversely, $\mathcal{F}$ forwards all messages from an instance of F on tape r to the corresponding user in role r.

We model corruption as follows in $\mathcal{F}$. The adversary can send a *corrupt* message of the form $(\mathsf{Corrupt}, (p, lsid, r))$ for a local session $(p, lsid, r)$ to $\mathcal{F}$. The machine $\mathcal{F}$ only accepts this message if the local session $(p, lsid, r)$ is not already part of a global session, and in this case records $(p, lsid, r)$ as *corrupted*. For every corrupted local session, $\mathcal{F}$ forwards all messages from a user of that local session to the adversary (instead of forwarding them to F) and vice versa. This models that the adversary has full control over corrupted local sessions. Note that once a local session is part of a global session, the local session or its corresponding global session can still be corrupted at any time according to the way corruption is defined in F, which we do not restrict.

Finally, a user may ask $\mathcal{F}$ whether or not a local session was corrupted before being part of a global session by sending the message $(lsid, p, \mathsf{Corrupted?})$. Then, $\mathcal{F}$ replies accordingly with true or false. This, as usual, guarantees that the environment is aware of who is corrupted, preventing the simulator from simply corrupting all local sessions. Whether or not a user can ask about the corruption status of F is completely up to the definition of F, which, again, we do not restrict.

We note that the above definition of a multi-session version of an ideal functionality applies to any ideal functionality F.

Technically, for a real protocol to realize a multi-session local-SID functionality the simulator must be able to group instances of the simulated real protocol into a global session before interaction with the functionality F is possible. This means that a real protocol needs to allow for the grouping of instances by whatever mechanism (where the mechanisms is typically intertwined with the rest of the protocol). In particular, the grouping is part of the protocol, and hence, can now be precisely modeled and analyzed. For example, for authentication, key exchange, secure channel protocols and the like, being able to tell which instances are grouped together is an essential part of what these protocols (have to) guarantee and different protocols use different mechanisms; these mechanisms should be part of the analysis. Conversely, before there was one fixed mechanism for grouping instances, namely pre-established SIDs. Real protocols needed to make sure that they in fact can be grouped according to the SID they obtained, and hence, they had to use the SID in some essential way. Moreover, the SIDs came from outside of the protocol, and hence, their establishment was not part of the protocol.

3.3 The Universal Composition Theorem Without Pre-Established SIDs

We now present our universal composition theorem. Let $\mathcal{P}$ be a multi-session protocol using a multi-session local-SID functionality $\mathcal{F}'$ and let $\mathcal{F}$ be a multi-session local-SID functionality. In-

formally, our theorem states that if $\mathcal{P}|\mathcal{F}'$ realizes $\mathcal{F}$ in the single-session setting, then $\mathcal{P}|\mathcal{F}'$ realizes $\mathcal{F}$ in the multi-session setting. The important point here is that, by the definition of multi-session local-SID functionalities, no pre-established SIDs (nor their use in the protocol) are required.

To formulate this theorem, we consider a machine F_{single} that is put on top of $\mathcal{P}|\mathcal{F}'$ and $\mathcal{F}$, respectively, and that allows an environment to create *at most one* session, i.e., only one user is allowed per role. We note that an alternative to using F_{single} would be to restrict the environment explicitly.

To be able to prove the composition theorem, we need to restrict the class of simulators used to prove that $\mathcal{P}|\mathcal{F}'$ realizes $\mathcal{F}$ in the single-session case. For this purpose, we define the following simulation relation: We say that $\mathcal{P}|\mathcal{F}'$ *single-session realizes* $\mathcal{F}$ (denoted by $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \leq^* F_{\text{single}}|\mathcal{F}$) if i) $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \leq F_{\text{single}}|\mathcal{F}$, i.e., according to Definition 1, there exists a simulator *Sim* such that for all $\mathcal{E}$ it holds that $\mathcal{E}|F_{\text{single}}|\mathcal{P}|\mathcal{F}' \equiv \mathcal{E}|Sim|F_{\text{single}}|\mathcal{F}$, and ii) *Sim* is a machine which operators in two stages as follows: In the first stage, *Sim* simply emulates the system $\mathcal{P}|\mathcal{F}'$, where *session-start* messages from $\mathcal{F}$ are forward to the emulated $\mathcal{P}$. If *Sim* receives a) a *session-create* message for the emulated $\mathcal{F}'$ from the adversary and this message is accepted by $\mathcal{F}'$ or b) the **corrupted** flag of an emulated instance M_r in $\mathcal{P}$ is set to true, then *Sim* enters its second stage. Once in the second stage, *Sim* is not restricted whatsoever. If, in the first stage, the emulated $\mathcal{P}|\mathcal{F}'$ produces I/O output, then *Sim* terminates. (In this case the simulation fails.) This is in fact not a restriction: Every protocol that produces I/O output if *Sim* is in its first stage would not realize $\mathcal{F}$, i.e., $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \not\leq F_{\text{single}}|\mathcal{F}$. The reason is that in the first stage, the instances in $\mathcal{P}$ run independently. Now, if an environment emulated all but one instance, in $F_{\text{single}}|\mathcal{F}$ no session would be created, and hence, no output at the I/O interface would be produced.

So, altogether the only restriction we put on *Sim* is that it emulates the real protocol in its first stage. This is what simulators would typically do anyway. In fact, we think that for most applications $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \leq F_{\text{single}}|\mathcal{F}$ implies $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \leq^* F_{\text{single}}|\mathcal{F}$.

Moreover, our restriction seems unavoidable in order to prove our composition theorem. First recall that for the classical universal composition theorems (Theorem 2 and Canetti's composition theorem) the proof is by a hybrid argument. In the i-th hybrid system the environment emulates the first $< i$ sessions as real protocols (real sessions) and the last $> i$ sessions as ideal (single-session simulator plus ideal functionality). The i-th session is external. Since every session is identified by a pre-established SID, the environment knows exactly and from the start on which instances of machines form one session. In particular, it knows from the start on whether a session should be emulated as real or ideal and which messages must be relayed to the external session. In our setting, this does not work since we do not assume pre-established SIDs: Initially, the (hybrid) environment does not know to which session an instance $(p, lsid, r)$ will belong. In particular, it does not know whether it will belong to an ideal or real session. This is only determined if $(p, lsid, r)$ is included in a (valid) *session-create* message to $\mathcal{F}'$. So unless an instance $(p, lsid, r)$ does not behave the same in the ideal and real session up to this point, consistent simulation would not be possible. Now, by our assumption that the simulator in its first stage simulates the real protocol, the environment can first simulate the real protocol for the instance $(p, lsid, r)$. Once this instance is included in a (valid) *session-create* message to $\mathcal{F}'$, and hence, the environment knows whether the instance belongs to an ideal or real session, the simulation can be continued accordingly. More concretely, if it turns out that $(p, lsid, r)$ belongs to an ideal session, the environment starts the emulation of the simulator for that session with the current configurations of all emulated instances for that session. Again, this is possible because up to this point the simulator too would have only simulated these instance as real protocols. For the i-th session, the environment guesses the instances that shall belong to it. Following this idea, we proved our composition theorem stated next.

THEOREM 3. *Let $\mathcal{F}$ and $\mathcal{F}'$ be two multi-session local-SID functionalities and let $\mathcal{P}$ be a multi-session real protocol that uses $\mathcal{F}'$. If $F_{\text{single}}|\mathcal{P}|\mathcal{F}' \leq^* F_{\text{single}}|\mathcal{F}$, then $\mathcal{P}|\mathcal{F}' \leq \mathcal{F}$.*

We note that Theorem 3 can be applied iteratively: For example, if we have that $F_{\text{single}}|\mathcal{P}_1|\mathcal{F}_1 \leq^* F_{\text{single}}|\mathcal{F}_2$ and $F_{\text{single}}|\mathcal{P}_2|\mathcal{F}_2 \leq^* F_{\text{single}}|\mathcal{F}_3$, then, by Theorem 3 and Theorem 1, $\mathcal{P}_2|\mathcal{P}_1|\mathcal{F}_1 \leq \mathcal{F}_3$.

4. JOINT STATE COMPOSITION WITHOUT PRE-ESTABLISHED SIDS

Universal composition theorems, such as Theorem 2 and the composition theorem of Canetti, assume that different protocol sessions have disjoint state; in particular, each session has to use fresh randomness. (Theorem 3 makes this assumption too, but we exclude this theorem from the following discussion since it does not assume pre-established SIDs.) This can lead to inefficient and impractical protocols, since, for example, in every session fresh long-term symmetric and public/private keys have to be used. Canetti and Rabin [11] therefore proposed to combine the universal composition theorems with what they called composition theorems with joint state. By now, joint state composition theorems for several cryptographic primitives are available, including joint state composition theorems for digital signatures [11, 20] and public-key encryption [20] as well as encryption with long-term symmetric keys [21]. These theorems provide mechanisms that allow to turn a system with independent sessions (i.e., sessions with disjoint state) into a system where the same (long-term symmetric and public/private) keys may be used in different sessions. This joint state comes "for free" in the sense that it does not require additional proofs. However, there is a price to pay: Just as the universal composition theorems, the joint state composition theorems assume pre-established SIDs. Moreover, the mechanisms used by existing joint state theorems for specific cryptographic primitives, such as encryption and digital signatures, prefix *all* plaintexts to be encrypted (with long-term symmetric or public/private keys) and messages to be signed by the unique pre-established SIDs; by this, interference between different sessions is prevented. While this is a good design principle, these theorems are unsuitable for the modular analysis of an existing protocol that does not employ these mechanisms: If such a protocol is secure in the single-session setting, then its multi-session version obtained by combining universal composition with joint state composition, and hence, the version of the protocol in which messages are prefixed with pre-established SIDs, is secure as well. But from this it does in general not follow that the original protocol, which may be drastically different, is also secure in the multi-session setting. In fact, by the above joint-state constructions insecure protocols can be turned into secure ones (see Figure 1). In particular, since real-world security protocols typically do not use pre-established SIDs, at least not explicitly and not in the particular way stipulated by the theorems, the joint state composition theorems are unsuitable for the modular and faithful analysis of such protocols; at most idealized/modified protocols, but not the original real-world protocols, can be analyzed in this modular way. For example, in Step 3 of the TLS Handshake Protocol (see Figure 2), the client sends the pre-master key encrypted to the server. In the variant of TLS obtained by the joint state theorems, a unique SID

		original	*modified*
1.	$A \to B$:	$\{\!\|N_A, A\|\!\}_{k_B}$	$\{\!\|sid, N_A, A\|\!\}_{k_B}$
2.	$B \to A$:	$\{\!\|N_A, N_B\|\!\}_{k_A}$	$\{\!\|sid, N_A, N_B\|\!\}_{k_A}$
3.	$A \to B$:	$\{\!\|N_B\|\!\}_{k_B}$	$\{\!\|sid, N_B\|\!\}_{k_B}$

Figure 1: The original Needham-Schroeder Public-Key Protocol is insecure [24]. Its modified version, resulting from the joint-state construction, which prefixes every plaintext with a pre-established SID *sid* is secure (see [22] for details).

sid would be included in the plaintext as well. By this alone, unlike the original version of TLS, this message is bound to session *sid*.

In this section, we therefore propose a joint state composition theorem which does not require to modify the protocol under consideration. In particular, it does not rely on pre-established SIDs and the mechanism of prefixing messages with such SIDs.

In our joint state theorem we consider a multi-session real protocol $\mathcal{P}$ which uses an ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ proposed in [23]. The functionality $\mathcal{F}_{\text{crypto}}$ allows its users to perform the following operations in an ideal way: i) generate symmetric keys, including pre-shared keys, ii) generate public/private keys, iii) derive symmetric keys from other symmetric keys, iv) encrypt and decrypt messages and ciphertexts, respectively (public-key encryption and both unauthenticated and authenticated symmetric encryption are supported), v) compute and verify MACs and digital signatures, and vi) generate fresh nonces. All symmetric and public keys can be part of plaintexts to be encrypted under other symmetric and public keys. Derived keys can be used just as freshly generated symmetric keys. As shown in [23], $\mathcal{F}_{\text{crypto}}$ can be realized under standard cryptographic assumptions, subject to natural restrictions on the environment. We briefly recall $\mathcal{F}_{\text{crypto}}$ and its realization in Section 4.1.

Every instance of a machine M_r in $\mathcal{P}$ has access to $\mathcal{F}_{\text{crypto}}$. In other words, $\mathcal{F}_{\text{crypto}}$ is the joint state of all sessions of $\mathcal{P}$: Different sessions may have access to the same public/private and symmetric keys in $\mathcal{F}_{\text{crypto}}$.

Now, informally speaking, our joint state composition theorem states that under a certain condition on $\mathcal{P}$, which we call *implicit (session) disjointness*, it is sufficient to analyze $\mathcal{P}$ (which may use $\mathcal{F}_{\text{crypto}}$) in the single-session setting to obtain security in the multi-session setting, where all sessions may use the *same* ideal crypto functionality $\mathcal{F}_{\text{crypto}}$. (We note that by the universal composition theorem, $\mathcal{F}_{\text{crypto}}$ can be replaced by its realization.) It seems that most real-world protocols satisfy implicit disjointness and that this property can be verified easily, as illustrated by our case studies in Section 5.

In what follows, we briefly recall the ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ and its realization. We then introduce the notion of implicit disjointness and present our joint state composition theorem.

4.1 The Ideal Crypto Functionality

We now briefly recall the ideal functionality $\mathcal{F}_{\text{crypto}}$, proposed in [23], which, as mentioned, supports several cryptographic operations. The formulation here is slightly modified (see below).

Description of $\mathcal{F}_{\text{crypto}}$. Just as multi-session local-SID functionalities introduced in Section 3, $\mathcal{F}_{\text{crypto}}$ is parametrized by a number n of roles. For every role, $\mathcal{F}_{\text{crypto}}$ has one I/O input and output tape. Again, a user of $\mathcal{F}_{\text{crypto}}$ is identified within $\mathcal{F}_{\text{crypto}}$ by a tuple $(p, lsid, r)$, where p is a PID, *lsid* a LSID, and r a role.

Users of $\mathcal{F}_{\text{crypto}}$, and its realization, do not get their hands on the actual symmetric keys stored in the functionality, but only on pointers to these keys, since otherwise no security guarantees could be provided; users obtain the actual public keys though. A user can perform the operations mentioned above (encryption, etc.). Upon a key generation request, an adversary can corrupt a key, which is then marked "known" in $\mathcal{F}_{\text{crypto}}$ (see below). A user can ask whether a key one of her pointers points to is corrupted.

The functionality $\mathcal{F}_{\text{crypto}}$ keeps track of which user has access to which symmetric keys (via pointers) and which keys are known to the environment/adversary, i.e., have been corrupted or have been encrypted under a known key, and as a result became known. For this purpose, $\mathcal{F}_{\text{crypto}}$ maintains a set $\mathcal{K}$ of all symmetric keys stored within $\mathcal{F}_{\text{crypto}}$, a set $\mathcal{K}_{\text{known}} \subseteq \mathcal{K}$ of *known* keys, and a set $\mathcal{K}_{\text{unknown}} := \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ of *unknown* keys.

To illustrate the internal behavior of $\mathcal{F}_{\text{crypto}}$ and to point out the mentioned modification to the original version of $\mathcal{F}_{\text{crypto}}$, we sketch the behavior of $\mathcal{F}_{\text{crypto}}$ for authenticated encryption and decryption, with requests (Enc, ptr, x) and (Dec, ptr, y): We first consider the case that *ptr* points to an unknown key, i.e., a key in $\mathcal{K}_{\text{unknown}}$. The plaintext x may contain pointers to symmetric keys. Before x is actually encrypted, such pointers are replaced by the keys they refer to, resulting in a message x'. Now, not the actual message, but a random message of the same length is encrypted. If this results in a ciphertext y', then the pair (x', y') is stored in $\mathcal{F}_{\text{crypto}}$ and y' is returned to the user. Decryption of y succeeds only if exactly one pair of the form (x'', y) is stored. In this case, x'' with embedded keys replaced by pointers is returned. In case *ptr* points to a key marked known, i.e., a key in $\mathcal{K}_{\text{known}}$, the adversary is asked for a ciphertext (in case of encryption) or a plaintext (in case of decryption).[1] Furthermore, all keys contained in x' are marked known as they are encrypted under a known key.

Realization of $\mathcal{F}_{\text{crypto}}$. In [23], a realization $\mathcal{P}_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$ has been proposed based on standard cryptographic assumptions on schemes: IND-CCA2-secure schemes for public-key and unauthenticated symmetric encryption, an IND-CPA- and INT-CTXT-secure scheme for authenticated symmetric encryption, UF-CMA-secure MAC and digital signature schemes, and pseudo-random functions for key derivation. These schemes are used to realize $\mathcal{F}_{\text{crypto}}$ in the expected way. To show that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$ it is necessary to restrict the environment: Environments should not cause the so-called commitment problem (once an unknown symmetric key was used for encryption, it should not become known) and should not generate key cycles; without these restrictions, much stronger cryptographic assumptions would be necessary, which go beyond what is typically assumed for the security of real-world security protocols. Protocols (in particular, real-world security protocols) using $\mathcal{F}_{\text{crypto}}$ typically satisfy these restrictions and this is easy to verify for a given protocol, as discussed and illustrated in [23].

4.2 Our Criterion: Implicit Disjointness

We now introduce the notion of *implicit (session) disjointness*, already mentioned at the beginning of Section 4. Recall that we are interested in the security of the system $\mathcal{P}|\mathcal{F}_{\text{crypto}}$, where $\mathcal{P}$ is a multi-session protocol in which all sessions may use the same $\mathcal{F}_{\text{crypto}}$. As explained before, implicit disjointness is a condition on $\mathcal{P}$ which should allow to analyze the security of $\mathcal{P}$ in a single-session setting in order to obtain security of $\mathcal{P}$ in a multi-session setting, without assuming pre-established SIDs and without modifying $\mathcal{P}$. Intuitively, implicit disjointness is a condition that en-

[1] This constitutes a slight modification to the original ideal functionality in [23], where in this case encryption and decryption were performed with algorithms previously provided by the adversary. The new version helps in the proof of our joint state theorem. It is just as useful for analyzing protocols and can be realized in exactly the same way as the original version.

sures that different sessions of $\mathcal{P}$ cannot "interfere", even though they share state, in the form of information stored in $\mathcal{F}_{\text{crypto}}$, including public/private and pre-shared keys, and the information stored along with these keys, e.g., plaintext-ciphertext pairs. In order to define the notion of implicit disjointness, we first introduce some notation and terminology.

Partnering Functions. In the definition of implicit disjointness, we assume the existence of a partnering function[2] which groups users $(p, lsid, r)$, more precisely, the corresponding instances of machines M_r in a run of $\mathcal{P}$, into sessions. Formally, a *partnering function* τ for $\mathcal{P}|\mathcal{F}_{\text{crypto}}$ is a polynomial-time computable, partial function that maps every sequence α of configurations of an instance of a machine M_r in $\mathcal{P}$ to an SID (which is an arbitrary bit string) or $\perp$. For every environment $\mathcal{E}$, (partial) run ρ of $\mathcal{E}|\mathcal{P}|\mathcal{F}_{\text{crypto}}$, and every user $(p, lsid, r)$, we define $\tau_{(p,lsid,r)}(\rho) := \tau(\alpha)$ where α is the projection of ρ to the sequence of configurations of the machine M_r with PID p and LSID $lsid$. We say that $(p, lsid, r)$ and $(p', lsid', r')$ *are partners* (or *belong to the same session*) in a (partial) run ρ if $\tau_{(p,lsid,r)}(\rho) = \tau_{(p',lsid',r')}(\rho) \neq \perp$.

We say that τ is *valid* for $\mathcal{P}$ if, for every environment $\mathcal{E}$ for $\mathcal{P}|\mathcal{F}_{\text{crypto}}$, the following holds with overwhelming probability (the probability is taken over runs ρ of $\mathcal{E}|\mathcal{P}|\mathcal{F}_{\text{crypto}}$): For every user $(p, lsid, r)$ in ρ, the following conditions are satisfied. i) Once an SID is assigned, it is fixed, i.e., if $\tau_{(p,lsid,r)}(\rho'') \neq \perp$, then it holds $\tau_{(p,lsid,r)}(\rho') = \tau_{(p,lsid,r)}(\rho'')$ for every prefix ρ' of ρ and every prefix ρ'' of ρ'. ii) Corrupted users do not belong to sessions, i.e., if $(p, lsid, r)$ is corrupted in ρ (i.e., the flag **corrupted** is set to true in the corresponding instances of M_r), then $\tau_{(p,lsid,r)}(\rho) = \perp$. iii) Every session contains at most one user per role, i.e., for every partner $(p', lsid', r')$ of $(p, lsid, r)$ in ρ, it holds that $r \neq r'$ or $(p', lsid', r') = (p, lsid, r)$.

In practice, partnering functions are typically very simple. In our case studies (Section 5), we use conceptually the same partnering function for all protocols; basically partners are determined based on the exchanged nonces.

Construction and Destruction Requests. We call an encryption, MAC, and sign request (for $\mathcal{F}_{\text{crypto}}$ by an instance M_r of $\mathcal{P}$, i.e., a user) a *construction* request and a decryption, MAC verification, and signature verification request a *destruction* request.

Now, roughly speaking, implicit disjointness says that whenever some user sends a destruction request, then the user who sent the "corresponding" construction request belongs to the same session according to τ. This formulation is, however, too strong. For example, an adversary could send a ciphertext coming from one session to a different session where it is successfully decrypted. But further inspection of the plaintext might lead to the rejection of the message (e.g., because excepted nonces did not appear or MAC/signature verification failed). We therefore need to introduce the notion of a *successful* destruction request. For this purpose, we also introduce what we call *tests*.

Tests and Successful Destruction Requests. We imagine that a user $(p, lsid, r)$ (more precisely, the corresponding instance of M_r) after every destruction request runs some deterministic algorithm `test` which outputs *accept* or *reject*, where, besides the response received, the run of `test` may depend on and may even modify the state of $(p, lsid, r)$. We require that `test` satisfies the following conditions: If the destruction request is a MAC/signature verification request, then `test` simply outputs the result of the verification. If the destruction request is a decryption request, but decryption failed (i.e., $\mathcal{F}_{\text{crypto}}$ returned an error message), then `test` returns *reject*. Otherwise, if decryption did not fail, and hence, a plaintext was returned, `test` is free to output *accept* or *reject*. In the latter case—reject—, we require the state of $(p, lsid, r)$ to be the same as if decryption had failed (i.e., as if $\mathcal{F}_{\text{crypto}}$ had returned an error message) in the first place; this ensures that the state of $(p, lsid, r)$ does not depend on the plaintext that was returned. The algorithm `test` may itself make destruction requests (but no construction requests), e.g., decrypt nested ciphertexts or verify embedded MACs/signatures, which are subject to the same constraints. Also, key generation and key derivation are allowed within a test. The requirements on `test` reflect what protocols typically do (see Section 5.2 for an example).

Now, we say that a destruction request is *accepted* if the test performed after the request returns *accept*. We say that it is *ideal* if the key used in the destruction request is marked unknown in $\mathcal{F}_{\text{crypto}}$ or is an uncorrupted public/private key in $\mathcal{F}_{\text{crypto}}$ and, in case of a decryption request, the ciphertext in that request is stored in $\mathcal{F}_{\text{crypto}}$ (and hence, it was produced by $\mathcal{F}_{\text{crypto}}$ and the corresponding stored plaintext is returned).

Correspondence Between Construction and Destruction Requests. We now define when a construction request corresponds to a destruction request. Let ρ be a run of the system $\mathcal{E}|\mathcal{P}|\mathcal{F}_{\text{crypto}}$ and let m_c and m_d be construction and destruction request, respectively, such that m_c was sent by some instance to $\mathcal{F}_{\text{crypto}}$ before m_d was sent by some (possibly other) instance to $\mathcal{F}_{\text{crypto}}$ in ρ. Then, we say that m_c *corresponds* to m_d in ρ if i) m_c is an encryption and m_d a decryption request under the same key (for public-key encryption/decryption, under corresponding public/private keys) such that the ciphertext in the response to m_c from $\mathcal{F}_{\text{crypto}}$ coincides with the ciphertext in m_d, ii) m_c is a MAC/signature and m_d a MAC/signature verification request under the same key/corresponding keys such that the message in m_c coincides with the message in m_d (the MACs/signatures do not need to coincide).

Explicitly Shared (Symmetric) Keys. For implicit disjointness, we only impose restrictions on what we call explicitly shared (symmetric) keys. These are pre-shared symmetric keys or keys (directly or indirectly) derived from such keys in different sessions with the same seed. We note that in most protocols pre-shared keys are the only explicitly shared keys since derived keys are typically derived from seeds that are unique to the session.

Definition 2 (implicit disjointness). *Let $\mathcal{P}$ be a multi-session protocol that uses $\mathcal{F}_{\text{crypto}}$ and τ be a valid partnering function for $\mathcal{P}|\mathcal{F}_{\text{crypto}}$. Then, $\mathcal{P}$ satisfies* implicit (session) disjointness w.r.t. τ *if for every environment $\mathcal{E}$ for $\mathcal{P}|\mathcal{F}_{\text{crypto}}$ the following holds with overwhelming probability for runs ρ of $\mathcal{E}|\mathcal{P}|\mathcal{F}_{\text{crypto}}$:*

(a) Every explicitly shared key is either always marked unknown or always marked known in $\mathcal{F}_{\text{crypto}}$.

(b) Whenever some user $(p, lsid, r)$ (i.e., an instance of M_r) performed an accepted and ideal destruction request with an explicitly shared key or a public/private key at some point in ρ, say after the partial run ρ', then there exists some user $(p', lsid', r')$ that has sent a corresponding construction request such that both users are partners or both users are corrupted in ρ'.

Most protocols can easily be seen to satisfy (a) because explicitly shared keys are typically not sent around (i.e., encrypted by other keys), and hence, since they can be corrupted upon generation only, they are either corrupted (i.e., always known) or always unknown. As already mentioned, our case studies (Section 5) demonstrate that

[2]The concept of partnering functions has been used to define security in game-based definitions, which led to discussions whether the obtained security notions are reasonable [4, 5, 3, 9, 13, 17]. Here, we use partnering functions as part of our *criterion* (implicit disjointness) but not as part of the security definition itself; security means realizing an ideal functionality (see Theorem 4).

(b) too is typically satisfied by real-world protocols and can easily be checked. We note that (b) can be interpreted as a specific correspondence assertion, and it might be possible to check (b) using automated techniques, such as CryptoVerif [6].

4.3 The Joint State Composition Theorem Without Pre-Established SIDs

In this section, we present our joint state composition theorem. To be able to prove this theorem, we need to restrict the class of simulators used to prove that $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ realizes $\mathcal{F}$ in the single-session case. For this purpose, similarly to Section 3.3, we define the following simulation relation, where τ is a valid partnering function for $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ and $\mathcal{F}$ is a multi-session local-SID functionality: We say that $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ *single-session realizes* $\mathcal{F}$ *w.r.t.* τ (denoted by $F_{\text{single}}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq^{\tau} F_{\text{single}}\,|\,\mathcal{F}$) if i) $F_{\text{single}}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq F_{\text{single}}\,|\,\mathcal{F}$, i.e., according to Definition 1, there exists a simulator Sim_{τ} such that for all $\mathcal{E}$ it holds that $\mathcal{E}\,|\,F_{\text{single}}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \equiv \mathcal{E}\,|\,Sim_{\tau}\,|\,F_{\text{single}}\,|\,\mathcal{F}$, and ii) Sim_{τ} is a machine which operators in two stages: Analogously to the simulators defined in Section 3.3, in the first stage Sim_{τ} emulates the system $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$. Just as in Section 3.3, Sim_{τ} enters its second stage, in which Sim_{τ} is unrestricted, if an emulated instance of M_r in $\mathcal{P}$ set its **corrupted** flag to true. In Section 3.3, simulators also entered the second stage if a *session-create* message (addressed to $\mathcal{F}'$) was received. Such messages do not occur here. Instead, whenever activated, Sim_{τ} computes $\tau(\alpha_r)$ for all $r \leq n$, where α_r is the current sequence of configurations of the emulated instance of M_r. If τ signals a session, i.e., $\tau(\alpha_1) = \cdots = \tau(\alpha_n) \neq \perp$, then Sim_{τ} enters its second stage, in which it is unrestricted.

Analogously to Section 3.3, we can observe that the only restriction we put on Sim_{τ} is that it emulates the real protocol in its first stage. As already argued in Section 3.3, this appears to be unavoidable and does not seem to be a restriction in practice.

We are now ready to present our joint state composition theorem, with $\mathcal{F}_{\text{crypto}}$ serving as the joint state. Since our theorem does not assume pre-established SIDs, protocols analyzed using this theorem do not need to be modify/idealize by prefixing SIDs to messages. The usage of our theorem is discussed in more detail in Section 5.

THEOREM 4. *Let $\mathcal{F}$ be a multi-session local-SID functionality and let $\mathcal{P}$ be a multi-session protocol that uses $\mathcal{F}_{\text{crypto}}$ and satisfies implicit disjointness w.r.t. τ.*

If $F_{\text{single}}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq^{\tau} F_{\text{single}}\,|\,\mathcal{F}$, then $\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq \mathcal{F}$.

PROOF SKETCH. We first construct a machine $\mathcal{Q}_{\tau}$ which simulates $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ except that it uses a different copy of $\mathcal{F}_{\text{crypto}}$ for every session (according to τ). Using implicit disjointness, we can show that $\mathcal{E}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \equiv \mathcal{E}\,|\,\mathcal{Q}_{\tau}$ for every environment $\mathcal{E}$. We then show that $\mathcal{Q}_{\tau}$ realizes $\mathcal{F}$, using $F_{\text{single}}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq^{\tau} F_{\text{single}}\,|\,\mathcal{F}$. □

5. APPLICATIONS

In this section, we discuss, using key exchange and secure channels as an example, how Theorems 3 and 4 can be used to analyze protocols in a modular and faithful way. While our discussion focuses on the analysis of properties of real-world security protocols, our theorems should be useful beyond this domain.

5.1 Proving Security of Key Exchange and Secure Channel Protocols

We consider a standard secure channel ideal functionality F_{sc} and an ideal functionality $F_{\text{key-use}}$ for key usability. The latter functionality, which is inspired by the notion of key usability proposed in [14], is new and of independent interest. It is very similar to a standard key exchange functionality. However, parties do not obtain the actual exchanged key but only a pointer to this key. They can then use this key to perform *ideal* cryptographic operations, e.g., encryption, MACing, key derivation, etc., similarly to $\mathcal{F}_{\text{crypto}}$. Compared to the standard key exchange functionality, $F_{\text{key-use}}$ has two big advantages: i) One can reason about the session key (and keys derived from it) still in an *ideal* way, which greatly simplifies the analysis when used in higher level protocols. ii) $F_{\text{key-use}}$ can be realized by protocols which use the session key in the key exchange, e.g., for key confirmation. In what follows, let $\mathcal{F}_{\text{sc}} = \mathcal{F}[F_{\text{sc}}]$ and $\mathcal{F}_{\text{key-use}} = \mathcal{F}[F_{\text{key-use}}]$ denote the multi-session local-SID functionalities of F_{sc} and $F_{\text{key-use}}$, respectively.

To illustrate the use of Theorems 3 and 4, consider, for example, the task of proving that a multi-session protocol $\mathcal{Q}$ which is based on a multi-session key exchange protocol $\mathcal{P}$ realizes $\mathcal{F}_{\text{sc}}$, where both $\mathcal{Q}$ and $\mathcal{P}$ could be real-world security protocols.

While a proof from scratch would, similarly to proofs in a game-based setting, require involved reduction arguments and would be quite complex, using our framework the proof is very modular, with every proof step being relatively small and simple: First, instead of using the actual cryptographic schemes, $\mathcal{P}$ can use $\mathcal{F}_{\text{crypto}}$ (at least for the operations supported by $\mathcal{F}_{\text{crypto}}$). As a result, for the rest of the proof merely information-theoretic reasoning is needed, often not even probabilistic reasoning, in particular no reduction proofs (at least as far as the operations supported by $\mathcal{F}_{\text{crypto}}$ are concerned). The remaining proof steps are to show: i) $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ satisfies implicit disjointness, ii) $\mathcal{P}^{\mathcal{F}_{\text{crypto}}}$ single-session realizes $\mathcal{F}_{\text{key-use}}$, and iii) $\mathcal{Q}^{\mathcal{F}_{\text{key-use}}}$ single-session realizes $\mathcal{F}_{\text{sc}}$. (Since, the session key established by $\mathcal{F}_{\text{key-use}}$ can be used for ideal cryptographic operations, the argument for Step iii) is still information-theoretic.) We note that only Step i) needs some (information-theoretic) reasoning on multiple sessions, but only to show implicit disjointness. This is easy, as illustrated by our case studies (see below); the proof often merely needs to consider the security properties of a small fraction of the primitives used in the protocol. Now, by i), ii), and Theorem 4, we obtain $\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq \mathcal{F}_{\text{key-use}}$. Theorem 3 and iii) imply $\mathcal{Q}^{\mathcal{F}_{\text{key-use}}} \leq \mathcal{F}_{\text{sc}}$. By Theorem 1 and since $\mathcal{Q} \leq \mathcal{Q}$, we have $\mathcal{Q}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq \mathcal{Q}\,|\,\mathcal{F}_{\text{key-use}}$, and hence, $\mathcal{Q}\,|\,\mathcal{P}^{\mathcal{F}_{\text{crypto}}} \leq \mathcal{F}_{\text{sc}}$ by transitivity of $\leq$.

5.2 Case Studies

In our case studies (see [22] for details), we consider real-world key exchange protocols, namely IEEE 802.11i, SSH, SSL/TLS, IPsec, and EAP-PSK. We show that these protocols, for which we model the cryptographic core, satisfy implicit disjointness (see below); we also give an example of a (secure) protocol, namely the Needham-Schroeder-Lowe Public-Key Protocol, that does not satisfy implicit disjointness. Step iii) (see above), and hence, with Theorem 3, also $\mathcal{Q}^{\mathcal{F}_{\text{key-use}}} \leq \mathcal{F}_{\text{sc}}$, is proved for a generic secure channel protocol $\mathcal{Q}$ of which many real-world protocols are instances (see [22]). Providing full proofs for Step ii) for the key exchange protocols of our case studies is beyond the scope of this paper. However, ii) partly follows from existing work, from [23] for IEEE 802.11i and from [16] for SSL/TLS. For example, in [16] Gajek et al. showed single-session security of TLS; they use the joint state composition theorem by Canetti and Rabin to obtain security in the multi-session setting, which, however, as discussed only proves security of a modified/idealized version of TLS (see the remarks on TLS at the beginning of Section 4 and Figure 2). Using our theorems and the fact that TLS satisfies implicit disjointness, the result by Gajek et al. now also implies security of the (original) version of TLS in the multi-session setting, without pre-established SIDs prefixed to all plaintexts and signed messages.

		original	*modified*
1.	$C \to S$:	c_1, N_C	c_1, N_C
2.	$S \to C$:	$S, k_S, c_2, N_S,$	$S, k_S, c_2, N_S,$
3.	$C \to S$:	$C, k_C, \{\!\|PMS\|\!\}_{k_S}, \mathrm{sig}_{k_C}(handshake),$ $\{F(MS, c_3 \| handshake)\}_{IKCS,EKCS}$	$C, k_C, \{\!\|sid, PMS\|\!\}_{k_S}, \mathrm{sig}_{k_C}(sid, handshake),$ $\{F(MS, c_3 \| handshake)\}_{IKCS,EKCS}$
4.	$S \to C$:	$\{F(MS, c_4 \| handshake)\}_{IKSC,EKSC}$	$\{F(MS, c_4 \| handshake)\}_{IKSC,EKSC}$

Figure 2: The TLS Handshake Protocol (Key Exchange Method: RSA) and its modified version.

For SSL/TLS and SSH, we now show that they satisfy implicit disjointness; for details and the proofs for other protocols see [22].

Implicit Disjointness of SSL/TLS. The cryptographic core of the TLS Handshake Protocol with RSA encryption is depicted in Figure 2 on the left (we consider the variant where the client authenticates itself using digital signatures): N_C and N_S are nonces generated by C and S, respectively; the premaster secret *PMS* is chosen randomly by C and is encrypted under the public key of S ($\{\!|PMS|\!\}_{k_S}$); $c_0, \ldots, c_4$ are distinct constants; F is a pseudo-random function; the master secret *MS* is derived from *PMS* as follows: $MS = F(PMS, c_0 \| N_C \| N_S)$; $\{m\}_{k_1,k_2}$ denotes MAC-then-encrypt, i.e., $\{m\}_{k_1,k_2} = \{m, \mathrm{mac}_{k_1}(m)\}_{k_2}$; the symmetric encryption and MAC keys *EKCS*, *EKSC*, *IKCS*, *IKSC* are derived from *MS* using F and the nonces N_C and N_S as a seed; *handshake* stands for the concatenation of all previous messages, that is, $handshake = c_1 \| N_C \| S \| k_S \| c_2 \| N_S \| C \| k_C \| \{\!|PMS|\!\}_{k_S}$. In Step 3 of the protocol, the server performs the following test (as soon as a check fails, the whole message is dropped): It first decrypts the first ciphertext (using $\mathcal{F}_{\text{crypto}}$). If successful, it checks that the signature is over the expected message. If so, it verifies the signature $\mathrm{sig}_{k_C}(handshake)$ (using $\mathcal{F}_{\text{crypto}}$). If successful, S derives the keys *MS*, *EKCS*, etc. and decrypts the second ciphertext (using $\mathcal{F}_{\text{crypto}}$). If this succeeds, the MAC within the plaintext is verified (using $\mathcal{F}_{\text{crypto}}$). If successful, the test accepts and S continues the protocol.

Modeling this protocol as a multi-session real protocol $\mathcal{P}_{\text{TLS}} = !M_C \mid !M_S$ that uses $\mathcal{F}_{\text{crypto}}$ for all cryptographic operations (i.e., public-key and symmetric encryption, key derivation, digital signatures, and MAC) is straightforward. The protocol $\mathcal{P}_{\text{TLS}}$ is meant to realize $\mathcal{F}_{\text{key-use}}$, i.e., after the keys are established, the parties can send encryption and decryption requests to M_C and M_S which are MACed and encrypted under the corresponding keys. Corruption is defined such that the adversary can corrupt the public/private keys of parties (via $\mathcal{F}_{\text{crypto}}$) and can corrupt instances of M_C and M_S when they are created. In particular, the adversary can gain complete control over a party by corrupting her public/private keys and all her instances of M_C and M_S.

We provide a proof sketch that $\mathcal{P}_{\text{TLS}}$ satisfies implicit disjointness (see [22] for details). The proof does not need to exploit security of symmetric encryption. Moreover, the proof merely requires syntactic arguments (rather than probabilistic reasoning or reduction arguments) since we can use $\mathcal{F}_{\text{crypto}}$ for the cryptographic primitives.

The partnering function τ_{TLS} for $\mathcal{P}_{\text{TLS}}$ we use is the obvious one: Let ρ be a run of $\mathcal{E} \mid \mathcal{P}_{\text{TLS}} \mid \mathcal{F}_{\text{crypto}}$ for some environment $\mathcal{E}$ and α be the projection of ρ to an instance of M_r for some user $(p, lsid, r)$ (where $r \in \{C, S\}$). If $(p, lsid, r)$ is corrupted, then $\tau_{\text{TLS}}(\alpha) := \perp$. Otherwise, if $r = C$ and α contains at least the first two messages of the protocol, then $\tau_{\text{TLS}}(\alpha) := (N_C, N_S)$, where N_S is the server's nonce $(p, lsid, r)$ received and N_C is the nonce $(p, lsid, r)$ generated; analogously for the case $r = S$. It is easy to see that τ_{TLS} is valid for $\mathcal{P}_{\text{TLS}}$ because ideal nonces (i.e., nonces generated using $\mathcal{F}_{\text{crypto}}$) do not collide.

Theorem 5. *$\mathcal{P}_{\text{TLS}}$ satisfies implicit disjointness w.r.t. τ_{TLS}.*

Proof sketch. All symmetric keys (*PMS*, *MS*, *EKSC*, etc.) are, by definition, not explicitly shared: *PMS* is not a pre-shared key but a freshly generated symmetric key; *MS* is derived from *PMS* and all other keys are derived from *MS*. Hence, we only have to show (b) of Definition 2 for public-key encryption and digital signatures. More precisely, the only relevant cases are when the server performs a decryption request with k_S (to obtain *PMS*) or when it performs a verification request to verify the signature of the client.

We now consider the former case (decryption request with k_S); the latter follows a similar (even simpler) argumentation. In this proof sketch, we only consider the case where the server which makes the decryption request is uncorrupted and where the key k_C he received is uncorrupted (in $\mathcal{F}_{\text{crypto}}$) as well. (We refer to our technical report [22] for the case of corruption. The argument there requires a more precise description of our protocol and corruption model than what we can present in this extended abstract.)

So, let us assume that an uncorrupted instance of the server, say $\hat{M}_S$, performed an accepted and ideal decryption request. Let N_C be the nonce $\hat{M}_S$ received, let N_S be the nonce generated by $\hat{M}_S$, let k_S be its public key, let k_C be the public key received, and *ct* be the ciphertext received (containing *PMS*) and on which $\hat{M}_S$ performed the decryption request under consideration. Since the decryption request is accepted, by the definition of the test the server performs, we know that the *handshake* message has the required format and the signature verified. From this we can conclude that an uncorrupted instance made a signing request to $\mathcal{F}_{\text{crypto}}$ with (a pointer to) the private key of k_C and the message *handshake*; a corrupted instance would not have had access to an uncorrupted signing key. This instance must be in role C (so say the instance is $\hat{M}_C$), since uncorrupted server instances do not produce signatures. Since *handshake* contains N_C and N_S, we know that these are the nonces generated and received, respectively, by $\hat{M}_C$. Consequently, $\hat{M}_C$ and $\hat{M}_S$ are partners according to τ_{TLS}. Since the ciphertext *ct* and the public key k_S are contained in *handshake*, it follows that $\hat{M}_C$ must have encrypted *PMS* under k_S and obtained the ciphertext *ct* from $\mathcal{F}_{\text{crypto}}$. Hence, we have shown, as desired, that the partner $\hat{M}_C$ of $\hat{M}_S$ has issued the corresponding encryption request. □

Implicit Disjointness of SSH. The cryptographic core of the key exchange protocol of SSH—for which we show implicit disjointness—is depicted in Figure 3, with $K = g^{xy}$ and $sid = H(N_C, N_S, k_S, g^x, g^y, K)$, where H is a hash function. The symmetric encryption and MAC keys *EKCS*, *EKSC*, *IKCS*, *IKSC* are derived from K using H and *sid* as a seed. (The details are not relevant for proving implicit disjointness.) By $\{m\}_{k_1,k_2}$ we denote encrypt-and-MAC, i.e., $\{m\}_{k_1,k_2} = \{m\}_{k_2}, \mathrm{mac}_{k_1}(m)$. The formal model of SSH as a multi-session real protocol $\mathcal{P}_{\text{SSH}} = !M_C \mid !M_S$ is similar to the one for TLS. However, $\mathcal{P}_{\text{SSH}}$ only uses $\mathcal{F}_{\text{crypto}}$ for digital signatures; all other cryptographic operations (i.e., encryption, MAC, hashing) are carried out by M_C and M_S itself because $\mathcal{F}_{\text{crypto}}$ does not support Diffie-Hellman key exchange yet, and hence, K (and all derived keys) cannot be a key in $\mathcal{F}_{\text{crypto}}$. Still, in the proof that $\mathcal{P}_{\text{SSH}}$ satisfies implicit disjointness, we only need to do a reduction argument to the collision resistance of the hash function, since $\mathcal{P}_{\text{SSH}}$ uses $\mathcal{F}_{\text{crypto}}$

1. $C \to S$: c_1, N_C
2. $S \to C$: c_2, N_S
3. $C \to S$: g^x
4. $S \to C$: $k_S, g^y, \mathrm{sig}_{k_S}(sid)$
5. $C \to S$: $\{C, k_C, \mathrm{sig}_{k_C}(sid, C, k_C)\}_{IKCS,EKCS}$
6. $S \to C$: $\{\text{“success”}\}_{IKSC,EKSC}$

Figure 3: The SSH Key Exchange Protocol.

for digital signatures and security of the encryption scheme, the MAC scheme, or the Diffie-Hellman key exchange is not needed.

The partnering function τ_{SSH} for $\mathcal{P}_{\mathrm{SSH}}$ is the obvious one: It is defined similarly to TLS except that the SID is $sid = H(N_C, N_S, k_S, g^x, g^y, K)$ instead of (N_C, N_S). To show that it is valid, we need that the hash function is collision resistant; alternatively, one could define $sid = (N_C, N_S)$, in which case collision resistance is not needed to show that τ_{SSH} is valid, but then collision resistance would be necessary to show implicit disjointness. With τ_{SSH}, implicit disjointness of $\mathcal{P}_{\mathrm{SSH}}$ follows very easily since *sid* is part of every signature.

Theorem 6. *$\mathcal{P}_{\mathrm{SSH}}$ satisfies implicit disjointness w.r.t. τ_{SSH}.*

6. REFERENCES

[1] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext Recovery Attacks against SSH. In *S&P 2009*, pages 16–26. IEEE Computer Society, 2009.

[2] B. Barak, Y. Lindell, and T. Rabin. Protocol Initialization for the Framework of Universal Composability. Technical Report 2004/006, Cryptology ePrint Archive, 2004. `http://eprint.iacr.org`.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

[4] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.

[5] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *STOC'95*, pages 57–66. ACM, 1995.

[6] B. Blanchet. Computationally Sound Mechanized Proofs of Correspondence Assertions. In *CSF 2007*, pages 97–111. IEEE Computer Society, 2007.

[7] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[8] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical Report 2000/067, Cryptology ePrint Archive, December 2005. `http://eprint.iacr.org`.

[9] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[10] R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, 2002.

[11] R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.

[12] I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and Fixing Public-key Kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.

[13] K.-K. R. Choo and Y. Hitchcock. Security Requirements for Key Establishment Proof Models: Revisiting Bellare-Rogaway and Jeong-Katz-Lee Protocols. In C. Boyd and J. M. G. Nieto, editors, *ACISP 2005*, volume 3574 of *LNCS*, pages 429–442. Springer, 2005.

[14] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *CSFW 2006*, pages 321–334. IEEE Computer Society, 2006.

[15] J. P. Degabriele and K. G. Paterson. On the (In)Security of IPsec in MAC-then-Encrypt Configurations. In *CCS 2010*. ACM, 2010.

[16] S. Gajek, M. Manulis, O. Pereira, A. Sadeghi, and J. Schwenk. Universally Composable Security Analysis of TLS. In J. Baek, F. Bao, K. Chen, and X. Lai, editors, *ProvSec 2008*, volume 5324 of *LNCS*, pages 313–327. Springer, 2008.

[17] K. Kobara, S. Shin, and M. Strefler. Partnership in key exchange protocols. In W. Li, W. Susilo, U. K. Tupakula, R. Safavi-Naini, and V. Varadharajan, editors, *ASIACCS 2009*, pages 161–170. ACM, 2009.

[18] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425. Springer, 2003.

[19] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *CSFW 2006*, pages 309–320. IEEE Computer Society, 2006.

[20] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digitial Signature Functionalities with Local Computation. In *CSF 2008*, pages 270–284. IEEE Computer Society, 2008.

[21] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *CSF 2009*, pages 293–307. IEEE Computer Society, 2009.

[22] R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. Technical Report 2011/406, Cryptology ePrint Archive, 2011. `http://eprint.iacr.org`.

[23] R. Küsters and M. Tuengerthal. Ideal Key Derivation and Encryption in Simulation-based Security. In A. Kiayias, editor, *CT-RSA 2011*, volume 6558 of *LNCS*, pages 161–179. Springer, 2011.

[24] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[25] M. Ray and S. Dispensa. Renegotiating TLS. November 2009. Available at `http://extendedsubset.com/Renegotiating_TLS.pdf`.

[26] P. Rogaway and T. Stegers. Authentication without Elision: Partially Specified Protocols, Associated Data, and Cryptographic Models Described by Code. In *CSF 2009*, pages 26–39. IEEE Computer Society, 2009.

[27] E. Tews and M. Beck. Practical Attacks against WEP and WPA. In D. A. Basin, S. Capkun, and W. Lee, editors, *WISEC 2009*, pages 79–86. ACM, 2009.