Composition Theorems Without Pre-Established Session Identifiers

Ralf Küsters and Max Tuengerthal

University of Trier, Germany
{kuesters,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.

1 Introduction

Universal composition theorems, such as Canetti's composition theorem in the UC model [8] and Küsters' composition theorem in the IITM model [27], 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 [13] 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 [9] 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., [14, 17, 1, 35, 33] for recent examples), their comprehensive analysis still poses a big challenge, as often pointed out in the literature (see, e.g., [34, 26, 12]). 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 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 [30]. 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 [30], $\mathcal{F}_{\text{crypto}}$ 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 [19]. 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 can be found in the appendix.

2 Simulation-based Security

We briefly recall the framework of simulation-based security, following [27] (see also Appendix A). 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 $\mathcal{S} = M_1 \,|\, \cdots \,|\, M_k \,|\, !M'_1 \,|\, \cdots \,|\, !M'_{k'}$ 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) environments: *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 [27].

Definition 1 ([27]). *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 [27], 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 ([27]). *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 ([27]). *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 [8].

3 Universal Composition Without Pre-Established SIDs

Universal composition theorems, such as Theorem 2 and Canetti's composition theorem, 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 [9] 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.[1] 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

[1] In Canetti's second method, the initiator of a protocol gets the SID from the I/O interface in the first message. All other parties get the SID from the first network message. At first glance, it looks as if this might solve some of the problems described above. However, this is not the case. Every party still gets the SID in the first message (from I/O or network) and this is still some kind of prior agreement, namely what we call pre-establishment. The important point—due to the way multi-session ideal functionalities are defined—is that in the real protocol parties with the same (pre-established) SID have to use this SID in an essential way to realize the session of the ideal functionality with that SID; the original protocol typically does not use such SIDs (see also the end of Section 3.2 and the beginning of Section 4). Furthermore, Canetti's second method is impractical: For every SID a party must not run more than one instance. So, a responder (who receives the SID to be used from an initiator) would have to remember all SIDs used so far to ensure this.

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 $\mathcal{P} = !M_1 \mid \cdots \mid !M_n$ 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 $\mathbf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ 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 $(\mathit{lsid}, p, \mathsf{Corrupted?})$ to M_r (on the I/O tape), M_r replies with $(\mathit{lsid}, p, \mathsf{Corrupted}, \mathbf{corrupted})$. 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 $\mathcal{F}$ (see Figure 4 in the appendix for pseudo-code):

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 $(\mathit{lsid}, p, \mathsf{Start}, m)$ 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 $(\mathsf{Create}, \mathit{sid})$ to $\mathcal{F}$ where $\mathit{sid} = (p_1, \mathit{lsid}_1, 1), \ldots, (p_n, \mathit{lsid}_n, n)$ 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 $(p_r, \mathit{lsid}_r, r)$. 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 $\mathsf{Corrupted?}$ (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 $(\mathsf{Corrupt}, (p, \mathit{lsid}, r))$ 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 $(\mathit{lsid}, p, \mathsf{Corrupted?})$. 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. Informally, 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 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 $\mathcal{E} \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$, 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 $\mathtt{true}$, 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 (see Appendix B for the proof).

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

Fig. 1. The original Needham-Schroeder Public-Key (NSPK) protocol is insecure [31]. Its modified version, resulting from the joint-state construction, which prefixes every plaintext with a pre-established SID *sid* is secure (see Appendix D.2 for details).

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 Canetti's composition theorem, 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 [13] 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 [13, 28] and public-key encryption [28] as well as encryption with long-term symmetric keys [29]. 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 *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 [30]. 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 [30], $\mathcal{F}_{\text{crypto}}$ 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 first 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 [30], which, as mentioned, supports several cryptographic operations. More details are given in Appendix C.1. 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 $(\mathsf{Enc}, \mathit{ptr}, x)$ and $(\mathsf{Dec}, \mathit{ptr}, y)$: 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).[2] Furthermore, all keys contained in x' are marked known as they are encrypted under a known key.

Realization of $\mathcal{F}_{\text{crypto}}$. In [30], 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 [30].

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 ensures 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[3] 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 $\bot$. 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 $\tau_{(p,\mathit{lsid},r)}(\rho) := \tau(\alpha)$ 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 $\tau_{(p,\mathit{lsid},r)}(\rho) = \tau_{(p',\mathit{lsid}',r')}(\rho) \neq \bot$.

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 $\tau_{(p,\mathit{lsid},r)}(\rho'') \neq \bot$, then it holds $\tau_{(p,\mathit{lsid},r)}(\rho') = \tau_{(p,\mathit{lsid},r)}(\rho'')$ 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 $\tau_{(p,\mathit{lsid},r)}(\rho) = \bot$. 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 $(p', \mathit{lsid}', r') = (p, \mathit{lsid}, r)$.

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.

[2] This constitutes a slight modification to the original ideal functionality in [30], 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.

[3] 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, 11, 15, 25]. 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).

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 (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 [7].

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 $\mathcal{E} \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathit{Sim}_{\tau} \,|\, F_{\text{single}} \,|\, \mathcal{F}$, 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., $\tau(\alpha_1) = \cdots = \tau(\alpha_n) \neq \bot$, 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 (see Appendix C.2 for the full proof), 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 [16], 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 Appendix D 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 Appendix D.8). 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 [30] for IEEE 802.11i and from [19] for SSL/TLS. For example, in [19] 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.

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

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): p_C and p_S are the PIDs of C and S, respectively; 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

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

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

of all previous messages, that is, $\mathit{handshake} = c_1\|N_C\|p_S\|k_S\|c_2\|N_S\|p_C\|k_C\|\{\!|PMS|\!\}_{k_S}$. 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 $\mathrm{sig}_{k_C}(\mathit{handshake})$ (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, digital signatures, key derivation, 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 Appendix D.3 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 (i.e., the 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. (See Appendix D.3 for the case of corruption. The argument there requires a more precise description of our protocol and corruption model.)

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

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. $C \to S$: $\{p_C, k_C, \mathrm{sig}_{k_C}(sid, p_C, k_C)\}_{IKCS,EKCS}$
6. $S \to C$: $\{\text{“success”}\}_{IKSC,EKSC}$

Fig. 3. The SSH Key Exchange Protocol.

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 $\mathcal{F}_{\text{crypto}}$ 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 $\mathcal{P}_{\text{SSH}}$ 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 $\mathcal{P}_{\text{SSH}}$ follows very easily since *sid* is part of every signature.

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

References

1. M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext Recovery Attacks against SSH. In *IEEE Symposium on Security and Privacy (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/2004/006/`.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 2000.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In D. Stinson, editor, *Advances in Cryptology – Crypto '93, 13th Annual International Cryptology Conference*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.
5. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM, 1995.

6. F. Bersani and H. Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. RFC 4764 (Experimental), Jan. 2007.
7. B. Blanchet. Computationally Sound Mechanized Proofs of Correspondence Assertions. In *20th IEEE Computer Security Foundations Symposium (CSF 2007)*, pages 97–111. IEEE Computer Society, 2007.
8. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, 2001.
9. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical Report 2000/067, Cryptology ePrint Archive, December 2005. `http://eprint.iacr.org/2000/067/`.
10. R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.
11. R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2001.
12. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference*, volume 2442 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2002.
13. R. Canetti and T. Rabin. Universal Composition with Joint State. In *Advances in Cryptology, 23rd Annual International Cryptology Conference (CRYPTO 2003), Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.
14. 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.
15. 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, *Information Security and Privacy, 10th Australasian Conference (ACISP 2005)*, volume 3574 of *Lecture Notes in Computer Science*, pages 429–442. Springer, 2005.
16. A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 321–334. IEEE Computer Society, 2006.
17. J. P. Degabriele and K. G. Paterson. On the (In)Security of IPsec in MAC-then-Encrypt Configurations. In *17th ACM Conference on Computer and Communications Security (CCS 2010)*. ACM, 2010.
18. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176.
19. 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, *Provable Security, Second International Conference (ProvSec 2008)*, volume 5324 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2008.
20. C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2005)*. The Internet Society, 2005.
21. D. Hofheinz, D. Unruh, and J. Müller-Quade. Polynomial Runtime and Composability. Technical Report 2009/023, Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/2009/023/`.
22. IEEE Standard 802.11-2007. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Part 11 of IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements, June 2007.
23. IEEE Standard 802.11i-2004. Medium Access Control (MAC) Security Enhancements, Amendment 6 to IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, July 2004.
24. C. Kaufman, P. Hoffman, Y. Nir, and P. Eronen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996 (Proposed Standard), Sept. 2010. Updated by RFC 5998.
25. 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, *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security (ASIACCS 2009)*, pages 161–170. ACM, 2009.
26. H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003.

27. R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.
28. R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digitial Signature Functionalities with Local Computation. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 270–284. IEEE Computer Society, 2008.
29. R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 293–307. IEEE Computer Society, 2009.
30. R. Küsters and M. Tuengerthal. Ideal Key Derivation and Encryption in Simulation-based Security. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011, The Cryptographers' Track at the RSA Conference 2011, Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 161–179. Springer, 2011.
31. G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
32. R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.
33. M. Ray and S. Dispensa. Renegotiating TLS. Available at `http://extendedsubset.com/Renegotiating_TLS.pdf`, November 2009.
34. P. Rogaway and T. Stegers. Authentication without Elision: Partially Specified Protocols, Associated Data, and Cryptographic Models Described by Code. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 26–39. IEEE Computer Society, 2009.
35. E. Tews and M. Beck. Practical Attacks against WEP and WPA. In D. A. Basin, S. Capkun, and W. Lee, editors, *Proceedings of the Second ACM Conference on Wireless Network Security (WISEC 2009)*, pages 79–86. ACM, 2009.
36. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), Jan. 2006.
37. T. Ylonen and C. Lonvick. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253 (Proposed Standard), Jan. 2006.

A The IITM Model

While our concepts and results are quite model-independent, in our proofs we of course need to consider a concrete model for simulation-based security. As mentioned in Section 2, we use the model proposed in [27], called the IITM model. For the readers convenience, we provide some more background on this model, although the details are not essential to be able to follow the paper.

While being in the spirit of Canetti's UC model [9], the IITM model resolves some technical problems of the UC model caused by the way the runtime of interactive Turing machines is defined (see, e.g., discussions in [27, 28, 21]). As pointed out in [28], these problems also affect the general joint state composition theorem in the UC model. (We note that, in the IITM model, this theorem is simply a corollary of the universal composition theorem.)

In the IITM model, security notions and composition theorems are formalized based on a relatively simple, but expressive general computational model in which so-called inexhaustible interactive Turing machines (IITMs) and systems of IITMs are defined. We now provide some more information on this general computational model. The security notions and composition theorems based on this model have already been presented in Section 2. However, we present the definition of a session version of a system, used in Theorem 2

The General Computational Model. The general computational model is defined in terms of systems of IITMs. An *inexhaustible interactive Turing machine (IITM) M* is a probabilistic polynomial-time Turing machine with named input and output tapes. The names determine how different IITMs are connected in a system of IITMs. An IITM runs in one of two modes, CheckAddress and Compute. The CheckAddress mode is used as a generic mechanism for addressing copies of IITMs in a system of IITMs, as explained below. The runtime of an IITM may depend on the length of the input received so far and in *every* activation an IITM may perform a polynomial-time computation; this is why these ITMs are called inexhaustible.

A *system* $\mathcal{S}$ of IITMs 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 IITMs such that the names of input tapes of different IITMs 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 IITM is active and all other IITMs wait for new input; the first IITM to be activated in a run of $\mathcal{S}$ is the so-called master IITM, of which a system has at most one. To illustrate runs of systems, consider, for example, the system $\mathcal{S} = M_1 \,|\, !M_2$ and assume that M_1 has an output tape named c, M_2 has an input tape named c, and M_1 is the master IITM. (There may be other tapes connecting M_1 and M_2.) Assume that in the run of $\mathcal{S}$ executed so far, one copy of M_2, say M'_2, has been generated and that M_1 just sent a message m on tape c. This message is delivered to M'_2 (as the first, and, in this case, only copy of M_2). First, M'_2 runs in the $\mathsf{CheckAddress}$ mode with input m; this is a deterministic computation which outputs "accept" or "reject". If M'_2 accepts m, then M'_2 gets to process m (in the $\mathsf{Compute}$ mode) and could, for example, send a message back to M_1. Otherwise, a new copy M''_2 of M_2 with fresh randomness is generated and M''_2 runs in $\mathsf{CheckAddress}$ mode with input m. If M''_2 accepts m, then M''_2 gets to process m. Otherwise, M''_2 is removed again, the message m is dropped, and the master IITM (in this case M_1) is activated. The master IITM is also activated if the currently active IITM does not produce output at the end of its activation (and hence, does not trigger another machine). A run stops if the master IITM does not produce output or an IITM outputs a message on a tape named $\mathsf{decision}$. Such a message is considered to be the overall output of the system.

Session Versions. We now defined the session version of a system, as used in Theorem 2. A *session version* $\underline{\mathcal{S}} = \underline{M_1} \,|\, \cdots \,|\, \underline{M_k} \,|\, \underline{!M'_1} \,|\, \cdots \,|\, \underline{!M'_{k'}}$ of a system $\mathcal{S} = M_1 \,|\, \cdots \,|\, M_k \,|\, !M'_1 \,|\, \cdots \,|\, !M'_{k'}$ is obtained from $\mathcal{S}$ by replacing every machine M in $\mathcal{S}$ by its session version $\underline{M}$, where $\underline{M}$ simulates M and acts as a "wrapper" around M: In its $\mathsf{CheckAddress}$ mode $\underline{M}$ only accepts messages prefixed by some specific SID. The SID used is the one with which $\underline{M}$ was first activated. Moreover, $\underline{M}$ prefixes all messages output by M with that SID. So with $\underline{M}$ a specific instance of M can be addresses via its SID.

B Universal Composition Without Pre-Established SIDs

In this section, we provide a formal definition of multi-session local-SID (ideal) functionalities in pseudo-code and prove Theorem 3.

As in Section 3.2, 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. For example, $F = F_{\text{key-use}}$ or $F = F_{\text{sc}}$ as defined in Appendix D.1. The multi-session local-SID (ideal) functionality $\mathcal{F}[F]$ is given in pseudocode in Figure 4. We note that the ideal functionality F is not aware of the local SIDs of its users. Often, this is not needed (e.g., $F_{\text{key-use}}$ and F_{sc} do not need this). If it is needed that F is aware of the local SIDs, then F could be defined such that expects the local SIDs to be given in the optional bit string in the *session-start* message. Of course, a protocol using such an F (more precisely: using $\mathcal{F}[F]$) has to truthfully include the local SID in this request, but this is just a modeling aspect.

Before proving Theorem 3, we prove the following lemma, which is used in the proof of Theorem 3: Every multi-session protocol that single-session-realizes a multi-session local-SID functionality, it holds that if a user produces I/O output, then this user is corrupted or belongs to a (global) session where all users are uncorrupted. Intuitively, this holds because such I/O output is impossible in the ideal world (i.e., with the ideal functionality).

Lemma 1. *Let $\mathcal{F}$ and $\mathcal{F}'$ be multi-session local-SID functionalities and let $\mathcal{P} = !M_1 \,|\, \ldots \,|\, !M_n$ 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 for every environment $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{F}'$ the following holds with overwhelming probability, where the probability is over runs ρ of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$: If an instance of M_r, for some r, produces I/O output to $\mathcal{E}$ at some point in ρ, then either*

i) this instance of M_r is corrupted (i.e., its flag **corrupted** *is* true*) at that point in ρ or*

$\mathcal{F}[F]$

Tapes: input: t_r^{in} for $r = 1, \ldots, n$ (I/O tapes), $t_{\text{adv}}^{\text{in}}$ (network tape)
output: t_r^{out} for $r = 1, \ldots, n$ (I/O tapes), $t_{\text{adv}}^{\text{out}}$ (network tape)

We say that "m is received from user $(p, lsid, r)$" if the message $(lsid, p, m)$ is received on tape t_r^{in}. By "send m to $(p, lsid, r)$" we denote that $(lsid, p, m)$ is output on tape t_r^{out}. Similarly, we say that "m is received from/sent to t_{adv}" if m is received on $t_{\text{adv}}^{\text{in}}$ or m is output on $t_{\text{adv}}^{\text{out}}$, respectively.

State: The state of $\mathcal{F}[F]$ is maintained by two mappings $\mathbf{LS}\colon \{0,1\}^* \to \{0,1\}^* \cup \{\bot\}$ and $\mathbf{S}\colon \{0,1\}^* \to \{0,1\}^* \cup \{\bot\}$. In the initial state, $\mathbf{LS}$ and $\mathbf{S}$ map every bit string to $\bot$. ("LS" stands for "local sessions" and "S" stands for "(global) sessions" because $\mathbf{LS}(\mathit{user})$ stores the state of the user $\mathit{user} = (\mathit{lsid}, p, m)$ and $\mathbf{S}(\mathit{sid})$ stores the state of the session identified by sid.)

CheckAddress: Every message on every tape is always accepted.

Compute:

- *I/O input from users:* Upon receiving a message m from user $\mathit{user} = (p, \mathit{lsid}, r)$ (for some m, p, lsid, r) do:
  1. If $m = (\mathsf{Start}, m')$ for some bit string m' and $\mathbf{LS}(\mathit{user}) = \bot$, then set $\mathbf{LS}(\mathit{user}) := (0, m')$ and send (user, m) to t_{adv}.
  2. If $m = \mathsf{Corrupted?}$, then send $(\mathsf{Corrupted}, \mathsf{true})$ to user if $\mathbf{LS}(\mathit{user}) = \mathsf{corrupted}$, otherwise, send $(\mathsf{Corrupted}, \mathsf{false})$ to user.
  3. If $m \neq \mathsf{Corrupted?}$ and $\mathbf{LS}(\mathit{user}) = (1, \mathit{sid})$ for some $\mathit{sid} = \mathit{user}_1, \ldots, \mathit{user}_n$ (note that $\mathit{user}_r = \mathit{user}$), then simulate the instance $\mathbf{S}(\mathit{sid})$ of F with I/O input m for role r (and update the configuration stored in $\mathbf{S}(\mathit{sid})$). Let m' be the output. If $m' = \varepsilon$ is the empty output, then produce empty output as well. Otherwise, if m' is I/O output for role r', then send m' to $\mathit{user}_{r'}$. Otherwise (i.e., $m' \neq \varepsilon$ is network output), send (sid, m') to t_{adv}.
  4. If $m \neq \mathsf{Corrupted?}$ and $\mathbf{LS}(\mathit{user}) = \mathsf{corrupted}$, then send (user, m) to t_{adv}.
- *Network input:* Upon receiving a message m from t_{adv} do:
  5. If $m = (\mathsf{Create}, \mathit{sid})$ where $\mathit{sid} = \mathit{user}_1, \ldots, \mathit{user}_n$ such that $\mathbf{LS}(\mathit{user}_r) = (0, m_r)$ for some m_r, for all $r \leq n$, then set $\mathbf{LS}(\mathit{user}_r) := (1, \mathit{sid})$ for all $r \leq n$, set $\mathbf{S}(\mathit{sid})$ to be the initial configuration of a new instance of F, and simulate this new instance of F with I/O input $(\mathsf{Create}, m_1, \ldots, m_n)$ for role 1 as in rule 3.
  6. If $m = (\mathsf{Corrupt}, \mathit{user})$ for some user such that $\mathbf{LS}(\mathit{user}) = (0, m)$ for some m, then set $\mathbf{LS}(\mathit{user}) := \mathsf{corrupted}$ and send $\mathsf{Corrupted}$ to t_{adv}.
  7. If $m = (\mathsf{Output}, \mathit{user}, m')$ for some user and m' such that $\mathbf{LS}(\mathit{user}) = \mathsf{corrupted}$, then send m' to user.
  8. If $m = (\mathit{sid}, m')$ for some $\mathit{sid} = \mathit{user}_1, \ldots, \mathit{user}_n$ such that $\mathbf{S}(\mathit{sid}) \neq \bot$, then simulate the instance $\mathbf{S}(\mathit{sid})$ of F with network input m' as in rule 3.
- Upon I/O or network input not matching a rule above, the input is ignored (i.e., produce empty output).

Fig. 4. The multi-session local-SID (ideal) functionality $\mathcal{F}[F]$ of an ideal functionality F (with n roles).

ii) this instance of M_r belongs to a (global) session in $\mathcal{F}'$ at that point in ρ and all users (i.e., the corresponding instances of M_r) of this session are not *corrupted at that point in ρ.*

We note that in the assumptions of the above lemma it is not required that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq^* F_{\text{single}} \,|\, \mathcal{F}$ but only that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq F_{\text{single}} \,|\, \mathcal{F}$. As mentioned in Section 3.3, this lemma in particularly shows that the simulator for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq^* F_{\text{single}} \,|\, \mathcal{F}$ is not restricted because it terminates in the first stage when the emulated $\mathcal{P}$ outputs a message at its I/O interface.

Proof. Assume that there exists an environment $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{F}'$ such that with non-negligible probability (in the security parameter), say ε, in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$, the statement of the lemma does *not* hold. That is, some instance M_r produces I/O output to $\mathcal{E}$ at some point in ρ such that

i) M_r is not corrupted at that point in ρ and
ii) M_r does not belong to a session in $\mathcal{F}'$ at that point in ρ or there exists a user that is in the same session in $\mathcal{F}'$ as M_r and this user is corrupted at that point in ρ.

Let $p_{\mathcal{E}}$ be the polynomial (in the security parameter) that bounds the runtime of $\mathcal{E}$ (such a polynomial exists because $\mathcal{E}$ is polynomially bounded). In particular, $p_{\mathcal{E}}$ also bounds the number of instances of M_r in every run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$. Furthermore, let Sim be a simulator for $F_{\text{single}} \,|\, \mathcal{F}$ such that

$$\mathcal{E} \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F} \tag{1}$$

for every environment $\mathcal{E}$ for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$.

We now construct an environment $\mathcal{E}'$ for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ (and, hence, for $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$), which distinguishes between $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$ with non-negligible probability (i.e., this contradicts (1)), as we show below. The system $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ or $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$, respectively, is called the *external system* and the users in the external system are called *external users* in the following description of $\mathcal{E}'$.

At first $\mathcal{E}'$ chooses a bit $b \in \{0, 1\}$ uniformly at random. Intuitively, $\mathcal{E}'$ tries to guess which of the cases—M_r does not belong to a session (case $b = 0$) or some user in M_r's session is corrupted (case $b = 1$)—occurs. First, we describe $\mathcal{E}'$ in the case where $b = 0$: Then, $\mathcal{E}'$ chooses a number $i \leq p_{\mathcal{E}}$ uniformly at random. ($\mathcal{E}'$ tries to guess the instance of M_r that produces I/O output before it is corrupted or belongs to a session.) Then, $\mathcal{E}'$ exactly emulates $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ except that all (I/O and network) messages from $\mathcal{E}$ to the i-th user (i.e., the user which receives the i-th *session-start* message from $\mathcal{E}$) are sent to the external system. Vice versa, network messages from the external system are forwarded to $\mathcal{E}$. Note that only one user exists in the external system. If $\mathcal{E}'$ receives an I/O message from the external system, i.e., from the external user, then $\mathcal{E}'$ checks whether the external user is corrupted by sending a Corrupted? request to the external system. If the external user is corrupted, then $\mathcal{E}'$ outputs 0 on the decision tape (which stops the run with overall output 0). Otherwise, $\mathcal{E}'$ outputs 1 on the decision tape. If $\mathcal{E}'$ never received I/O output from the external system and the emulation of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ stopped, then $\mathcal{E}'$ outputs 0 on the decision tape.

Second, if $b = 1$, then $\mathcal{E}'$ chooses $i_1, \ldots, i_n \leq p_{\mathcal{E}}$ uniformly at random. ($\mathcal{E}'$ tries to guess the session M_r belongs to where some user is corrupted.) Then, $\mathcal{E}'$ exactly emulates $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ except that all (I/O and network) messages from $\mathcal{E}$ to the i_r-th user for some $r \leq n$ (i.e., the user which receives the i_r-th *session-start* message from $\mathcal{E}$) are sent to the external system. Vice versa, network messages from the external system are forwarded to $\mathcal{E}$. Furthermore, if $\mathcal{E}$ sends a valid *session-create* message to $\mathcal{F}'$ where the user of role r is the i_r-th user, for all $r \leq n$, then $\mathcal{E}'$ sends this *session-create* message to the external system. If $\mathcal{E}'$ receives an I/O message m from the external system, i.e., from some external user, then $\mathcal{E}'$ checks whether the external user is corrupted by sending a Corrupted? request for this user to the external system. If this user is corrupted, then $\mathcal{E}'$ forwards m to $\mathcal{E}$. Otherwise, $\mathcal{E}'$ checks whether some external user is corrupted (by sending a Corrupted? request for all other external users to the external system). If this is the case, then $\mathcal{E}'$ outputs 1 on the decision tape. Otherwise, $\mathcal{E}'$ outputs 0 on the decision tape. If $\mathcal{E}'$ never received I/O output from the external system from an uncorrupted external user and the emulation of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ stopped, then $\mathcal{E}'$ outputs 0 on the decision tape.

Next, we show that $\mathcal{E}'$ distinguishes between $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$ with non-negligible probability which is a contradiction to (1).

It is easy to see that $\mathcal{E}'$ when running together with $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}$ never outputs 1 on the decision tape: In the case where $b = 0$, by definition of multi-session local-SID functionalities, $F_{\text{single}} \,|\, \mathcal{F}$ only produces I/O output after a local session (i.e., user) in $\mathcal{F}$ is corrupted or a session has been created. But $\mathcal{F}$ only contains one local session (which corresponds to the external user) and $\mathcal{E}'$ verifies that this local session is uncorrupted before outputting 1. Furthermore, $\mathcal{F}$ never creates a session (i.e., accepts a *session-create* message from Sim) because $n \geq 2$ and $\mathcal{F}$ only contains one local session. In the case where $b = 1$, $\mathcal{E}'$ would only output 1 on the decision tape if there exists an external user that is corrupted and there exists an external user that is uncorrupted and produces I/O output to $\mathcal{E}$. But this is impossible by definition of multi-session local-SID functionalities because $\mathcal{F}$ contains at most one local session per role. Furthermore, these local sessions either belong to a (global) session in which case none is corrupted or a local session is corrupted in which case every local session that produces I/O output to $\mathcal{E}$ must be corrupted.

On the other hand, in a run of $\mathcal{E}' \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$, $\mathcal{E}'$ outputs 1 with probability at least $\frac{\varepsilon}{2p_{\mathcal{E}}^{n+1}}$ because $\mathcal{E}'$ outputs 1 if it guessed b and the external user(s) such that the statement of the lemma does hold for this user(s). Since ε is non-negligible and $p_{\mathcal{E}}$ is a polynomial, $\frac{\varepsilon}{2p_{\mathcal{E}}^{n+1}}$ also is non-negligible. □

Proof of Theorem 3. To prove Theorem 3, we first define a simulator Sim for $\mathcal{F}$ and then show that $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}' \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}$ for every environment $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{F}'$.

In the following definition of Sim, let Sim' be the simulator for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq^* F_{\text{single}} \,|\, \mathcal{F}$, i.e., it holds that $\mathcal{E} \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \equiv \mathcal{E} \,|\, \mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F}$ for every environment $\mathcal{E}$ for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ and, as described in Section 3.3, Sim' exactly simulates $\mathcal{P} \,|\, \mathcal{F}'$ until the session is created (in $\mathcal{F}'$) or an instance gets corrupted. Throughout this proof, we use the following abbreviations:

$$\begin{aligned} \mathcal{R} &:= F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \\ \mathcal{I} &:= \mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F} \ . \end{aligned}$$

So, we have:

$$\mathcal{E}' \,|\, \mathcal{R} \equiv \mathcal{E}' \,|\, \mathcal{I} \tag{2}$$

for every environment $\mathcal{E}'$ for $\mathcal{R}$.

There is a strong correspondence between (global) sessions in $\mathcal{F}'$ and (global) session in $\mathcal{F}$. But there are sessions in $\mathcal{F}'$ which do not correspond to a session in $\mathcal{F}$. Namely, these are sessions in $\mathcal{F}'$ where some of the users (i.e., the instances of M_r in $\mathcal{P}$) that belong to this session are corrupted (i.e., the flag **corrupted** of M_r is true). In $\mathcal{F}$ this cannot be a session because corrupted users must not belong to sessions. This does not break simulation because the simulator Sim can corrupt these users in $\mathcal{F}$. Furthermore, by Lemma 1, uncorrupted users in such sessions do not produce I/O output to the environment until they are corrupted. To be able to talk about these sessions easier in the following, we call the *session-create* messages that create such sessions *session-create* messages with corrupted users. More formally, we say that a *session-create* messages m is a *session-create message with corrupted users* (at some point in a run of $\mathcal{P} \,|\, \mathcal{F}'$ for some environment) if at that point in the run m is valid for $\mathcal{F}'$ (i.e., $\mathcal{F}'$ would accept m and the session would be created) and there exists a user (p, lsid, r) in m such that the instance of M_r corresponding to (p, lsid, r) is corrupted (i.e., the flag **corrupted** of this instance of M_r is true). All other *session-create* messages are called *session-create message without corrupted users.*

Definition 3 (The Simulator Sim). *The IITM Sim has the same network interface as $\mathcal{P} \,|\, \mathcal{F}'$ and connects to the network interface of $\mathcal{F}$ (i.e., $\mathit{Sim} \,|\, \mathcal{F}$ has the same external interface as $\mathcal{P} \,|\, \mathcal{F}'$). Upon a session-start message for a user (which is forwarded by $\mathcal{F}$ to Sim), Sim starts the simulation of an instance of M_r (in $\mathcal{P}$) and $\mathcal{F}'$ for this user. It forwards all network output/input of this instance to/from the environment and vice versa. If M_r produces I/O output, Sim terminates (in this case the simulation fails but we will show that this occurs only with negligible probability). If this instance gets corrupted, Sim corrupts the corresponding local session in $\mathcal{F}$ and continues the simulation of M_r; Sim now also forwards all I/O output/input of this instance to/from the user which is now possible because the corresponding local session in $\mathcal{F}$ is corrupted. If the simulated $\mathcal{F}'$ receives a valid session-create message m, then Sim does the following: If m is a session-create message with corrupted users (as defined above), then Sim simply continues the simulation for the users of this session, i.e., Sim forwards m to the simulated $\mathcal{F}'$. Otherwise, Sim continues the simulation for the users of this session by a simulated instance of Sim'. Because Sim' in its first stage exactly simulated $\mathcal{P} \,|\, \mathcal{F}'$, Sim can adjust the state of Sim' appropriately.*

For the rest of the proof, we fix an environment $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{F}'$. Let $p_{\mathcal{E}}$ be a polynomial (in the security parameter) that bounds the overall runtime (i.e., taken steps) of $\mathcal{E}$. (By definition of environmental systems in the IITM model such a polynomial exists.) Since only $\mathcal{E}$ can create new instances of machines in $\mathcal{P}$ by sending requests to them, the overall number of these instances is bounded by $p_{\mathcal{E}}$.

Next, we define hybrid systems H_i for all $i \in \mathbf{N}$. Basically, H_i emulates $\mathcal{E}$ interacting with $\mathcal{P} \,|\, \mathcal{F}'$ and $\mathit{Sim} \,|\, \mathcal{F}$ such that the first $i-1$ sessions are handled by $\mathcal{P} \,|\, \mathcal{F}'$ and all later session are handled by $\mathit{Sim} \,|\, \mathcal{F}$. But all sessions where some users are corrupted are handled by $\mathcal{P} \,|\, \mathcal{F}'$, i.e., we are only counting sessions created by *session-create* messages without corrupted users.

Definition 4 (The Hybrid System H_i). *The IITM H_i for every $i \in \mathbf{N}$ has only two external tapes:* start *and* decision*. It emulates $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ as follows.*

For every user $(p, lsid, r)$, as long as $\mathcal{E}$ has not sent a valid session-create message without corrupted users (as defined above) which contains the user $(p, lsid, r)$, H_i emulates an instance of M_r for this user (just as in $\mathcal{P} \,|\, \mathcal{F}'$). If such an instance of M_r produces I/O output to $\mathcal{E}$ before it is corrupted, then H_i terminates; we say that H_i terminates with an error*. (We will show that this only happens with negligible probability.)*

Next, we define what happens if $\mathcal{E}$ sends the j-th session-create message m without corrupted users to $\mathcal{F}'$, for some $j \in \mathbf{N}$. (That is, $j = 1$ for the first such session-create message output by $\mathcal{E}$, $j = 2$ for the second, etc.) Then, H_i does the following:

i) If $j < i$, then H_i simply continues the emulation of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$, i.e., H_i forwards m to the emulated $\mathcal{F}'$. (Note that, from now on, the users of this session may produces I/O output to $\mathcal{E}$ without being corrupted.)

ii) Otherwise (i.e., $j \geq i$), H_i creates a new copy of $\mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F}$ for this (global) session where the state of Sim', F_{single}, and $\mathcal{F}$ is set as if this system was used from the beginning for simulating the users of this session. Because Sim' in its first stage exactly simulated $\mathcal{P} \,|\, \mathcal{F}'$, H_i can adjust the state of Sim' appropriately. (This is exactly what Sim does, see Definition 3.) From then on, H_i forwards all messages from $\mathcal{E}$ to this session and users of this session to this copy of $\mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F}$. Vice versa, H_i forwards all messages from this copy of $\mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F}$ to $\mathcal{E}$.

Finally, if $\mathcal{E}$ produces output m on the decision tape (which then terminates the run with overall output m), then H_i outputs m on the decision tape.

By construction of H_1 and definition of *Sim*, it is easy to see that

$$H_1 \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F} \tag{3}$$

because every session in H_1 is treated as in $\mathit{Sim} \,|\, \mathcal{F}$ (it always holds that $j \geq 1$). Moreover, by construction of $H_{p_\mathcal{E}+1}$ and by Lemma 1 (note that obviously $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq^* F_{\text{single}} \,|\, \mathcal{F}$ implies $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}' \leq F_{\text{single}} \,|\, \mathcal{F}$), one can show that

$$H_{p_\mathcal{E}+1} \equiv \mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}' \tag{4}$$

because all sessions in $H_{p_\mathcal{E}+1}$ are handled as in $\mathcal{P} \,|\, \mathcal{F}'$ (it always holds that $j < p_\mathcal{E} + 1$ because $\mathcal{E}$ is bounded by $p_\mathcal{E}$). Note that, by Lemma 1, $H_{p_\mathcal{E}+1}$ terminates with an error only with negligible probability.

Given a run of H_i (for some $i \in \mathbf{N}$), we say that the *j-th user* is the user corresponding to the j-th instance of M_r which is created by $\mathcal{E}$ (by sending a *session-start* message for this user).

To show that H_i is indistinguishable from H_{i+1}, for all $i \in \mathbf{N}$, we introduce hybrid systems $\hat{H}_{i,i_1,\dots,i_n}$, for all $i, i_1, \dots, i_n \in \mathbf{N}$, which are supposed to run with either $\mathcal{R}$ or $\mathcal{I}$. Similar to H_i, $\hat{H}_{i,i_1,\dots,i_n}$ emulates $\mathcal{E}$ interacting with $\mathcal{P} \,|\, \mathcal{F}'$ and $\mathit{Sim} \,|\, \mathcal{F}$ but the i-th session is handled by $\mathcal{R}$ or $\mathcal{I}$, respectively. To successfully outsource the i-th session, $\hat{H}_{i,i_1,\dots,i_n}$ has to know which users belong to this session even before this session is created. Therefore, $\hat{H}_{i,i_1,\dots,i_n}$ is additionally parametrized by $i_1, \dots, i_n$ which tells $\hat{H}_{i,i_1,\dots,i_n}$ that the i_r-th user plays role r in the i-th session.

Definition 5 (The Hybrid System $\hat{H}_{i,i_1,\dots,i_n}$). *The IITM $\hat{H}_{i,i_1,\dots,i_n}$ for every $i, i_1, \dots, i_n \in \mathbf{N}$ is an environment for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ (and, hence, for $\mathit{Sim}' \,|\, F_{\text{single}} \,|\, \mathcal{F}$).*

Similar to H_i, $\hat{H}_{i,i_1,\dots,i_n}$ simulates $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$ as follows.

For every user $(p, lsid, r)$ who is not the i_r-th user, as long as $\mathcal{E}$ has not sent a valid session-create message without corrupted users (as defined above) which contains the user $(p, lsid, r)$, $H_{i,i_1,\dots,i_n}$ emulates an instance of M_r for this user (just as in $\mathcal{P} \,|\, \mathcal{F}'$). If such an instance of M_r produces I/O output to $\mathcal{E}$ before it is corrupted, then $H_{i,i_1,\dots,i_n}$ outputs a special error message to the decision tape (i.e., the run stops and the overall output is this error message); we say that $H_{i,i_1,\dots,i_n}$ terminates with an error*. (We will show that this only happens with negligible probability.)*

Messages from $\mathcal{E}$ to the i_r-th user, for every $r \leq n$, are forwarded by $\hat{H}_{i,i_1,\dots,i_n}$ to the external session (i.e., to $\mathcal{R}$ or $\mathcal{I}$, respectively). Vice versa, $\hat{H}_{i,i_1,\dots,i_n}$ forwards messages from the external session to $\mathcal{E}$.

Next, we define what happens if $\mathcal{E}$ sends the j-th session-create message m without corrupted users to $\mathcal{F}'$, for some $j \in \mathbf{N}$. Then, $\hat{H}_{i,i_1,\dots,i_n}$ does the following:

i) If $j = i$, then $H_{i,i_1,\ldots,i_n}$ performs the following check: For all $r \leq n$, $H_{i,i_1,\ldots,i_n}$ verifies that the user in role r in m is the i_r-th user and that this user is uncorrupted in the external session ($H_{i,i_1,\ldots,i_n}$ can do this by sending a request of the form Corrupted? *for this user to the external session). If this check fails, then $H_{i,i_1,\ldots,i_n}$ terminates with an error (because the external session does not consist of the users $i_1, \ldots, i_n$ or some of these users are corrupted when the session is created). Otherwise (i.e., the check succeeds), $\hat{H}_{i,i_1,\ldots,i_n}$ forwards m to the external session. From then on, $\hat{H}_{i,i_1,\ldots,i_n}$ forwards all messages from $\mathcal{E}$ to the this session to the external session. Vice versa, $\hat{H}_{i,i_1,\ldots,i_n}$ forwards all messages from the external session to $\mathcal{E}$.*

ii) Otherwise (i.e., $j \neq i$), if some user in m is the i_r-th user (for some $r \leq n$), then $H_{i,i_1,\ldots,i_n}$ terminates with an error (because the external session is not the i-th session or does not consist of the users $i_1, \ldots, i_n$).

iii) Otherwise (i.e., in particular, $j \neq i$), $\hat{H}_{i,i_1,\ldots,i_n}$ behaves like H_i, see Definition 4 i–ii).

Finally, if $\mathcal{E}$ produces output m on tape decision *(which then terminates the run with overall output m), then $\hat{H}_{i,i_1,\ldots,i_n}$ verifies that the i-th session exists, i.e., that (at some point) $\hat{H}_{i,i_1,\ldots,i_n}$ has forwarded a session-create message to the external session. If this is the case, then $\hat{H}_{i,i_1,\ldots,i_n}$ outputs m on* decision. *Otherwise, $\hat{H}_{i,i_1,\ldots,i_n}$ outputs 0 on* decision. *(We note that this is needed in the proof of Lemma 2).*

Next, we basically show that (under the condition that $i_1, \ldots, i_n$ correctly specify the i-th session) H_i is indistinguishable from $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$ and that H_{i+1} is indistinguishable from $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{R}$.

First, we define events to reason about the i-th session in a run of the hybrid systems and we develop further notation: For every $i, i', i_1, \ldots, i_n \in \mathbf{N}$, we define $B_{H_{i'}}(i, i_1, \ldots, i_n)$ to be the set of runs of $H_{i'}$ where at some point in the run $\mathcal{E}$ sends the i-th *session-create* message without corrupted users (as defined above), say this message is m, and the user in role r in m is the i_r-th user (as defined above), for every $r \leq n$. (That is, in particularly, the i-th session has been created.) Furthermore, for every $i, i' \in \mathbf{N}$, we define $B_{H_{i'}}(i)$ to be the set of runs of $H_{i'}$ where the i-th session is never created, i.e., where $\mathcal{E}$ never sends an i-th *session-create* message without corrupted users (as defined above). We note that because $\mathcal{E}$ is bounded by $p_\mathcal{E}$, $B_{H_{i'}}(i, i_1, \ldots, i_n) = \emptyset$ for all $i, i', i_1, \ldots, i_n$ where $i_r > p_\mathcal{E}$ for some $r \leq n$. Hence, for all $i, i' \in \mathbf{N}$, it holds that

$$B_{H_{i'}}(i) \cup \bigcup_{i_1,\ldots,i_n \leq p_\mathcal{E}} B_{H_{i'}}(i, i_1, \ldots, i_n) \tag{5}$$

is the disjoint union of all runs of $H_{i'}$.

By $\Pr[\mathcal{S} \rightsquigarrow 1]$, we denote the probability that the a run of the system $\mathcal{S}$ produces overall output 1 (i.e., the output on the decision tape is 1). Furthermore, given a set A of runs of $\mathcal{S}$, by $\Pr[A]$ we denote the sum of the probabilities of the runs in A. We consider $\{\mathcal{S} \rightsquigarrow 1\}$ as the set of all runs of $\mathcal{S}$ that produce overall output 1, i.e., $\Pr[\mathcal{S} \rightsquigarrow 1] = \Pr[\{\mathcal{S} \rightsquigarrow 1\}]$. We will often omit the curly braces around $\{\mathcal{S} \rightsquigarrow 1\}$. For example, we write $\Pr[\mathcal{S} \rightsquigarrow 1 \cap A]$ to denote $\Pr[\{\mathcal{S} \rightsquigarrow 1\} \cap A]$ (i.e., the probability of all runs in A that produce overall output 1).

Lemma 2. *For all $i, i_1, \ldots, i_n \in \mathbf{N}$ it holds that*

$$\Pr[\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I} \rightsquigarrow 1] = \Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i, i_1, \ldots, i_n)] \tag{6}$$

$$\Pr[\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{R} \rightsquigarrow 1] = \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i, i_1, \ldots, i_n)] \ . \tag{7}$$

Proof. We first show (6). In both systems H_i and $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$, the i-th session is treated as $\mathcal{I}$. It is easy to define an injective mapping β from $B_{H_i}(i, i_1, \ldots, i_n)$ (i.e., the set of runs of H_i where the i-th session exists and consists of the users $i_1, \ldots, i_n$) to runs of $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$ such that for every run $\rho \in B_{H_i}(i, i_1, \ldots, i_n)$, the probability of ρ and $\beta(\rho)$ is the same and ρ outputs 1 (i.e., the overall output on the decision tape is 1) iff $\beta(\rho)$ outputs 1. The run $\beta(\rho)$ is obtained from ρ by outsourcing the i-th session to $\mathcal{I}$. This can be done without changing the view of the environment or anything else because it is independent from any other session. Let $\mathrm{rng}(\beta)$ be the range of β, i.e., $\mathrm{rng}(\beta)$ is the set of runs ρ' of $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$ such that there exists a run $\rho \in B_{H_i}(i, i_1, \ldots, i_n)$ with $\beta(\rho) = \rho'$. That is, by β, we have shown that

$$\Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i, i_1, \ldots, i_n)] = \Pr[\{\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I} \rightsquigarrow 1\} \cap \mathrm{rng}(\beta)] \ . \tag{8}$$

Note that not every run of $\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}$ is in the range of β, i.e., β is not surjective. Next, we show that every such run does not output 1: Therefore, let ρ' be a run of $\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}$ such that $\rho' \notin \mathrm{rng}(\beta)$, i.e., there does not exist a run $\rho \in B_{H_i}(i,i_1,\dots,i_n)$ with $\alpha(\rho) = \rho'$. First, we note that if the i-th session does not exists in ρ', then by definition of $\hat{H}_{i,i_1,\dots,i_n}$, $\hat{H}_{i,i_1,\dots,i_n}$ never outputs 1. So, in the following, assume that the i-th session has been created. If the i-th session in ρ' would consist of the users $i_1,\dots,i_n$ (i.e., let m be the i-th *session-create* message without corrupted users that $\mathcal{E}$ sends to $\mathcal{F}'$ in ρ', then the user for role r in m is the i_r-th user in ρ', for all $r \leq n$), then there would exist $\rho \in B_{H_i}(i,i_1,\dots,i_n)$ with $\alpha(\rho) = \rho'$. (This ρ could easily be constructed from ρ'.) Hence, we can conclude that, by definition of $\hat{H}_{i,i_1,\dots,i_n}$, that $\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}$ terminates with an error. In particular, the output of ρ' is not 1. We have now shown that

$$\Pr[\{\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1\} \setminus \mathrm{rng}(\beta)] = 0 \ . \tag{9}$$

Since the probability that a run of $\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}$ outputs 1 (i.e., $\Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1]$) is the sum of the probability that a run in the range of β outputs 1 plus the probability that a run of $\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}$ that is not in the range of β outputs 1, by the above, we conclude as follows:

$$\begin{aligned}\Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1] &= \Pr[\{\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I}\} \rightsquigarrow 1 \cap \mathrm{rng}(\beta)] + \Pr[\{\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1\} \setminus \mathrm{rng}(\beta)] \\ &\overset{(8),(9)}{=} \Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i,i_1,\dots,i_n)] \ .\end{aligned}$$

The proof of (7) is similar to the proof of (6). We only have to replace the external system $\mathcal{I}$ by $\mathcal{R}$. □

Finally, to prove Theorem 3, we combine (3), (4), (6), and (7) to show that $\mathcal{E}\,|\,\mathcal{P}\,|\,\mathcal{F}' \equiv \mathcal{E}\,|\,\mathit{Sim}\,|\,\mathcal{F}$. For all $i,i_1,\dots,i_n \in \mathbf{N}$, let $\delta_{i,i_1,\dots,i_n}$ be the advantage of $\hat{H}_{i,i_1,\dots,i_n}$ in distinguishing between $\mathcal{I}$ and $\mathcal{R}$. More formally:

$$\delta_{i,i_1,\dots,i_n} := \left|\Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{R} \rightsquigarrow 1] - \Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1]\right| \ .$$

Next, we define an environment $\hat{H}_\$$ for $\mathcal{R}$ (and, hence, for $\mathcal{I}$). We define $\hat{H}_\$$ to be the IITM that chooses $i,i_1,\dots,i_n \leq p_\mathcal{E}$ such that it maximizes $\delta_{i,i_1,\dots,i_n}$ (this can easily be done in polynomial-time by computing all $p_\mathcal{E}^{n+1}$ possible values $\delta_{i,i_1,\dots,i_n}$) and then, $\hat{H}_\$$ emulates $\hat{H}_{i,i_1,\dots,i_n}$. By construction of $\hat{H}_\$$, for all $i,i_1,\dots,i_n \leq p_\mathcal{E}$, we have that

$$\delta_{i,i_1,\dots,i_n} \leq \left|\Pr[\hat{H}_\$\,|\,\mathcal{R} \rightsquigarrow 1] - \Pr[\hat{H}_\$\,|\,\mathcal{I} \rightsquigarrow 1]\right| \ . \tag{10}$$

Now, we have that

$$\begin{aligned}&\left|\Pr[H_1 \rightsquigarrow 1] - \Pr[H_{p_\mathcal{E}+1} \rightsquigarrow 1]\right| \\ &\leq \sum_{i \leq p_\mathcal{E}} \left|\Pr[H_i \rightsquigarrow 1] - \Pr[H_{i+1} \rightsquigarrow 1]\right| \\ &\overset{(5)}{\leq} \sum_{i,i_1,\dots,i_n \leq p_\mathcal{E}} \left|\Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i,i_1,\dots,i_n)] - \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i,i_1,\dots,i_n)]\right| \\ &\quad + \sum_{i \leq p_\mathcal{E}} \left|\Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i)] - \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i)]\right| \ .\end{aligned}$$

It is easy to see that $\Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i)] - \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i)] = 0$ for all $i \leq p_\mathcal{E}$ because, in runs where there exists no i-th session, the systems behave identically. Using Lemma 2, we can further conclude that

$$\begin{aligned}&\sum_{i,i_1,\dots,i_n \leq p_\mathcal{E}} \left|\Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i,i_1,\dots,i_n)] - \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i,i_1,\dots,i_n)]\right| \\ &\overset{(6),(7)}{=} \sum_{i,i_1,\dots,i_n \leq p_\mathcal{E}} \left|\Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{I} \rightsquigarrow 1] - \Pr[\hat{H}_{i,i_1,\dots,i_n}\,|\,\mathcal{R} \rightsquigarrow 1]\right| \\ &\overset{(10)}{\leq} p_\mathcal{E}^{n+1} \cdot \left|\Pr[\hat{H}_\$\,|\,\mathcal{R} \rightsquigarrow 1] - \Pr[\hat{H}_\$\,|\,\mathcal{I} \rightsquigarrow 1]\right| \ .\end{aligned}$$

By (2), we have that $\hat{H}_\$ \,|\, \mathcal{R} \equiv \hat{H}_\$ \,|\, \mathcal{I}$, i.e., the above is negligible (as a function in the security parameter) because the number of roles n is constant (it does not depend on the security parameter) and $p_\mathcal{E}$ is a polynomial (in the security parameter). So, we have shown that $H_1 \equiv H_{p_\mathcal{E}+1}$. By (3) and (4), we further conclude that $\mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F} \equiv \mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}'$, i.e., $\mathcal{P} \,|\, \mathcal{F}' \leq \mathcal{F}$. This concludes the proof of Theorem 3.

C Joint State Composition Without Pre-Established SIDs

In this section, we provide more details for Section 4 and the proof of Theorem 4.

C.1 The Ideal Crypto Functionality

Here, we present more details of the ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ so that the proof of Theorem 4 can be fully understood. Full details of $\mathcal{F}_{\text{crypto}}$ can be found in [30].

As mentioned in Section 4.1, parties can use $\mathcal{F}_{\text{crypto}}$ i) to generate symmetric keys, including pre-shared keys, ii) to derive symmetric keys from other symmetric keys, iii) to encrypt and decrypt bit strings (public-key encryption and both unauthenticated and authenticated symmetric encryption is supported), iv) to compute and verify MACs and digital signatures, and v) to generate fresh nonces, where all the above operations are done in an ideal way. All symmetric and public keys can be part of plaintexts to be encrypted under other symmetric and public keys. We emphasize that derived keys can be used just as other symmetric keys. It is left up to the protocol that uses $\mathcal{F}_{\text{crypto}}$ how to interpret (parts of) bit strings, e.g., as length fields, nonces, ciphertexts, MACs, digital signatures, non-interactive zero-knowledge proofs, etc.

The ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ is parametrized by what is called a *leakage algorithm* L, a probabilistic polynomial time algorithm which takes as input a security parameter η and a message x, and returns the information that may be leaked about x. Typical examples are i) $L(1^\eta, x) = 0^{|x|}$ and ii) the algorithm that returns a random bit string of length $|x|$. Both leakage algorithms leak exactly the length of x. In this paper, we always use the leakage algorithm given in the second example which returns a random bit string of the length of the message. In the proof of Theorem 4, we need that collisions among ciphertexts that have been ideally produced, i.e., are the result of encrypting $\overline{x} = L(1^\eta, x')$ instead of the actual plaintext x' (see below), occur only with negligible probability. As discussed in [28], this can be guaranteed by our choice of the leakage algorithm and assuming that the domain of plaintexts only contains long messages, i.e., of length at least η. We assume this in this paper.[4] The functionality $\mathcal{F}_{\text{crypto}}$ is also parameterized by a number $n \geq 2$ which defines the number of roles in a protocol that uses $\mathcal{F}_{\text{crypto}}$; $\mathcal{F}_{\text{crypto}}$ has one I/O input and output tape for each role.

In $\mathcal{F}_{\text{crypto}}$, symmetric keys are equipped with types. Keys that may be used for authenticated encryption have type authenc-key, those for unauthenticated encryption have type unauthenc-key. We have the types mac-key for MAC keys and pre-key for keys from which new keys can be derived. All types are disjoint, i.e., a key can only have one type, reflecting common practice that a symmetric key only serves one purpose. For example, a MAC key is not used for encryption and keys from which other keys are derived are typically not used as encryption/MAC keys.

While users of $\mathcal{F}_{\text{crypto}}$, and its realization, are provided with the actual public keys generated within $\mathcal{F}_{\text{crypto}}$ (the corresponding private keys remain in $\mathcal{F}_{\text{crypto}}$), they 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. These pointers may be part of the messages given to $\mathcal{F}_{\text{crypto}}$ for encryption. Before a message is actually encrypted, the pointers are replaced by the keys they refer to. Upon decryption of a ciphertext, keys embedded in the plaintext are first turned into pointers before the plaintext is given to the user. In order to be able to identify pointers/keys, we assume pointers/keys in plaintexts to be tagged according to their types. Such messages are called *well-tagged* messages. As discussed in [30], for real-world protocols, it is typically possible to define tagging in such a way that the message formats used in these

[4] Alternatively, one could assume that the encryption schemes provided by the adversary guarantee this property. Since IND-CPA security implies this property, this assumption would be fulfilled by the realization of $\mathcal{F}_{\text{crypto}}$.

protocols is captured precisely on the bit level. Furthermore, if one considers protocols which use the same keys but different tagging schemes, even then in most cases it should be possible to define tagging such that both protocols are precisely modeled. In particular, this is the case if the message formats of the protocols have a syntactically different structure because the tagging function can then branch on the protocol and implement different tagging schemes for each protocol.

A *user* of $\mathcal{F}_{\text{crypto}}$ is identified, within $\mathcal{F}_{\text{crypto}}$, by the tuple (p, lsid, r), where p is a party name, $r \leq n$ a role, and *lsid* a local SID; similar to multi-session protocols and local-SID functionalities. In particular, on the tape for role r, $\mathcal{F}_{\text{crypto}}$ expects requests to be prefixed by tuples of the form (lsid, p), and conversely $\mathcal{F}_{\text{crypto}}$ prefixes answers with (lsid, p).

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, among others, $\mathcal{F}_{\text{crypto}}$ 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.

Before any cryptographic operation can be performed, $\mathcal{F}_{\text{crypto}}$ expects to receive (descriptions of) algorithms from the ideal adversary for symmetric and public-key encryption/decryption as well as the generation and verification of MACs and digital signatures. Also, $\mathcal{F}_{\text{crypto}}$ expects to receive public/private keys for encryption/decryption and verifying/signing from the ideal adversary. In the realization of $\mathcal{F}_{\text{crypto}}$, we assume that parties know the public keys of the other parties, e.g., because they use some kind of public-key infrastructure. We do not put any restrictions on these algorithms; all security guarantees that $\mathcal{F}_{\text{crypto}}$ provides are made explicit within $\mathcal{F}_{\text{crypto}}$ without relying on specific properties of these algorithms. As a result, when using $\mathcal{F}_{\text{crypto}}$ in the analysis of systems, one can abstract from these algorithms entirely. We now sketch the operations that $\mathcal{F}_{\text{crypto}}$ provides.

Generating fresh, symmetric keys $[(\mathsf{New}, t)]$**.** A user (p, lsid, r) can ask $\mathcal{F}_{\text{crypto}}$ to generate a new key of type $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$. The request is forwarded to the adversary who is supposed to provide such a key, say the bit string k. The adversary can decide to corrupt k right away, in which case k is added to $\mathcal{K}_{\text{known}}$, and otherwise k is added to $\mathcal{K}_{\text{unknown}}$. However, before adding k to a set, $\mathcal{F}_{\text{crypto}}$ ensures that k is fresh and key guessing is prevented, i.e., in case k is uncorrupted, it may not belong to $\mathcal{K}$, and in case k is corrupted, it may not belong to $\mathcal{K}_{\text{unknown}}$. If $\mathcal{F}_{\text{crypto}}$ accepts k, a new pointer *ptr* to k is created (by increasing a counter) and returned to (p, lsid, r).

Establishing pre-shared keys $[(\mathsf{GetPSK}, t, \mathit{name})]$**.** This request is similar to (New, t). However, if $\mathcal{F}_{\text{crypto}}$ already recorded a key under (t, name), a new pointer to this key is returned. In particular, if different users invoke this command with the same name and type, they are provided with pointers to the same key. This allows users to establish shared keys: For example, users (p, lsid, r) and (p', lsid', r') can obtain pointers to a fresh key k shared between p and p' by each sending the request $(\mathsf{GetPSK}, t, (p, p'))$ to $\mathcal{F}_{\text{crypto}}$. While, by such a request, p (p') gets a new pointer in every local session and role, this pointer will point to the same key k.

Key derivation $[(\mathsf{Derive}, \mathit{ptr}, t, s)]$**.** A user (p, lsid, r) can ask to derive a key of type $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$ from a seed s (an arbitrary bit string) and a key, say k, of type $\mathsf{pre\text{-}key}$ the pointer *ptr*, which has to belong to the user, points to. This request is forwarded to the adversary, who is supposed to provide a new key, similarly to the request (New, t). However, the adversary may not corrupt this key; it is considered to be unknown if and only if k is unknown. Furthermore, if there already exists a key k' derived from k and s—a fact that $\mathcal{F}_{\text{crypto}}$ keeps track of—, the adversary has to provide exactly this key k'. This guarantees that key derivation requests with the same key, type, and seed yield a pointer to the same key.

Encryption $[(\mathsf{Enc}, \mathit{ptr}, x), (\mathsf{PKEnc}, p', \mathit{pk}, x)]$ **and decryption** $[(\mathsf{Dec}, \mathit{ptr}, y), (\mathsf{PKDec}, y)]$**.** We concentrate on unauthenticated encryption/decryption (see Section 4.1 for authenticated encryption/decryption; public-key encryption under the public key of party p' and decryption under the private key of party p is similar to unauthenticated encryption/decryption). A user (p, lsid, r) can ask to encrypt a well-tagged message x using a pointer *ptr* that has to belong to the user and points to a key, say k, of type $\mathsf{unauthenc\text{-}key}$. We first consider the case that $k \in \mathcal{K}_{\text{unknown}}$. First, all pointers in x, which again have to belong to the user, are replaced by the actual keys, resulting in a message x'. Then, the *leakage* $\overline{x} = L(1^\eta, x')$ of x' is encrypted under k using the encryption algorithm previously provided by the adversary (see above). The resulting ciphertext y' (if any) is returned to the user and (x', y') is stored by $\mathcal{F}_{\mathrm{crypto}}$ for later decryption of y' under k. Upon decryption of y, $\mathcal{F}_{\mathrm{crypto}}$ first checks if exactly one pair of the form (x'', y) is stored. In this case, x'' with embedded keys replaced by pointers is returned to the user. If there are more than one such a pair, then an error is returned to the user because unique decryption is impossible. If no such pair is stored, the adversary is asked for a plaintext x'' which is returned to the user as above.[5] However, $\mathcal{F}_{\mathrm{crypto}}$ only accepts x'' if no embedded key is marked unknown, i.e., is in $\mathcal{K}_{\mathrm{unknown}}$ (this prevents "guessing" of keys). Furthermore, $\mathcal{F}_{\mathrm{crypto}}$ adds every key embedded in x'' to $\mathcal{K}_{\mathrm{known}}$ (if it is not already in $\mathcal{K}_{\mathrm{known}}$). In case *ptr* points to a key marked known, i.e., a key in $\mathcal{K}_{\mathrm{known}}$, the adversary is asked for a ciphertext (in case of encryption) or a plaintext (in case of decryption). In case of decryption, this plaintext is treated exactly as the plaintext provided by the adversary above.

Computing and verifying MACs and digital signatures $[(\mathsf{Mac}, \mathit{ptr}, x)$, $(\mathsf{MacVerify}, \mathit{ptr}, x, \sigma)$, (Sign, x), $(\mathsf{SigVerify}, p', \mathit{pk}, x, \sigma)]$**.** We concentrate on computing and verifying MACs (signing with the private key of party p and verifying signatures under the public key *pk* of party p' are similar). A user (p, lsid, r) can ask $\mathcal{F}_{\mathrm{crypto}}$ to MAC an *arbitrary (uninterpreted) bit string* x using a pointer *ptr* that has to belong to the user and points to a key, say k, of type $\mathsf{mac\text{-}key}$. Then, $\mathcal{F}_{\mathrm{crypto}}$ computes the MAC of x under k using the MAC algorithm previously provided by the adversary. The resulting MAC σ (if any) is returned to the user. If $k \in \mathcal{K}_{\mathrm{unknown}}$, $\mathcal{F}_{\mathrm{crypto}}$ records x for later verification with k; σ is not recorded since we allow an adversary to derive a new MAC from a given one on the same message.

For verification, $\mathcal{F}_{\mathrm{crypto}}$ runs the MAC verification algorithm previously provided by the adversary on x, σ, and k. If $k \in \mathcal{K}_{\mathrm{known}}$, $\mathcal{F}_{\mathrm{crypto}}$ returns the result of the verification to the user. If $k \in \mathcal{K}_{\mathrm{unknown}}$, this is done too, but success is only returned if x previously has been recorded for k.

Generating fresh nonces $[(\mathsf{NewNonce})]$**.** Similarly to generating fresh keys, nonces can be generated by users, where nonces are guaranteed to not collide.

Store $[(\mathsf{Store}, t, k)]$**.** A user (p, lsid, r) can ask $\mathcal{F}_{\mathrm{crypto}}$ to *store* some bit string k with some type $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$ as a key. Then, $\mathcal{F}_{\mathrm{crypto}}$ forwards this request to the adversary who is supposed to reply with $\mathit{fail} \in \{\mathsf{false}, \mathsf{true}\}$. If $\mathit{fail} = \mathsf{true}$ or k belongs to $\mathcal{K}_{\mathrm{unknown}}$, $\mathcal{F}_{\mathrm{crypto}}$ will return an error message to the user.[6] Otherwise, $\mathcal{F}_{\mathrm{crypto}}$ creates a new pointer to k which is given to the user. The key k is added to $\mathcal{K}_{\mathrm{known}}$.

Retrieve $[(\mathsf{Retrieve}, \mathit{ptr})]$**.** A user (p, lsid, r) can ask $\mathcal{F}_{\mathrm{crypto}}$ to *retrieve* a key k pointer *ptr* points to for user (p, lsid, r). Then, $\mathcal{F}_{\mathrm{crypto}}$ informs the adversary that k is retrieved[7], adds k to $\mathcal{K}_{\mathrm{known}}$, and returns k to the user.

Further operations For further operations, including the request for public keys, checking the corruption status of keys, and checking whether two pointers point to the same key we refer the reader to [30].

This concludes the description of $\mathcal{F}_{\mathrm{crypto}}$. Next, we further comment on the slight modifications to the original $\mathcal{F}_{\mathrm{crypto}}$ in [30]. The modifications comprise non-ideal encryption/decryption (e.g., under known keys) and storing and retrieving of keys, see above. In the proof of Theorem 4 (more precisely in the proof of in Lemma 6 in Appendix C.2), we need that the adversary knows which keys are marked known (i.e., in $\mathcal{K}_{\mathrm{known}}$) and which are marked unknown (i.e., in $\mathcal{K}_{\mathrm{unknown}}$). Furthermore, the adversary needs to have the ability to prevent the environment (i.e., some user) to add a particular key to the set of known keys $\mathcal{K}_{\mathrm{known}}$. Our modifications enable the adversary to do this. We note that since we only strengthen the abilities of the adversary, every realization of the original version of $\mathcal{F}_{\mathrm{crypto}}$ is a realization of our modified version of $\mathcal{F}_{\mathrm{crypto}}$. But we did

[5] As already mentioned in Section 4.1 for authenticated encryption, this constitutes a slight modification to the original ideal functionality in [30]. See below for further remarks.

[6] This constitutes a slight modification to the original ideal functionality in [30], where the adversary is not asked before storing. See below for further remarks.

[7] This constitutes a slight modification to the original ideal functionality in [30], where the adversary is not informed about retrieve requests. See below for further remarks.

not strengthen the adversary too much, i.e., our version of $\mathcal{F}_{\text{crypto}}$ is just as useful for analyzing protocols as the original version.

C.2 The Joint State Composition Theorem without Pre-Established SIDs

Before we prove Theorem 4, we formally define the notion of explicitly shared symmetric keys, which has informally been introduced in Section 4.2.

Definition 6. *Let $\mathcal{P}$ be a multi-session protocol that uses $\mathcal{F}_{\text{crypto}}$, τ be a partnering function for $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$, and $\mathcal{E}$ be an environment for $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$. We (inductively) define that a symmetric key k in $\mathcal{F}_{\text{crypto}}$ is* explicitly shared *in a run ρ of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ w.r.t. τ if there exist two distinct users $(p, lsid, r)$ and $(p', lsid', r')$ which are* not *partners (i.e., $\tau_{(p,lsid,r)}(\rho) \neq \tau_{(p',lsid',r')}(\rho)$ or $\tau_{(p,lsid,r)}(\rho) = \tau_{(p',lsid',r')}(\rho) = \bot$) and* not *both corrupted in ρ (i.e., the flag* **corrupted** *is set to* false *in at least one of the corresponding instances of M_r or $M_{r'}$, respectively) such that*

i) both users have sent a request to $\mathcal{F}_{\text{crypto}}$ to setup a pre-shared key and both obtained a pointer to k (i.e., the names used in the two requests coincided) or

ii) both users have sent a request to $\mathcal{F}_{\text{crypto}}$ to derive a key from the same explicitly shared key using the same seed and both obtained a pointer to k.

Proof of Theorem 4. The proof is structured into two steps: First, we show that $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ is indistinguishable from the IITM $\mathcal{Q}_\tau$ 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). Second, we show that $\mathcal{Q}_\tau$ realizes $\mathcal{F}$. We note that these two steps are independent from each other. The first step only requires that $\mathcal{P}$ satisfies implicit disjointness but not that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$. In contrast, the second step only requires that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$ but not that $\mathcal{P}$ satisfies implicit disjointness. We also note that $\mathcal{Q}_\tau$ is very similar to a multi-session protocol that uses a multi-session local-SID functionality as in Theorem 3. The proof that $\mathcal{Q}_\tau$ realizes $\mathcal{F}$ is indeed similar to the proof of Theorem 3.

Throughout this proof, we fix a multi-session protocol $\mathcal{P}$ that uses $\mathcal{F}_{\text{crypto}}$ and a partnering function τ such that τ is valid for $\mathcal{P}$. Furthermore, let $\mathcal{E}$ be an environment for $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ and let $p_{\mathcal{E}}$ be a polynomial (in the security parameter η) that bounds the overall runtime (i.e., taken steps) of $\mathcal{E}$. (By definition of environmental systems in the IITM model such a polynomial exists.) Since only $\mathcal{E}$ can create new instances of machines in $\mathcal{P}$ by sending requests to them, the overall number of these instances is bounded by $p_{\mathcal{E}}$. In the following, we explicitly mention where the assumptions that $\mathcal{P}$ satisfies implicit disjointness w.r.t. τ and that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$ are needed.

Collisions of Ciphertexts. First, we prove a general lemma that shows that ciphertext produced by $\mathcal{F}_{\text{crypto}}$ do not collide in our setting (except with negligible probability). We say that an instance of $\mathcal{F}_{\text{crypto}}$ is *(ciphertext) collision free* if there does not exist bit strings x, x', y such that $x \neq x'$, (x, y) is recorded (upon ideal encryption, i.e., under a unknown/uncorrupted key) for the some (symmetric or public/private) key, and (x', y) is recorded for some (possibly different) key. The next lemma shows that $\mathcal{F}_{\text{crypto}}$ is collision free with overwhelming probability.

Lemma 3. *The probability that, in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$, $\mathcal{F}_{\text{crypto}}$ is always collision free is overwhelming (in the security parameter).*

Proof. First, we note that we do not need to consider non-ideally produced ciphertext (i.e., ciphertexts provided by the environment) because they are not stored in $\mathcal{F}_{\text{crypto}}$. Only ideally produced ciphertexts, i.e., ciphertexts which are the encryption of the leakage of a message, are stored in $\mathcal{F}_{\text{crypto}}$. The leakage algorithm L, given x, outputs a random message of length $|x|$. Furthermore, as mentioned in Appendix C.1, we assume that the domain of plaintexts contains only long messages, i.e., of length $\geq \eta$ (where η is the security parameter). Hence, the probability that two leakages (i.e., messages returned by the leakage algorithm) are the same is negligible. Since the decryption of an encryption of a leakage yields the leakage, different leakages encrypt to different ciphertexts. So, such ciphertexts collide only with negligible probability. □

No Implicitly Shared (Symmetric) Keys. The next lemma basically says that unknown symmetric keys that are not explicitly shared are not shared among sessions. This lemma holds because an unknown symmetric key k which is not explicitly shared is either a freshly generated key or derived (directly or indirectly) from a freshly generated key. Hence, k originates from one user. Since k is unknown, another user can only obtain a pointer to k by decrypting a ciphertext which contains k (or a key in the chain of derivations). Implicit disjointness guarantees that only users in the same session decrypt a ciphertext containing k and accept. Hence, only users of the same session can have a pointer to k.

Lemma 4. *If $\mathcal{P}$ satisfies implicit disjointness w.r.t. τ, then the following holds with overwhelming probability, where the probability is over runs ρ of $\mathcal{E}\,|\,\mathcal{P}\,|\,\mathcal{F}_{\text{crypto}}$: If two distinct users (p, lsid, r) and (p', lsid', r') both have a pointer to a key k which is marked unknown in $\mathcal{F}_{\text{crypto}}$ and k is not explicitly shared (in ρ), then both users are partners in ρ (i.e., $\tau_{(p,\mathit{lsid},r)}(\rho) = \tau_{(p',\mathit{lsid}',r')}(\rho) \neq \bot$) or both users are corrupted.*

Proof. Let ρ be a run of $\mathcal{E}\,|\,\mathcal{P}\,|\,\mathcal{F}_{\text{crypto}}$ such that $\mathcal{F}_{\text{crypto}}$ in ρ is always collision free and (a) and (b) in Definition 2 are satisfied. We show that for ρ the statement of the lemma holds. By Lemma 3 and because $\mathcal{P}$ satisfies implicit disjointness w.r.t. τ, we then have shown the statement of the lemma for an overwhelming set of runs.

We show this by the method of considering a minimal counterexample. Assume that there exist a symmetric key k (in $\mathcal{F}_{\text{crypto}}$ in ρ) such that i) k is marked unknown in $\mathcal{F}_{\text{crypto}}$, ii) k is not explicitly shared, iii) there exists two distinct users (p, lsid, r) and (p', lsid', r') that both have a pointer to k, and iv) not both users are partners or not both users are corrupted. Furthermore, we assume that k is the first such key in ρ, i.e., for every other such a key k', it holds that k' has been added to $\mathcal{K}$ (the set of all keys in $\mathcal{F}_{\text{crypto}}$) after k has been added to $\mathcal{K}$. We will lead this to a contradiction, which proves the lemma.

Let (p, lsid, r) be the user that obtained the first pointer to k in ρ. Furthermore, let m be the request that (p, lsid, r) sent to $\mathcal{F}_{\text{crypto}}$ such that the response of $\mathcal{F}_{\text{crypto}}$ contained the first pointer to k. Furthermore, let $(p', \mathit{lsid}', r') \neq (p, \mathit{lsid}, r)$ be the first user that obtained a pointer to k in ρ such that (p', lsid', r') and (p, lsid, r) are not partners or not both (p', lsid', r') and (p, lsid, r) are corrupted. (By assumption such a user (p', lsid', r') exists.) Let m' be the first request that (p', lsid', r') sent to $\mathcal{F}_{\text{crypto}}$ such that the response of $\mathcal{F}_{\text{crypto}}$ contained the first pointer to k for (p', lsid', r'). By definition of $\mathcal{F}_{\text{crypto}}$, new pointers are only generated upon the following requests: (New, t), (Store, t, k), $(\mathsf{GetPSK}, t, \mathit{name})$, $(\mathsf{Derive}, \mathit{ptr}, t, s)$, $(\mathsf{Dec}, \mathit{ptr}, y)$, or (PKDec, y). Next, we show that m' cannot be such a requests, which is the desired contradiction.

Of course, by definition of $\mathcal{F}_{\text{crypto}}$ and because k is unknown, m' is not of the form (New, t) or (Store, t, k).

If m' is of the form $(\mathsf{GetPSK}, t, \mathit{name})$, then, by definition of $\mathcal{F}_{\text{crypto}}$ and because k is unknown, m is of the form $(\mathsf{GetPSK}, t, \mathit{name})$ too, because the response of m contained the first pointer to k. Since k is not explicitly shared, (p, lsid, r) and (p', lsid', r') are partners or both are corrupted. Contradiction. Hence, m' is not of the form $(\mathsf{GetPSK}, t, \mathit{name})$.

If m' is of the form $(\mathsf{Derive}, \mathit{ptr}', t, s)$, then, by definition of $\mathcal{F}_{\text{crypto}}$ and because k is unknown, m is of the form $(\mathsf{Derive}, \mathit{ptr}, t, s)$ and the pointers ptr (of (p, lsid, r)) and ptr' (of (p', lsid', r')) point to the same key, say k', in $\mathcal{F}_{\text{crypto}}$. Furthermore, k' is unknown, i.e., i) holds for k'. If k' would be explicitly shared, then k also would be explicitly shared. Hence, ii) holds for k'. Also, iii) and iv) hold for k' because (p, lsid, r) and (p', lsid', r') have a pointer to k'. But this contradicts the minimality of k. Hence, m' is not of the form $(\mathsf{Derive}, \mathit{ptr}', t, s)$.

If m' is of the form (PKDec, y), then, by definition of $\mathcal{F}_{\text{crypto}}$ and because k is unknown, the decryption request m' was performed ideally by $\mathcal{F}_{\text{crypto}}$, i.e., m' is an ideal destruction request. Furthermore, m' is an accepted destruction request of user (p', lsid', r'). By (b) in Definition 2 (implicit disjointness), there exists some user $(p'', \mathit{lsid}'', r'')$ that has sent a corresponding (to m') construction request such that (p', lsid', r') and $(p'', \mathit{lsid}'', r'')$ are partners or both (p', lsid', r') and $(p'', \mathit{lsid}'', r'')$ are corrupted. That is, $(p'', \mathit{lsid}'', r'')$ has a pointer to k and encrypted k under the public key of p'. Since (p', lsid', r') does not has a pointer to k before it receives the response to m', it holds that $(p'', \mathit{lsid}'', r'') \neq (p', \mathit{lsid}', r')$. Since (p', lsid', r') and $(p'', \mathit{lsid}'', r'')$ are partners or both (p', lsid', r') and $(p'', \mathit{lsid}'', r'')$ are corrupted, it holds that (p, lsid, r) and $(p'', \mathit{lsid}'', r'')$ are not partners or not both (p, lsid, r) and $(p'', \mathit{lsid}'', r'')$ are corrupted. But this contradicts the assumption that (p', lsid', r') is the first such user. Hence, m' is not of the form (PKDec, y).

Similarly to the case of public-key encryption, if m' is of the form (Dec, ptr, y), then, by definition of $\mathcal{F}_{\text{crypto}}$ and because k is unknown, the decryption request m' was performed ideally by $\mathcal{F}_{\text{crypto}}$, i.e., m' is an ideal destruction request; in particular, the key, say k', pointer ptr points to is unknown. Furthermore, m' is an accepted destruction request of user $(p', lsid', r')$. If k' is an explicitly shared key, then (b) in Definition 2 (implicit disjointness) holds for k' and exactly as above, we can show that m' is not of the form (Dec, ptr, y). Now, assume that k' is not an explicitly shared key. Then i) and ii) hold for k'. Also, iii) and iv) hold for k' because $(p, lsid, r)$ and $(p', lsid', r')$ have a pointer to k'. But this contradicts the minimality of k. Hence, m' is not of the form (Dec, ptr, y).

Altogether, we conclude that m' is not a request of the form (New, t), (Store, t, k), $(\mathsf{GetPSK}, t, name)$, $(\mathsf{Derive}, ptr, t, s)$, (Dec, ptr, y), or (PKDec, y), which is a contradiction. □

Definition of $\mathcal{Q}_\tau$. Before we define $\mathcal{Q}_\tau$ we introduce additional notation. We say that two collision free instances $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$ have *compatible states* if they correspond on the algorithms (provided by the environment) and on the status of keys. More formally, we say that $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$ have *compatible states* if the following holds:

i) $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$ correspond on all algorithms (e.g., for symmetric and public-key encryption/decryption) which have been provided by the environment.
ii) A public/private key of some party is corrupted in $\mathcal{F}^{(1)}_{\text{crypto}}$ iff it is corrupted in $\mathcal{F}^{(2)}_{\text{crypto}}$.
iii) For every symmetric key k (of some type t) that exists in both $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$, k is known in $\mathcal{F}^{(1)}_{\text{crypto}}$ iff k is known in $\mathcal{F}^{(2)}_{\text{crypto}}$. Furthermore, ciphertexts do not collide, i.e., there does not exist bit strings x, x', y such that $x \neq x'$, (x, y) is recorded for k in $\mathcal{F}^{(1)}_{\text{crypto}}$, and (x', y) is recorded for k in $\mathcal{F}^{(2)}_{\text{crypto}}$.
iv) There is no nonce that occurs in both $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$.
v) For every pre-shared key *name* (of some type t) that exists in both $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$, it holds that $(t, name)$ is corrupted in $\mathcal{F}^{(1)}_{\text{crypto}}$ iff $(t, name)$ is corrupted in $\mathcal{F}^{(2)}_{\text{crypto}}$ and it holds that $\mathcal{F}^{(1)}_{\text{crypto}}$ and $\mathcal{F}^{(2)}_{\text{crypto}}$ correspond on the value of the key (in case it is not corrupted).
vi) If the key k' (of some type t) is derived from a key k in $\mathcal{F}^{(1)}_{\text{crypto}}$ using seed s and k'' (of the same type t) is derived from k in $\mathcal{F}^{(2)}_{\text{crypto}}$ using the same seed s, then $k' = k''$.

Two collision free instances of $\mathcal{F}_{\text{crypto}}$ that have compatible states can be merged in the obvious way. The merged instance just contains all keys and, e.g., all recorded pairs of plaintexts/ciphertexts. By definition, the merged instance is collision free as well. Furthermore, several collision free instances of $\mathcal{F}_{\text{crypto}}$ that have pairwise compatible states can all be merge.

Now, we define the IITM $\mathcal{Q}_\tau$.

Definition 7 (The IITM $\mathcal{Q}_\tau$). *The IITM $\mathcal{Q}_\tau$ has the same I/O and network interface as $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$. It emulates $\mathcal{P}$ and several instances of $\mathcal{F}_{\text{crypto}}$ (basically one for every session). For every instance $(p, lsid, r)$ (by $(p, lsid, r)$ we denote the emulated instance of M_r in $\mathcal{P}$ with PID p and LSID lsid), $\mathcal{Q}_\tau$ creates an instance of $\mathcal{F}_{\text{crypto}}$, denoted by $\mathcal{F}^{(p,lsid,r)}_{\text{crypto}}$.*

Whenever some (emulated) instance of $\mathcal{F}_{\text{crypto}}$ produces (I/O or network) output, $\mathcal{Q}_\tau$ checks that all instances of $\mathcal{F}_{\text{crypto}}$ are collision free and have pairwise compatible states. If this is not the case, $\mathcal{Q}_\tau$ terminates. (In this case the environment could distinguish between $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ and $\mathcal{Q}_\tau$ but this happens only with negligible probability.)

Whenever $(p, lsid, r)$ produces (I/O or network) output, $\mathcal{Q}_\tau$ whether $(p, lsid, r)$ is corrupted (i.e., the **corrupted** *flag in $(p, lsid, r)$ is* true*) or $(p, lsid, r)$ belongs to some session (i.e., $\tau(\alpha) \neq \bot$ where α is the sequence of configurations of $(p, lsid, r)$ so far). If this is the case, $\mathcal{Q}_\tau$ sets $s :=$* corrupted *(where* corrupted *is a special symbol distinct from any bit string and $\bot$) or $s := \tau(\alpha)$, respectively. If $\mathcal{F}^{(p,lsid,r)}_{\text{crypto}}$ exists (i.e., it has not been merged with some other instances of $\mathcal{F}_{\text{crypto}}$ and removed afterwards) and $\mathcal{F}^{s}_{\text{crypto}}$ does not exist, then $\mathcal{Q}_\tau$ sets $\mathcal{F}^{s}_{\text{crypto}} := \mathcal{F}^{(p,lsid,r)}_{\text{crypto}}$ and removes $\mathcal{F}^{(p,lsid,r)}_{\text{crypto}}$. Otherwise, if $\mathcal{F}^{(p,lsid,r)}_{\text{crypto}}$ and $\mathcal{F}^{s}_{\text{crypto}}$ exist, then $\mathcal{Q}_\tau$*

merges $\mathcal{F}_{\text{crypto}}^{(p,lsid,r)}$ and $\mathcal{F}_{\text{crypto}}^{s}$ and replaces $\mathcal{F}_{\text{crypto}}^{s}$ by the merged instance of $\mathcal{F}_{\text{crypto}}$. (If $\mathcal{F}_{\text{crypto}}^{(p,lsid,r)}$ does not exists, then $\mathcal{Q}_\tau$ does nothing because it already has been merged with $\mathcal{F}_{\text{crypto}}^{s}$ and removed afterwards.)

Considering the above, $\mathcal{Q}_\tau$ performs the emulation as follows:

(a) Messages from environment to $(p, lsid, r)$: *If $\mathcal{Q}_\tau$ receives (I/O or network) input from the environment for $(p, lsid, r)$, then it just forwards it to $(p, lsid, r)$.*

(b) Messages from $(p, lsid, r)$ to the environment: *If $(p, lsid, r)$ sends a message m to then environment, then $\mathcal{Q}_\tau$ first merges instances of $\mathcal{F}_{\text{crypto}}$ as described above and then forwards m to the environment.*

(c) Requests of $(p, lsid, r)$ to $\mathcal{F}_{\text{crypto}}$: *Upon a request m of $(p, lsid, r)$ to $\mathcal{F}_{\text{crypto}}$, $\mathcal{Q}_\tau$ first merges instances of $\mathcal{F}_{\text{crypto}}$ as described above and then forwards m to the instance of $\mathcal{F}_{\text{crypto}}$ corresponding to $(p, lsid, r)$ (i.e., to $\mathcal{F}_{\text{crypto}}^{\text{corrupted}}$ if $(p, lsid, r)$ is corrupted, to $\mathcal{F}_{\text{crypto}}^{s}$ if $(p, lsid, r)$ has SID s according to τ, or to $\mathcal{F}_{\text{crypto}}^{(p,lsid,r)}$ otherwise).*

(d) Responses of $\mathcal{F}_{\text{crypto}}$ to $(p, lsid, r)$: *If some instance of $\mathcal{F}_{\text{crypto}}$ produces I/O output to $(p, lsid, r)$, then $\mathcal{Q}_\tau$ just forwards it to $(p, lsid, r)$.*

(e) Messages from $\mathcal{F}_{\text{crypto}}$ to the environment: *If some instance $\mathcal{F}'_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$ produces network output m to the environment, then $\mathcal{Q}_\tau$ does the following:*

   i) If m was triggered by a symmetric decryption request (of some instance $(p, lsid, r)$), i.e., m asks the environment to provide a plaintext and m contains a decryption key k and a ciphertext y, then $\mathcal{Q}_\tau$ checks if (x, y) (for some x) has been recorded for the key k in some other instance of $\mathcal{F}_{\text{crypto}}$. If this is the case, then $\mathcal{Q}_\tau$ sends an error message to $\mathcal{F}'_{\text{crypto}}$ (in response to m). This will trigger $\mathcal{F}'_{\text{crypto}}$ to send an error message to $(p, lsid, r)$ signaling that decryption failed. (Note that in $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$, in contrast to $\mathcal{Q}_\tau$, here x would be returned to $(p, lsid, r)$ as the plaintext instead of an error message. But this y has been produced in a different session, hence, because of implicit disjointness, $(p, lsid, r)$ would reject it and its state would be the same as if decryption failed.) Otherwise, $\mathcal{Q}_\tau$ forwards m to the environment.

   ii) Similarly, if m was triggered by a public-key decryption request (i.e., under the private key of party p), $\mathcal{Q}_\tau$ sends an error message to $\mathcal{F}'_{\text{crypto}}$ if the ciphertext y in m is recorded in some other instance of $\mathcal{F}_{\text{crypto}}$ for the public/private key of party p. Otherwise, $\mathcal{Q}_\tau$ forwards m to the environment.

   iii) In any other case (e.g., m was triggered by a symmetric key generation request and asks the environment to provide a fresh key), $\mathcal{Q}_\tau$ forwards m to the environment.

(f) Messages from the environment to $\mathcal{F}_{\text{crypto}}$: *If $\mathcal{Q}_\tau$ receives network input m from the environment for $\mathcal{F}_{\text{crypto}}$, then it does everything to preserve compatible states between the instances of $\mathcal{F}_{\text{crypto}}$. More precisely, $\mathcal{Q}_\tau$ does the following:*

   i) If m contains algorithms for (symmetric or public-key) encryption, MAC, or signature schemes, then $\mathcal{Q}_\tau$ sends m to every instance of $\mathcal{F}_{\text{crypto}}$. Furthermore, if $\mathcal{Q}_\tau$ creates a new instance $\mathcal{F}_{\text{crypto}}^{(p,lsid,r)}$ for new instances $(p, lsid, r)$, then $\mathcal{F}_{\text{crypto}}^{(p,lsid,r)}$ first obtains all algorithms that have been sent by the environment.

   ii) Otherwise, m is the answer to a message sent from some instance of $\mathcal{F}_{\text{crypto}}$ to the environment, e.g., to generate a new symmetric key. Because m contains $(p, lsid, r)$, $\mathcal{Q}_\tau$ knows to which instance of $\mathcal{F}_{\text{crypto}}$, say $\mathcal{F}'_{\text{crypto}}$, m is addressed. (Every $(p, lsid, r)$ belongs to exactly one instance of $\mathcal{F}_{\text{crypto}}$.) Note that $\mathcal{F}_{\text{crypto}}$ when receiving a message from the adversary checks whether this message is valid, e.g., upon key generation, $\mathcal{F}_{\text{crypto}}$ checks whether the key is fresh (i.e., not in $\mathcal{K}$) if it is uncorrupted or, if it is corrupted, that is either fresh or known (i.e., not in $\mathcal{K}_{\text{unknown}}$). If the message is not valid, $\mathcal{F}_{\text{crypto}}$ ignores it (i.e., produces empty output and waits for the next message to receive). Now, $\mathcal{Q}_\tau$ (before forwarding m to $\mathcal{F}'_{\text{crypto}}$) itself checks whether m would be valid for the instance of $\mathcal{F}_{\text{crypto}}$ which is obtained from merging all instances of $\mathcal{F}_{\text{crypto}}$ that $\mathcal{Q}_\tau$ emulates. For example, in case of uncorrupted key generation, $\mathcal{Q}_\tau$ verifies that the key is fresh in all instances of $\mathcal{F}_{\text{crypto}}$. If this check fails, then $\mathcal{Q}_\tau$ ignores m. Otherwise, $\mathcal{Q}_\tau$ forwards m to $\mathcal{F}'_{\text{crypto}}$. (This guarantees that the instances of $\mathcal{F}_{\text{crypto}}$ still have pairwise compatible states.)

Next, we show that $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ is indistinguishable from $\mathcal{Q}_\tau$. As mentioned above, this does not require that $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$.

Lemma 5. *If $\mathcal{P}$ satisfies implicit disjointness w.r.t. τ, then $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{Q}_\tau$.*

Proof. To prove Lemma 5, we define a one-to-one mapping between runs (or at least an overwhelming set of runs) of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ and runs of $\mathcal{E} \,|\, \mathcal{Q}_\tau$ such that corresponding runs have the same probability and overall output.

Therefore, let ρ be a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ such that $\mathcal{F}_{\text{crypto}}$ in ρ is always collision free, (a) and (b) in Definition 2 are satisfied for ρ, and there exists no implicitly shared keys that are unknown in ρ, i.e., the statement of Lemma 4 holds for ρ. Given such a run ρ, we first define the corresponding run ρ' of $\mathcal{E} \,|\, \mathcal{Q}_\tau$. Then, we show that the probability of ρ is the same as the probability of ρ' and that the overall output (i.e., the output on tape decision) is the same in ρ and ρ'. By Lemma 3, Lemma 4, and because $\mathcal{P}$ satisfies implicitly disjointness w.r.t. τ, the set of runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ that we consider has overwhelming probability and, hence, we can conclude that $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{Q}_\tau$.

We define ρ' to be the run of $\mathcal{E} \,|\, \mathcal{Q}_\tau$ where the following holds:

i) $\mathcal{E}$ uses the same random coins as $\mathcal{E}$ in ρ.
ii) The random coins of $\mathcal{Q}_\tau$ are defined such that:
   - The emulated $\mathcal{P}$ uses the same random coins as $\mathcal{P}$ in ρ.
   - The emulated copies of $\mathcal{F}_{\text{crypto}}$ jointly use the same random coins as $\mathcal{F}_{\text{crypto}}$ in ρ.

By construction, the probabilities of ρ and ρ' are equal. W.l.o.g., in the following, we assume that $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ in the run ρ is a single IITM. Furthermore, we assume that $\mathcal{E}$ is a single IITM. In [27], it has been shown that every system of IITMs can be emulated by a single IITM. This simplifies the proof because $\mathcal{Q}_\tau$ is a single IITM in ρ' and, hence, the structure of ρ and ρ' is more similar. In particular, now, every configuration in ρ is either $\mathcal{E}$ sending output to $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ or tape decision (which of course can only be the last configuration of ρ), $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ sending output to $\mathcal{E}$, or $\mathcal{E}$ or $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ producing empty output. Otherwise, we would have to talk about intermediate configurations, e.g., where $\mathcal{P}$ sends requests to $\mathcal{F}_{\text{crypto}}$, which do not exist in ρ'.

By induction on the length of prefixes of ρ, we show that the following holds for every prefix $\hat{\rho}$ of ρ (the corresponding prefix $\hat{\rho}'$ of ρ' is defined analogously to ρ'):

(∗) (a) The view of $\mathcal{E}$ in $\hat{\rho}$ is the same as the view of $\mathcal{E}$ in $\hat{\rho}'$.
(b) The last configuration of $\mathcal{E}$ in $\hat{\rho}$ is the same as the last configuration of $\mathcal{E}$ in $\hat{\rho}'$.
(c) The last configuration of $\mathcal{P}$ in $\hat{\rho}$ is the same as the configuration of the emulated $\mathcal{P}$ in the last configuration of $\mathcal{Q}_\tau$ in $\hat{\rho}'$.
(d) All emulated copies of $\mathcal{F}_{\text{crypto}}$ in the last configuration of $\mathcal{Q}_\tau$ in $\hat{\rho}'$ have compatible states.
(e) The last configuration of $\mathcal{F}_{\text{crypto}}$ in $\hat{\rho}$ is the same as the merged (see above) configuration of all emulated copies of $\mathcal{F}_{\text{crypto}}$ in the last configuration of $\mathcal{Q}_\tau$ in $\hat{\rho}'$.

After we have shown this, we can conclude as follows: Since (∗) in particularly holds for $\hat{\rho} = \rho$, only $\mathcal{E}$ might produce output to decision, and $\mathcal{E}$ uses the same randomness in both runs, the overall output of ρ is the same as the one of ρ'.

Clearly, (∗) holds if $\hat{\rho}$ has length 1, i.e., where $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ has never been activated in $\hat{\rho}$. Now, let $\hat{\rho}$ be a prefix of ρ of length > 1 and assume that (∗) holds for the prefix $\tilde{\rho}$ of $\hat{\rho}$ which is by one shorter than $\hat{\rho}$. There are two cases: In the last configuration of $\tilde{\rho}$ the last IITM that has been active (and possibly produced output) is either i) $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ or ii) $\mathcal{E}$. In case i), by (∗) for $\tilde{\rho}$, it trivially follows that the output of $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ in $\tilde{\rho}$ equals the output of $\mathcal{Q}_\tau$ in $\hat{\rho}'$. Hence, (∗) holds for $\hat{\rho}$. Next, we consider case ii). If $\mathcal{E}$ produces empty output or output to tape decision, then the run stops and nothing is to show. Now, assume that $\mathcal{E}$ produces output, say m, to $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$. Note that the message m might trigger $\mathcal{P}$ and $\mathcal{F}_{\text{crypto}}$ to send multiple messages between each other before output $\mathcal{P}$ or $\mathcal{F}_{\text{crypto}}$ outputs a message m' to $\mathcal{E}$. We show that (∗) is satisfied for every such in-between message.

If m is sent to $\mathcal{P}$, then (∗) trivially holds for the configuration that is reached when $\mathcal{P}$ outputs a message (either to $\mathcal{E}$ or to $\mathcal{F}_{\text{crypto}}$) because of (∗) (c) for $\tilde{\rho}$ and $\mathcal{P}$ uses the same randomness in ρ and ρ'.

On the other hand, if m is sent to $\mathcal{F}_{\text{crypto}}$ (i.e., to the network interface of $\mathcal{F}_{\text{crypto}}$), than, by definition of $\mathcal{Q}_\tau$ ($\mathcal{Q}_\tau$ accepts m if and only if the copy of $\mathcal{F}_{\text{crypto}}$ which is obtained from merging all copies of $\mathcal{F}_{\text{crypto}}$

would accept m), one can show that $(*)$ (d) and (e) still hold when $\mathcal{F}_{\text{crypto}}$ produces output (either to $\mathcal{E}$ or to $\mathcal{P}$). Of course $(*)$ (a), (b), and (c) hold as well because the configuration of $\mathcal{E}$ and $\mathcal{P}$ did not change at all.

Now, if $\mathcal{P}$ sends a request m' to $\mathcal{F}_{\text{crypto}}$, then we have to show that $(*)$ holds after $\mathcal{F}_{\text{crypto}}$ has processed this request and either returned a response to $\mathcal{P}$ or send a message to $\mathcal{E}$. It is easy to see that this is satisfied if m' is not a destruction request (i.e., m' is not of the form $(\mathsf{Dec}, \mathit{ptr}, y)$, (PKDec, y), $(\mathsf{MacVerify}, \mathit{ptr}, x, \sigma)$, or $(\mathsf{SigVerify}, p', \mathit{pk}, x, \sigma)$). For construction requests, e.g., m' is an encryption request, it holds because $\mathcal{F}_{\text{crypto}}$ in ρ and the corresponding copy of $\mathcal{F}_{\text{crypto}}$ in ρ' use the same randomness to produce the ciphertext. Request of the form $(\mathsf{Retrieve}, \mathit{ptr})$, $(\mathsf{Enc}, \mathit{ptr}, x)$, and $(\mathsf{PKEnc}, p', \mathit{pk}, x)$ might turn unknown keys into known keys in $\mathcal{F}_{\text{crypto}}$. By (a) in Definition 2 (i.e., explicitly shared keys do not change the known/unknown status) and because there are no implicitly shared keys (Lemma 4), we can conclude that if a key, say k, in ρ changes from unknown to known in $\mathcal{F}_{\text{crypto}}$, then k only exists in ρ' in the emulated copy of $\mathcal{F}_{\text{crypto}}$ that receives m' and, hence, it remains true that the copies of $\mathcal{F}_{\text{crypto}}$ in ρ' have compatible states ($(*)$ (d)). Next, we distinguish the three remaining cases:

- *MAC verification*, i.e., m' is of the form $(\mathsf{MacVerify}, \mathit{ptr}, x, \sigma)$: Let (p, lsid, r) be the user that sent m' to $\mathcal{F}_{\text{crypto}}$. By $(*)$ (e), we have that the key, say k, ptr points to is marked unknown in ρ iff it is marked unknown in ρ'. If k is marked known, then the same happens in ρ and ρ'. So, in the following, we assume that k is marked unknown.

  First, consider the case where verification in ρ succeeds, i.e., x is recorded as MACed under k in ρ. If k is an explicitly shared key, then, by (b) in Definition 2, x has been MACed by a user in the same session as (p, lsid, r) (or both users are corrupted) and, hence, x is recorded as MACed under k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses. It follows that m' is processed equally in ρ and ρ' and, hence, $(*)$ remains satisfied. If k is not an explicitly shared key, then it is not shared at all (Lemma 4). Hence, similarly, x is recorded as MACed under k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses and, hence, $(*)$ remains satisfied.

  Second, consider the case where verification in ρ does not succeed. Then, either $\mathcal{F}_{\text{crypto}}$ prevents forgery (i.e., the MAC verification algorithm say that the MAC verifies but x is not recorded as MACed under k) or the MAC verification algorithm does not verify the MAC (i.e., returns false). If x is not recorded as MACed for x in ρ, then it is not recorded as MACed for x in any copy of $\mathcal{F}_{\text{crypto}}$ in ρ'. Furthermore, if the MAC verification algorithm does not verify the MAC in ρ, then it does not verify it in ρ' because the same algorithm is run.

  Altogether, we conclude that $(*)$ remains satisfied after $\mathcal{F}_{\text{crypto}}$ has processed the request m'.
- *Signature verification*, i.e., m' is of the form $(\mathsf{SigVerify}, p', \mathit{pk}, x, \sigma)$: This case is analog to MAC verification, see above.
- *Symmetric authenticated decryption*, i.e., m' is of the form $(\mathsf{Dec}, \mathit{ptr}, y)$ and ptr points to a key k of type $\mathsf{authenc\text{-}key}$ in $\mathcal{F}_{\text{crypto}}$: Let (p, lsid, r) be the user that sent m' to $\mathcal{F}_{\text{crypto}}$. By $(*)$ (e), we have that k is marked unknown in ρ iff it is marked unknown in ρ'. If k is marked known, then the same happens in ρ and ρ'. So, in the following, we assume that k is marked unknown, i.e., that decryption is performed ideally by $\mathcal{F}_{\text{crypto}}$.

  Since ciphertexts do not collide, there are two cases in ρ: There exists a unique x' such that (x', y) is stored in $\mathcal{F}_{\text{crypto}}$ for key k, or there does not exist an x' such that (x', y) is stored in $\mathcal{F}_{\text{crypto}}$ for key k. In the latter case, also in ρ' it holds that there does not exist an x' such that (x', y) is stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ used by (p, lsid, r). Hence, in both runs ρ and ρ', an error (signaling that decryption failed) is returned to the user (p, lsid, r). So, $(*)$ remains satisfied.

  If there exists a unique x' such that (x', y) is stored in $\mathcal{F}_{\text{crypto}}$ for key k in ρ and (p, lsid, r) accepts the response of $\mathcal{F}_{\text{crypto}}$ (which contains x') in ρ, i.e., the test algorithm `test` which is run by (p, lsid, r) to determine whether to accept or reject the response of $\mathcal{F}_{\text{crypto}}$ returns "accept", then: If k is an explicitly shared key, then, by (b) in Definition 2, y has been produced by a user in the same session as (p, lsid, r) (or both users are corrupted) and, hence, (x', y) is stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses. It follows that m' is processed equally in ρ and ρ' and, hence, $(*)$ remains satisfied. If k is not an explicitly shared key, then it is not shared at all (Lemma 4). Hence, similarly, (x', y) is stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses and, hence, $(*)$ remains satisfied.

On the other hand, if there exists a unique x' such that (x', y) is stored in $\mathcal{F}_{\text{crypto}}$ for key k in ρ but (p, lsid, r) rejects the response of $\mathcal{F}_{\text{crypto}}$ (which contains x') in ρ, i.e., the test algorithm `test` which is run by (p, lsid, r) to determine whether to accept or reject the response of $\mathcal{F}_{\text{crypto}}$ returns "reject", then: If (x', y) is stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses in ρ', then the same happens in ρ and ρ'. If (x', y) is not stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses in ρ', then in ρ' the user (p, lsid, r) receives an error message (signaling that decryption failed). By definition of multi-session protocols and because (p, lsid, r) rejected the response in ρ, the state of (p, lsid, r) in ρ after `test` rejected the response is the same as the state of (p, lsid, r) if an error message would have been received. Hence, the state of (p, lsid, r) is the same in ρ and ρ' and, so, $(*)$ remains satisfied.

– *Symmetric unauthenticated decryption*, i.e., m' is of the form $(\mathsf{Dec}, \mathit{ptr}, y)$ and ptr points to a key k of type unauthenc-key in $\mathcal{F}_{\text{crypto}}$: Let (p, lsid, r) be the user that sent m' to $\mathcal{F}_{\text{crypto}}$. This case is similar to symmetric authenticated decryption, only the following cases differ:
  - If there does not exist an x' such that (x', y) is stored in $\mathcal{F}_{\text{crypto}}$ for key k in ρ, then also in ρ' there does not exist and x' such that (x', y) is stored for k in any copy of $\mathcal{F}_{\text{crypto}}$. Hence, in both runs ρ and ρ' a message is sent to the environment (network) to ask for a decryption of y and $(*)$ remains satisfied.
  - If there exists a unique x' such that (x', y) is stored for k in $\mathcal{F}_{\text{crypto}}$ in ρ, (p, lsid, r) rejects the response of $\mathcal{F}_{\text{crypto}}$ (which contains x') in ρ, and (x', y) is not stored for k in the copy of $\mathcal{F}_{\text{crypto}}$ that (p, lsid, r) uses in ρ', then: The copy of $\mathcal{F}_{\text{crypto}}$ for user (p, lsid, r) asks the environment to provide a decryption but, by definition of $\mathcal{Q}_\tau$, $\mathcal{Q}_\tau$ directly returns an error message to this copy of $\mathcal{F}_{\text{crypto}}$. Hence, as above, we can conclude that $(*)$ remains satisfied.

– *Public-key decryption*, i.e., m' is of the form (PKDec, y): This case is analog to symmetric unauthenticated decryption, see above.

This concludes the proof of Lemma 5. □

Next, we show that $\mathcal{Q}_\tau$ realizes $\mathcal{F}$. As mentioned above, this does not require that $\mathcal{P}$ satisfies implicit disjointness.

Lemma 6. *If $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$, then $\mathcal{Q}_\tau \leq \mathcal{F}$.*

Before we prove Lemma 6, we define a simulator Sim for $\mathcal{Q}_\tau \leq \mathcal{F}$ which uses several copies of the simulator Sim_τ for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$ (i.e., $\mathcal{E} \,|\, F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathit{Sim}_\tau \,|\, F_{\text{single}} \,|\, \mathcal{F}$ and Sim_τ is a restricted simulator as defined in Section 4.3: Sim_τ exactly simulates $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ in its first stage and only (possibly) deviates from this in its second stage).

Definition 8 (The Simulator *Sim*). *The IITM Sim has the same network interface as $\mathcal{Q}_\tau$ and connects to the network interface of $\mathcal{F}$ (i.e., $\mathit{Sim} \,|\, \mathcal{F}$ has the same external interface as $\mathcal{Q}_\tau$). It simulates the IITM $\mathcal{Q}_\tau$ (see Definition 7) as follows.*

Every session-start message for a user (which is forwarded by $\mathcal{F}$ to Sim), is forwarded by Sim to $\mathcal{Q}_\tau$. Also, Sim forwards all network output/input from/to $\mathcal{Q}_\tau$ to/from the environment.

During the simulation of $\mathcal{Q}_\tau$, Sim always checks whether there exists a new (complete) session, i.e., instances $M_1, \ldots, M_r$ of $\mathcal{P}$ (which are emulated by $\mathcal{Q}_\tau$) such that they are partners (according to τ). If Sim finds such a new session, then Sim continues the simulation of $\mathcal{Q}_\tau$ such that this session (i.e., the instances $M_1, \ldots, M_r$ and the corresponding copy of $\mathcal{F}_{\text{crypto}}$) is now handled by a new copy of Sim_τ. The initial state of this new copy of Sim_τ is set as if this session has always been handled by Sim_τ. Because Sim_τ in its first stage exactly simulated $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$, Sim can adjust the state of Sim_τ appropriately. Then, Sim forwards all output/input from/to Sim_τ to/from $\mathcal{F}$.

Furthermore, Sim always checks whether an instance M_r of $\mathcal{P}$ gets corrupted (i.e., the flag **corrupted** *in the state of M_r is set to* true*). If this is the case, then Sim corrupts the corresponding local session in $\mathcal{F}$. For a corrupted instance M_r, Sim forwards all I/O output/input from/to M_r to/from $\mathcal{F}$ (which forwards it to/from the user).*

Once an uncorrupted instance M_r belongs to a complete session, it does not produce I/O output because it is handled by a copy of Sim_τ which directly interfaces with $\mathcal{F}$. But an uncorrupted instance M_r that does *not belong to a complete session might produces I/O output. If this happens, then Sim terminates. (In this case the simulation fails but this happens only with negligible probability.)*

Proof of Lemma 6. The proof of Lemma 6 is very similar to the proof of Theorem 3 because $\mathcal{Q}_\tau$ is similar to $\mathcal{P} \,|\, \mathcal{F}'$ for some multi-session local-SID functionality $\mathcal{F}'$.

Similar to Lemma 1 in the proof of Theorem 3, using $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$, we can show that in every run of $\mathcal{E} \,|\, \mathcal{Q}_\tau$ for any environment $\mathcal{E}$ it holds that $\mathcal{Q}_\tau$ never produces I/O output for a user that is uncorrupted or does not belong to a complete (i.e., one user per role) global session (according to τ) (except with negligible probability). Intuitively, this holds because the simulator Sim_τ for $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$ only (possibly) creates a session in $\mathcal{F}$ in its second stage, i.e., after τ has signaled a complete global session, and $\mathcal{F}$ guarantees that there is no I/O output of uncorrupted users that do not belong to a global session.

For the rest of the proof, we fix an environment $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{F}'$. Let $p_\mathcal{E}$ be a polynomial (in the security parameter η) that bounds the overall runtime (i.e., taken steps) of $\mathcal{E}$. (By definition of environmental systems in the IITM model such a polynomial exists.) Since only $\mathcal{E}$ can create new instances of machines in $\mathcal{P}$ by sending requests to them, the overall number of these instances is bounded by $p_\mathcal{E}$. Also, we use the following abbreviations:

$$\mathcal{R} := F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$$
$$\mathcal{I} := \mathit{Sim}_\tau \,|\, F_{\text{single}} \,|\, \mathcal{F} \ .$$

So, we have that
$$\mathcal{E}' \,|\, \mathcal{R} \equiv \mathcal{E}' \,|\, \mathcal{I} \tag{11}$$
for every environment $\mathcal{E}'$ for $\mathcal{R}$.

We then define hybrid systems H_i for all $i \in \mathbf{N}$ similar to the proof of Theorem 3. Basically, H_i emulates $\mathcal{E}$ interacting with $\mathcal{Q}_\tau$ and $\mathit{Sim} \,|\, \mathcal{F}$ such that the first $i-1$ sessions are handled by $\mathcal{Q}_\tau$ and all later session are handled by $\mathit{Sim} \,|\, \mathcal{F}$. Here, the proof here is simpler than the one for Theorem 3 because we do not need to consider global sessions in $\mathcal{F}'$ which have corrupted users. Since τ is valid, it only signals sessions for uncorrupted users. So, the *i-th session* is simply the i-th complete session that τ signals.

By construction of H_1 and definition of Sim, we directly obtain that

$$H_1 \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F} \ . \tag{12}$$

Using that $\mathcal{Q}_\tau$ never produces I/O output for a user that is uncorrupted or does not belong to a complete global session (see above), we obtain that

$$H_{p_\mathcal{E}+1} \equiv \mathcal{E} \,|\, \mathcal{Q}_\tau \ . \tag{13}$$

Similar to Theorem 3, we define hybrid systems $\hat{H}_{i,i_1,\dots,i_n}$, for all $i, i_1, \dots, i_n \in \mathbf{N}$, which are supposed to run with either $\mathcal{R}$ or $\mathcal{I}$. The hybrid $\hat{H}_{i,i_1,\dots,i_n}$ emulates $\mathcal{E}$ interacting with $\mathcal{Q}_\tau$ and $\mathit{Sim} \,|\, \mathcal{F}$ but the i-th session is handled by $\mathcal{R}$ or $\mathcal{I}$, respectively. To successfully outsource the i-th session, $\hat{H}_{i,i_1,\dots,i_n}$ has to know which users belong to this session even before this session is created. Therefore, $\hat{H}_{i,i_1,\dots,i_n}$ is additionally parametrized by $i_1, \dots, i_n$ which tells $\hat{H}_{i,i_1,\dots,i_n}$ that the i_r-th user plays role r in the i-th session. Recall from Definition 7 (f) ii), that $\mathcal{Q}_\tau$ before forwarding a message from the environment to an instance of $\mathcal{F}_{\text{crypto}}$ does the following: $\mathcal{Q}_\tau$ verifies that the is valid for the instance of $\mathcal{F}_{\text{crypto}}$ which is obtained by merging all instances of $\mathcal{F}_{\text{crypto}}$ (because this prevents $\mathcal{E}$, e.g., from providing the same unknown key when a fresh key is requested; also, this message would be rejected in $\mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$, where there is only one instance of $\mathcal{F}_{\text{crypto}}$). Now, $\hat{H}_{i,i_1,\dots,i_n}$ has to do the same verification but it does not know the state of $\mathcal{F}_{\text{crypto}}$ in the external session. But due to our slight modifications of $\mathcal{F}_{\text{crypto}}$ (compared to [30], see Appendix C.1), $\hat{H}_{i,i_1,\dots,i_n}$ knows which keys are known and which are unknown in the external session and this enables $\hat{H}_{i,i_1,\dots,i_n}$ to do this verification exactly as $\mathcal{Q}_\tau$ would do. If the i-th session exists (i.e., has been created at some point) and the

$\{\!\|x\|\!\}_{k_A}$	–	encryption of message x under the public key of party A
$\mathrm{sig}_{k_A}(x)$	–	signature of message x under the private key of party A
$\{x\}_k$	–	encryption of message x under the symmetric key k
$\mathrm{mac}_k(x)$	–	MAC of message x under the symmetric key k

Table 1. Notation used in protocol descriptions.

parameter $i_1, \ldots, i_n$ correctly predict the i-th session, then, at the end of the run, $\hat{H}_{i,i_1,\ldots,i_n}$ simply outputs what (the emulated) $\mathcal{E}$ would output as the overall output of the run (i.e., on tape decision). Otherwise, $\hat{H}_{i,i_1,\ldots,i_n}$ guarantees that the overall output is *not* 1.

As in the proof of Theorem 3, for all $i, i', i_1, \ldots, i_n \in \mathbf{N}$, we define $B_{H_{i'}}(i, i_1, \ldots, i_n)$, to be the set of runs of $H_{i'}$ where the i-th complete session exists (i.e., at some point in the run τ signals the i-th session with one user per role) and the user in role r is the i_r-th user (i.e., for this user the i_r-th *session-start* message has been sent), for every $r \leq n$.

Similar to Lemma 2 in the proof of Theorem 3, for all $i, i_1, \ldots, i_n \in \mathbf{N}$, we can show that

$$\Pr[\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I} \rightsquigarrow 1] = \Pr[H_i \rightsquigarrow 1 \cap B_{H_i}(i, i_1, \ldots, i_n)] \tag{14}$$

$$\Pr[\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{R} \rightsquigarrow 1] = \Pr[H_{i+1} \rightsquigarrow 1 \cap B_{H_{i+1}}(i, i_1, \ldots, i_n)] \ . \tag{15}$$

That is, the probability that a run of $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$ (resp., $\hat{H}_{i,i_1,\ldots,i_n} \,|\, \mathcal{I}$) outputs 1 (i.e., the overall output on the decision tape is 1) equals the probability that a run of H_i (resp., H_{i+1}) outputs 1 where the i-th session consists of the users $i_1, \ldots, i_n$.

Finally, as for Theorem 3, using (14) and (15), we conclude that $H_1 \equiv H_{p_{\mathcal{E}}+1}$. By (12) and (13), we further conclude that $\mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F} \equiv \mathcal{E} \,|\, \mathcal{Q}_\tau$, i.e., $\mathcal{Q}_\tau \leq \mathcal{F}$. This concludes the proof of Lemma 6. □

If $\mathcal{P}$ satisfies implicit disjointness w.r.t. τ and $F_{\text{single}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq^\tau F_{\text{single}} \,|\, \mathcal{F}$, then Lemma 5 and Lemma 6, by transitivity of $\equiv$, directly imply $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathit{Sim} \,|\, \mathcal{F}$, which concludes the proof of Theorem 4.

D Applications

First, in Appendix D.1, we define ideal functionalities for key usability and secure channel. In Appendix D.2, we present an example of an *insecure* protocol, namely the Needham-Schroeder Public-Key (NSPK) protocol, which is turned into a *secure* protocol when prefixing messages with pre-established SIDs as done in previous joint state theorems. We also give an example of a secure protocol that does not satisfies implicit disjointness, namely the Needham-Schroeder-Lowe (NSL) protocol. Our case studies on the real-world security protocols SSL/TLS, IPsec, IEEE 802.11i, and EAP-PSK are presented in Appendix D.3 to D.7. We show that these real-world protocols all satisfy implicit disjointness. Of course, in this work, we cannot precisely model and analyze all details and variants of these protocols. Instead, we only model the cryptographic core and sketch the main ideas. Finally, we prove—in Appendix D.8—that a generic multi-session real protocol that uses the multi-session local-SID key usability functionality realizes the multi-session local-SID secure channel functionality.

In the following, to describe the protocols, we use the notation introduced in Table 1.

D.1 Multi-Session Local-SID Ideal Functionalities

Here, we provide definitions of ideal functionalities for key usability and secure channel which are mentioned in Section 5 and used below. Similar functionalities for other tasks, such as key exchange or authentication, can be defined analogously.

Key Usability. The ideal functionality for key usability is inspired by the notion of key usability proposed in [16]. It is very similar to a standard key exchange functionality. However, parties do not obtain the actual

$F_{\text{key-use}}(n, q, L, D)$

Tapes: input: t_r^{in} for $r = 1, \ldots, n$ (I/O tapes), $t_{\text{adv}}^{\text{in}}$ (network tape)
output: t_r^{out} for $r = 1, \ldots, n$ (I/O tapes), $t_{\text{adv}}^{\text{out}}$ (network tape)

We say that "m is received from t_r" if the message m is received on tape t_r^{in}. By "send m to t_r" we denote that m is output on tape t_r^{out}. Similarly, we say that "m is received from/sent to t_{adv}" if m is received on $t_{\text{adv}}^{\text{in}}$ or m is output on $t_{\text{adv}}^{\text{out}}$, respectively.

State: The state of $F_{\text{key-use}}$ is maintained by the following variables $\mathbf{enc}, \mathbf{dec}, \mathbf{decTable}, \mathbf{state}_1, \ldots, \mathbf{state}_n \in \{0,1\}^* \cup \{\bot\}$. In the initial state, the value of every variable is $\bot$. The variable **decTable** is interpreted as a set (where $\bot$ is the empty set). The variables **enc** and **dec** are interpreted as algorithms. Given a (description of an) algorithm A, by "$y \leftarrow A(x)$" we denote that a probabilistic execution of A is simulated on input x. If the simulation terminates within at most $q(|x|)$ steps, then y is set to its result, otherwise, y is set to $\bot$. Similarly, by "$y := A(x)$" we denote (enforced) deterministic execution of A.

CheckAddress: Every message on every tape is always accepted.

Compute:

```
• I/O input: Upon receiving a message $m$ from $t_r$ do:
    1. If $m = (\mathsf{Create}, m_1, \ldots, m_n)$ for some bit strings $m_1, \ldots, m_n$ and $r = 1$, then: If $m_1 = \cdots = m_n$, then set
       $\mathbf{state}_i := 0$ for all $i \le n$, otherwise, set $\mathbf{state}_i := \mathsf{error}$ for all $i \le n$. Then, send $m$ to $t_{\text{adv}}$.
       (Note that in the induced multi-session local-SID functionality $\mathcal{F}[F_{\text{key-use}}]$, this Create message is sent by
       $\mathcal{F}$ and that the bit strings $m_1, \ldots, m_n$ are the ones from the session-start messages. These bit strings can
       be used by an implementing protocol to express the intended partners, e.g., $m_r = (p_1, \ldots, p_n)$ for all $r \le n$.
       The condition $m_1 = \cdots = m_n$ then guarantees that the users agree on the corresponding partners.)
    2. If $m = (\mathsf{Enc}, x)$ for some $x \in D(\eta)$ and $\mathbf{state}_r = \mathsf{ok}$, then do:
        (a) Compute $\overline{x} \leftarrow L(1^\eta, x)$, $y \leftarrow \mathbf{enc}(k, \overline{x})$, and $\overline{x}' := \mathbf{dec}(k, y)$.
        (b) If $\overline{x}' \neq \overline{x}$, then $y := \bot$.
        (c) If $y \neq \bot$, then add $(x, y)$ to decTable.
        (d) Send $y$ to $t_r$.
    3. If $m = (\mathsf{Dec}, y)$ for some bit string $y$ and $\mathbf{state}_r = \mathsf{ok}$, then if exists unique $x$ such that
       $(x, y) \in \mathbf{decTable}$, then send $x$ to $t_r$, otherwise, send $\bot$ to $t_r$.
• Network input: Upon receiving a message $m$ from $t_{\text{adv}}$ do:
    4. If $m = (\mathsf{Alg}, enc, dec)$ for some $enc, dec \in \{0,1\}^*$ and $\mathbf{enc} = \bot$, then set $\mathbf{enc} := enc$, $\mathbf{dec} := dec$, and send
       Ack to $t_{\text{adv}}$.
    5. If $m = (\mathsf{Establish}, r)$ for some $r \le n$, $\mathbf{enc} \neq \bot$, and $\mathbf{state}_r = 0$: set $\mathbf{state}_r := \mathsf{ok}$ and send Established to $t_r$.
• Upon I/O or network input not matching a rule above, the input is ignored (i.e., produce empty output).
```

Fig. 5. The ideal key usability functionality $F_{\text{key-use}}$ is parametrized by a number of roles $n \in \mathbf{N}$, a polynomial q, a polynomial-time computable leakage algorithm L (e.g., $L(1^\eta, m) = 0^{|m|}$), and $D = \{D(\eta)\}_{\eta \in \mathbf{N}}$ where $D(\eta)$ is a domain of plaintexts which is decidable in polynomial time (in η).

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 $\mathcal{F}_{\text{crypto}}$. Here, we only define key usability for authenticated encryption but this could be easily extended, see below.

The ideal key usability functionality $F_{\text{key-use}}$ for a single session is defined in pseudocode in Figure 5 and described below. From $F_{\text{key-use}}$, by the definition in Section 3.2, we directly obtain the multi-session local-SID functionality $\mathcal{F}_{\text{key-use}} = \mathcal{F}[F_{\text{key-use}}]$. The machine $F_{\text{key-use}}$ is parameterized by a number n of roles, a polynomial q which bounds the run-time of the encryption and decryption algorithms provided by the (ideal) adversary, a polynomial-time computable leakage algorithm L (e.g., $L(1^\eta, m) = 0^{|m|}$), and a domain $D = \{D(\eta)\}_{\eta \in \mathbf{N}}$ of plaintexts where $D(\eta)$ which is decidable in polynomial-time (in the security parameter η).

Now, we describe $F_{\text{key-use}}$. First, $F_{\text{key-use}}$ waits for a message of the form $(\mathsf{Create}, m_1, \ldots, m_n)$ on the tape for role 1. We note that in the induced multi-session local-SID functionality $\mathcal{F}[F_{\text{key-use}}]$, this Create message is sent by $\mathcal{F}$ and that the bit strings $m_1, \ldots, m_n$ are the ones from the *session-start* messages. These bit strings can be used by an implementing protocol to express the intended partners, e.g., $m_r = (p_1, \ldots, p_n)$ for all $r \le n$. If *not* $m_1 = \cdots = m_n$, the functionality basically terminates. The condition $m_1 = \cdots = m_n$

```
$F_{\text{sc}}(n)$

Tapes: Exactly like $F_{\text{key-use}}$ (see Figure 5).
State: The state of $F_{\text{sc}}$ is maintained by the following variables $\textbf{state}_i, \textbf{queue}_{i,j} \in \{0,1\}^* \cup \{\bot\}$ for all
  $i, j \in \{1, \ldots, n\}$. In the initial state, the value of every variable is $\bot$. The variables $\textbf{queue}_{i,j}$ are interpreted as
  queues, where $\bot$ is the empty queue.
CheckAddress: Every message on every tape is always accepted.
Compute:
  • I/O input: Upon receiving a message $m$ from $t_r$ do:
     1. If $m = (\mathsf{Create}, m_1, \ldots, m_n)$ for some bit strings $m_1, \ldots, m_n$ and $r = 1$, then: If $m_1 = \cdots = m_n$, then set
        $\textbf{state}_i := 0$ for all $i \leq n$, otherwise, set $\textbf{state}_i := \mathsf{error}$ for all $i \leq n$. Then, send $m$ to $t_{\text{adv}}$.
        (See explanations in Figure 5.)
     2. If $m = (\mathsf{Send}, i, x)$ for some $i \leq n$ and some bit string $x$ and $\textbf{state}_r = \mathsf{ok}$, then append $x$ to the end of
        $\textbf{queue}_{r,i}$ and send $(r, \mathsf{Send}, i, 0^{|x|})$ to $t_{\text{adv}}$.
  • Network input: Upon receiving a message $m$ from $t_{\text{adv}}$ do:
     3. If $m = (\mathsf{Establish}, r)$ for some $r \leq n$ and $\textbf{state}_r = 0$, then set $\textbf{state}_r := \mathsf{ok}$ and send $\mathsf{Established}$ to $t_r$.
     4. If $m = (\mathsf{Deliver}, i, j)$ for some $i, j \leq n$, $\textbf{state}_j = \mathsf{ok}$, and $\textbf{queue}_{i,j}$ is not empty, then remove the first
        message in $\textbf{queue}_{i,j}$, let $x$ be this message, and send $x$ to $t_j$.
     5. If $m = (\mathsf{Drop}, i, j)$ for some $i, j \leq n$ and $\textbf{queue}_{i,j}$ is not empty, then remove the first message in $\textbf{queue}_{i,j}$
        and send $\mathsf{Ack}$ to $t_{\text{adv}}$.
  • Upon I/O or network input not matching a rule above, the input is ignored (i.e., produce empty output).
```

Fig. 6. The ideal secure channel functionality F_{sc} is parameterized by a number of roles $n \in \mathbf{N}$. Its strengthened variant F^+_{sc} differs from F_{sc} only in that drop requests are ignored.

guarantees that the users agree on the corresponding partners. Then, $F_{\text{key-use}}$ expects the (ideal) adversary to provide encryption and decryption algorithms. (Similar to $\mathcal{F}_{\text{crypto}}$, we do not put any restrictions on the algorithms provided by the adversary.) Furthermore, the adversary can instruct $F_{\text{key-use}}$, for every role $r \leq n$, to tell the user in the r-th role that the session is established. Once a user has received such a *session-established* message, the user can use $F_{\text{key-use}}$ to ideally encrypt and decrypt messages (similar to $\mathcal{F}_{\text{crypto}}$).

As mentioned above, we could extend $F_{\text{key-use}}$ by features of $\mathcal{F}_{\text{crypto}}$, e.g., generation of fresh keys, key derivation, and MACs. Then, $F_{\text{key-use}}$ basically would be a copy of $\mathcal{F}_{\text{crypto}}$ which is set up with one pre-shared key, namely the session key. This session key could be used to derive other keys, to MAC/verify message, or to encrypt/decrypt messages just like $\mathcal{F}_{\text{crypto}}$. Also, derived keys and freshly generated keys could be part of plaintexts. Altogether, this provides flexibility to a higher-level protocol that uses $F_{\text{key-use}}$.

We note that $F_{\text{key-use}}$ does not define any form of corruption, so, one instance of $F_{\text{key-use}}$ represents a single *honest* session. We could modify $F_{\text{key-use}}$ to allow the (ideal) adversary to corrupt the functionality but this is often not necessary if one uses—as we do—$F_{\text{key-use}}$ as part of the multi-session local-SID functionality $\mathcal{F}_{\text{key-use}} = \mathcal{F}[F_{\text{key-use}}]$. Recall from Section 3.2 that $\mathcal{F}_{\text{key-use}}$ defines corruption of local sessions before they belong to a (global) session, i.e., the (ideal) adversary can take over complete control of a local session before it belongs to a (global) session.

Secure Channel. We define two variants of secure channel functionalities F_{sc} and F^+_{sc}. While both functionalities do not allow that messages are replayed or reordered, F_{sc} allows that messages are dropped while F^+_{sc} prevents message loss. We note that F^+_{sc} is similar to the ideal secure channel functionality in [9].

The ideal secure channel functionalities F_{sc} and F^+_{sc} for a single session are defined in pseudocode in Figure 6 and described below. From these functionalities, by the definition in Section 3.2, we directly obtain the multi-session local-SID functionalities $\mathcal{F}_{\text{sc}} = \mathcal{F}[F_{\text{sc}}]$ and $\mathcal{F}^+_{\text{sc}} = \mathcal{F}[F^+_{\text{sc}}]$. The machines F_{sc} and F^+_{sc} are parameterized by a number n of roles.

Now, we describe F_{sc} (see below for F^+_{sc}). Similarly to $F_{\text{key-use}}$, F_{sc} at first waits for a message of the form $(\mathsf{Create}, m_1, \ldots, m_n)$ on the tape for role 1. Then, the (ideal) adversary can instruct F_{sc}, for every role $r \leq n$, to tell the user in the r-th role that the session is established. Once a user has received such a *session-established* message, the user can use F_{sc} to send messages to other users via the secure channel.

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

Fig. 7. The NSPK protocol.

1. $A \to B$: N'_A
2. $B \to A$: N'_B
3. $A \to B$: $\{\!|(p_A, p_B, N'_A, N'_B), N_A, p_A|\!\}_{k_B}$
4. $B \to A$: $\{\!|(p_A, p_B, N'_A, N'_B), N_A, N_B|\!\}_{k_A}$
5. $A \to B$: $\{\!|(p_A, p_B, N'_A, N'_B), N_B|\!\}_{k_B}$

Fig. 8. The idealized NSPK protocol.

1. $A \to B$: $\{\!|N_A, p_A|\!\}_{k_B}$
2. $B \to A$: $\{\!|N_A, N_B, p_B|\!\}_{k_A}$
3. $A \to B$: $\{\!|N_B|\!\}_{k_B}$

Fig. 9. The NSL protocol.

The adversary only learns the length of the messages that are sent via the secure channel and can instruct F_{sc} to deliver or drop messages. In particular, the adversary does not learn the actual messages and cannot reorder or replay the messages.

The strengthened variant F^+_{sc} is defined just as F_{sc}, except that the adversary cannot drop messages.

As for $F_{\text{key-use}}$, we note that F_{sc} (and F^+_{sc}) does not define any form of corruption, so, one instance of F_{sc} represents a single *honest* session but corruption is defined for $\mathcal{F}_{\text{sc}}$ (and $\mathcal{F}^+_{\text{sc}}$), see above.

We further remark that one could similarly define ideal functionalities for authenticated channels similar to the ideal functionality $\mathcal{F}_{\text{AUTH}}$ in [9]. Instead of $0^{|x|}$, the actual message x could be sent to the adversary. Furthermore, if the secure channel (or authenticated channel) does not guarantee replay protection or does not guarantee that messages arrive in the correct order, one could easily weaken the functionality appropriately.

D.2 The Needham-Schroeder Public-Key Protocol With and Without Lowe's Fix

The NSPK Protocol. The Needham-Schroeder Public-Key (NSPK) protocol [32] is sketched in Figure 7. In this, p_A is A's PID and N_A and N_B are nonces chosen by A and B, respectively. It can be shown that this protocol (appropriately modeled as a protocol in the IITM or UC model) is secure in a single session setting (e.g., realizes a single-session ideal key exchange functionality). Using the joint state composition theorems for public-key encryption in [28], we obtain security in the multi-session setting of an idealized NSPK protocol which assumes and uses pre-established SIDs. A concrete instance of such an idealized NSPK protocol where these SIDs are established by exchanging nonces (N'_A and N'_B), as discussed in [2], is the protocol depicted in Figure 8. As can be seen, this changes the NSPK protocol dramatically; in particular, while NSPK is insecure in the multi-session setting, (the protocol in) Figure 8 is secure. We note that Figure 8 in particularly applies Lowe's fix [31] (namely adding B's PID p_B to the second plaintext, see Figure 9) but it does much more; also, it trivially satisfies implicit disjointness.

The NSL Protocol. The Needham-Schroeder-Lowe (NSL) protocol [31] is sketched in Figure 9. In this, p_A and p_B are A's and B's PID, respectively, and N_A and N_B are the nonces chosen by A and B, respectively. This protocol (appropriately modeled as a multi-session protocol that uses $\mathcal{F}_{\text{crypto}}$ for public-key encryption) does not satisfy implicit disjointness because B would decrypt the first message (i.e., $\{\!|N_A, p_A|\!\}_{k_B}$) in any session. The reason that this protocol is still secure is because the link between the first message and the session is established by the third protocol message (i.e., $\{\!|N_B|\!\}_{k_B}$), see also [10]. Fortunately, as we see below, most real-world protocols do not use such kind of authentication but establish a link between every message (that contains ciphertexts, MACs, or signatures) and the session.

D.3 The SSL/TLS Protocol

The Transport Layer Security (TLS) protocol [18] consists of multiple subprotocols including TLS Handshake Protocols and the TLS Record Protocol. Basically, the Handshake Protocols are used to establish a session key and this session key is used in the Record Protocol to provide a secure channel. There are three Handshake Protocols two thereof are based on Diffie-Hellman key exchange and the third is based RSA encryption. Here, we only consider the third variant. Also, we consider the variant of the Handshake Protocol where the client authenticates itself using digital signatures. (There also exists a variant where the client is not authenticated but only the server.) We note that all other variants of the Handshake Protocols could also be analyzed using our methods. We assume that the server and client can verify the public key of the other side (e.g., using some kind of public key infrastructure). The cryptographic core of the this Handshake Protocol is depicted in Figure 2. In this, p_C and p_S are the PIDs of C and S, respectively; N_C and N_S are C's and S's nonce; k_C and k_S are C's and S's public key; the premaster secret PMS is chosen randomly by C; c_i for $i \leq 8$ are distinct constants; c_1, c_2, which are part of the client and server hello messages, are used to negotiate algorithms, we model this as constants; F is a pseudo-random function; the master secret MS is derived from PMS as follows: $\mathit{MS} = F(\mathit{PMS}, c_0 \| N_C \| N_S)$; $\{m\}_{k_1,k_2}$ denotes MAC-then-encrypt, i.e., $\{m\}_{k_1,k_2} = \{m, \mathrm{mac}_{k_1}(m)\}_{k_2}$; the encryption key client to server is $\mathit{EKCS} = F(\mathit{MS}, c_5 \| N_C \| N_S)$; the encryption key server to client is $\mathit{EKSC} = F(\mathit{MS}, c_6 \| N_C \| N_S)$; the integrity key client to server is $\mathit{IKCS} = F(\mathit{MS}, c_7 \| N_C \| N_S)$; the integrity key server to client is $\mathit{IKSC} = F(\mathit{MS}, c_8 \| N_C \| N_S)$; H is a hash function; *handshake* stands for the concatenation of all previous messages, i.e., $\mathit{handshake} = c_1 \| N_C \| p_S \| k_S \| c_2 \| N_S \| p_C \| k_C \| \; \{\!|\mathit{PMS}|\!\}_{k_S}$. 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 $\mathrm{sig}_{k_C}(\mathit{handshake})$ (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 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, digital signatures, key derivation, 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. Of course, the domain of plaintexts for $\mathcal{F}_{\text{key-use}}$ must not contain plaintexts of the form in message 3 and 4 of the protocol because, otherwise, trivially $\mathcal{P}_{\text{TLS}}$ does not realize $\mathcal{F}_{\text{key-use}}$ (decryption of the ciphertexts sent in message 3 and 4 would only succeed in $\mathcal{P}_{\text{TLS}}$ but not in $\mathcal{F}_{\text{key-use}}$). We assume that all public keys are known to all parties. More precisely, if an instance (of M_C or M_S) receives the public key k_S or k_C, respectively, then the instance checks that this public key is in fact the correct public key of the desired partner by comparing the received public key with the public key obtained from $\mathcal{F}_{\text{crypto}}$ for the desired partner. If it is not the correct public key, the instance halts. By this, we model that public keys are either already distributed (in a secure way) or that there is some reliable public key infrastructure. In Step 3 of the protocol, the client computes the message $\{F(\mathit{MS}, c_3 \| H(\mathit{handshake}))\}_{\mathit{IKCS},\mathit{EKCS}}$ as follows: First, the client derives the keys MS, IKCS, etc. according to the protocol using $\mathcal{F}_{\text{crypto}}$. Then, the client derives $F(\mathit{MS}, c_3 \| \mathit{handshake})$ using $\mathcal{F}_{\text{crypto}}$, i.e., the client obtains a pointer to this derived key. The client then also asks $\mathcal{F}_{\text{crypto}}$ (using a Retrieve-request) for the actual bit string of this derived key (which will mark this key known in $\mathcal{F}_{\text{crypto}}$, but this is not a problem). Finally, the client encrypts and MACs the obtained bit string (using $\mathcal{F}_{\text{crypto}}$). The server similarly computes the message $\{F(\mathit{MS}, c_4 \| \mathit{handshake})\}_{\mathit{IKSC},\mathit{EKSC}}$ in Step 4 of the protocol.

Corruption in $\mathcal{P}_{\text{TLS}}$ is modeled as follows. When an instance of M_C or M_S gets activated for the first time (i.e., upon receiving a *session-start* message), it sends a request to the adversary asking whether it is corrupted or not. If the adversary decides to corrupt this instance, then we call this instance *directly corrupted*. The adversary can only corrupt an instance at the beginning, i.e., before the instance starts executing the protocol. If an instance is not directly corrupted, we still call the instance *indirectly corrupted* if the public/private key of the partner (i.e., k_C or k_S) is corrupted in $\mathcal{F}_{\text{crypto}}$ (note that the instance can ask $\mathcal{F}_{\text{crypto}}$ for the corruption status of these keys, so it knows if it is indirectly corrupted). An instance that is not directly or indirectly corrupted is called *uncorrupted*. An directly or indirectly corrupted instance sets its flag **corrupted** to true. An indirectly corrupted or uncorrupted instance follows the protocol normally. Indirect corruption models that this party is honest but its partner is not (or that the adversary somehow obtained the private key of the partner). An directly corrupted instance on the other hand, gives complete control to the adversary by forwarding all messages between the adversary and the environment. Note that, by this definition, directly corrupted instances never send any requests to $\mathcal{F}_{\text{crypto}}$.[8] For an uncorrupted or indirectly corrupted instance, we assume that its public/private keys are uncorrupted in $\mathcal{F}_{\text{crypto}}$ and, if this

[8] We could allow the adversary to send requests to $\mathcal{F}_{\text{crypto}}$ in the name of a directly corrupted instance but then all keys this instance creates using $\mathcal{F}_{\text{crypto}}$ should be corrupted (including the public/private key of the party this instance belongs to). Hence, all keys this instance would have pointers to would be marked known in $\mathcal{F}_{\text{crypto}}$. Since

instance plays the role of a client, that the key PMS that it generates using $\mathcal{F}_{\text{crypto}}$ is uncorrupted in $\mathcal{F}_{\text{crypto}}$. More precisely, every instance of M_C and M_S always check that this condition is satisfied (by asking $\mathcal{F}_{\text{crypto}}$ for the corruption status of these keys) and if the condition is not met, then the instance halts. We note that (directly or indirectly) corrupted instances of $\mathcal{P}_{\text{TLS}}$ will, when realizing $\mathcal{F}_{\text{key-use}}$, correspond to corrupted local sessions in $\mathcal{F}_{\text{key-use}}$, i.e., the simulator will corrupt these local sessions in $\mathcal{F}_{\text{key-use}}$ and, hence, can exactly simulate these instances. This makes sense because even for indirectly corrupted instances, we do not assume any security guarantees to hold. We further remark that an adversary can gain complete control over a party by corrupting her public/private key in $\mathcal{F}_{\text{crypto}}$ and all her instances of M_C and M_S.

We now prove that $\mathcal{P}_{\text{TLS}}$ satisfies implicit disjointness. 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 $\mathcal{E} \,|\, \mathcal{P}_{\text{TLS}} \,|\, \mathcal{F}_{\text{crypto}}$ 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.

Proof of Theorem 5. All symmetric keys (i.e., the 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. Since only uncorrupted or indirectly corrupted instances, which follow the protocol, have access to $\mathcal{F}_{\text{crypto}}$, 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 (see below). So, let us assume that some uncorrupted or indirectly corrupted instance of M_S, say $(p_S, \mathit{lsid}_S, S)$, performed an accepted and ideal decryption request (in some run of $\mathcal{E} \,|\, \mathcal{P}_{\text{TLS}} \,|\, \mathcal{F}_{\text{crypto}}$ for some environment $\mathcal{E}$). Let p_C be the PID of the desired partner of $(p_S, \mathit{lsid}_S, S)$, let N_C be the nonce $(p_S, \mathit{lsid}_S, S)$ received, let N_S be the nonce generated by $(p_S, \mathit{lsid}_S, S)$, let k_S be its public key (i.e., the public key of party p_S), let k_C be the received public key (note that, by our modeling of $\mathcal{P}_{\text{TLS}}$, it is guaranteed that k_C equals the public key of party p_C in $\mathcal{F}_{\text{crypto}}$), and ct be the ciphertext received (containing PMS) and on which $(p_S, \mathit{lsid}_S, S)$ performed the decryption request under consideration. We distinguish two cases: i) k_C is uncorrupted and ii) k_C is corrupted (in $\mathcal{F}_{\text{crypto}}$). Case i) has already been considered in the proof sketch in Section 5.2. Now, assume that k_C is corrupted, i.e., that $(p_S, \mathit{lsid}_S, S)$ is indirectly corrupted. We show that this leads to a contradiction, i.e., that case ii) never occurs.

By definition of ideal and accepted decryption request, some instance of M_S or M_C must have encrypted PMS under k_S (using $\mathcal{F}_{\text{crypto}}$) and obtained ct as the ciphertext. This instance can only be an instance of M_C, say $(p'_C, \mathit{lsid}'_C, C)$, which is uncorrupted or indirectly corrupted (because directly corrupted instance do not have access to $\mathcal{F}_{\text{crypto}}$). Let p'_S be the PID of the desired partner of $(p'_C, \mathit{lsid}'_C, C)$, let N'_S be the nonce $(p'_C, \mathit{lsid}'_C, C)$ received, let N'_C be the nonce generated by $(p'_C, \mathit{lsid}'_C, C)$, let k'_C be its public key (i.e., the public key of party p'_C), let k'_S be the received public key (note that, by our modeling of $\mathcal{P}_{\text{TLS}}$, it is guaranteed that k'_S equals the public key of party p'_C in $\mathcal{F}_{\text{crypto}}$). We note that $p'_S = p_S$ and $k'_S = k_S$ because $(p'_C, \mathit{lsid}'_C, C)$ encrypted PMS under the public key k_S. In the test for accepting the decryption of ct, $(p_S, \mathit{lsid}_S, S)$ verified a MAC over $F(\mathit{MS}, c_3 \| \mathit{handshake})$ under the derived key $\mathit{IKCS} = F(\mathit{MS}, c_7 \| N_C \| N_S)$ with $\mathit{MS} = F(\mathit{PMS}, c_0 \| N_C \| N_S)$ (using $\mathcal{F}_{\text{crypto}}$ for MAC verification and key derivation). Since $(p'_C, \mathit{lsid}'_C, C)$ is uncorrupted or indirectly corrupted, the key PMS and, hence, also the keys MS, IKCS, etc. are all marked unknown in $\mathcal{F}_{\text{crypto}}$. Since i) the MAC verifies, ii) $\mathcal{F}_{\text{crypto}}$ guarantees that keys derived with different seeds do not collide, and iii) $\mathit{handshake}$ contains k_S, k_C, and ct, we obtain that $(p'_C, \mathit{lsid}'_C, C)$ in fact has created the MAC which $(p_S, \mathit{lsid}_S, S)$ verifies and that $k_C = k'_C$. But then, by assumption, k'_C is corrupted (in $\mathcal{F}_{\text{crypto}}$)

the adversary can perform all operations under known keys herself (outside of $\mathcal{F}_{\text{crypto}}$), allowing directly corrupted instances access to $\mathcal{F}_{\text{crypto}}$ would not give more power to the adversary.

although $(p'_C, lsid'_C, C)$ is uncorrupted or indirectly corrupted. This contradiction shows that case ii) never occurs.

Finally, let us assume that some uncorrupted or indirectly corrupted instance of M_S, say $(p_S, lsid_S, S)$, performed an accepted and ideal signature verification request (in some run of $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{TLS}} \,|\, \mathcal{F}_{\mathrm{crypto}}$ for some environment $\mathcal{E}$). Let p_C, N_C, N_S, k_S and k_C be as above. Since this verification is ideal and performed with the public key k_C, k_C is uncorrupted (in $\mathcal{F}_{\mathrm{crypto}}$). Furthermore, the signature is computed over *handshake* which contains the nonces N_C and N_S and the public key k_S. From this and because the signature verifies (the verification request is accepted), we conclude that there exists an instance of M_C, say $(p'_C, lsid'_C, C)$, which has created the signature (using $\mathcal{F}_{\mathrm{crypto}}$). Furthermore, this instance of M_C agrees on the public keys (i.e., its own public key is k_C and the received public key is k_S) and nonces (i.e., its own nonce is N_C and the received nonce is N_S). Hence, $p'_C = p_C$ and $(p'_C, lsid'_C, C)$ is uncorrupted (because k_C and k_S are uncorrupted). Hence, this instance of M_C is a partner of $(p_S, lsid_S, S)$ (w.r.t. τ_{TLS}) which has performed a corresponding signature creation request. □

D.4 The SSH Protocol

The Secure Shell (SSH) protocol, consists of several subprotocols. In this section, we analyze two main parts of SSH, see below. The SSH Transport Layer Protocol [37] first runs a key exchange protocol (typically Diffie-Hellman key exchange authenticated by digital signatures) to establishes a unique SID *sid* and a session key K. Then, this session key K is then used to derive encryption and MAC keys which are used to encrypt and MAC all following messages. The Transport Layer Protocol only authenticates the server S (not the client C), therefore, the SSH Authentication Protocol [36] is run (on top of the SSH Transport Layer Protocol) to authenticates the client C, e.g., by the Public Key Authentication Method or the Password Authentication Method. Here, we only consider the Public Key Authentication Method. The cryptographic core of the authenticated Diffie-Hellman key exchange (messages 1 to 4) with following Public Key Authentication Method (messages 5 and 6) is depicted in Figure 3. In this, N_C and N_S denote C's or S's nonces; c_i for $i \leq 6$ are distinct constants; c_1, c_2, which are part of first two protocol messages, are used to negotiate algorithms, we model this as constants; p is a publicly known large safe prime; g is a publicly known generator for a subgroup of $\mathrm{GF}(p)$; q is the order of the subgroup; the numbers x and y are chosen (by C and S, respectively) uniformly at random from $\{2, \ldots, q-1\}$; g^x and g^y are computed modulo p; k_S is S's public key; k_C is C's public key; the session key $K = g^{xy} \bmod p$; the SID $sid = H(N_C, N_S, k_S, g^x, g^y, K)$ where H is a hash function; $\{m\}_{k_1,k_2}$ denotes encrypt-and-MAC, i.e., $\{m\}_{k_1,k_2} = \{m\}_{k_2}, \mathrm{mac}_{k_1}(m)$; the encryption key client to server is $EKCS = H(K \| sid \| c_3 \| sid)$; the encryption key server to client is $EKSC = H(K \| sid \| c_4 \| sid)$; the integrity key client to server is $IKCS = H(K \| sid \| c_5 \| sid)$; the encryption key server to client is $IKSC = H(K \| sid \| c_6 \| sid)$. After this protocol is completed, a secure channel is established using the keys $EKCS, EKSC, IKCS, IKSC$.

Modeling this protocol as a multi-session real protocol $\mathcal{P}_{\mathrm{SSH}} = !M_C \,|\, !M_S$ that uses $\mathcal{F}_{\mathrm{crypto}}$ for digital signatures is straightforward. All other cryptographic operations (i.e., encryption, MAC, hashing) are carried out by M_C and M_S itself because $\mathcal{F}_{\mathrm{crypto}}$ does not support Diffie-Hellman key exchange and, hence, K (and all derived keys) cannot be a key in $\mathcal{F}_{\mathrm{crypto}}$. The protocol $\mathcal{P}_{\mathrm{SSH}}$ 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. Of course, the domain of plaintexts for $\mathcal{F}_{\text{key-use}}$ must not contain plaintexts of the form in message 5 and 6 of the protocol because, otherwise, trivially $\mathcal{P}_{\mathrm{SSH}}$ does not realize $\mathcal{F}_{\text{key-use}}$ (decryption of the ciphertexts sent in message 5 and 6 would only succeed in $\mathcal{P}_{\mathrm{SSH}}$ but not in $\mathcal{F}_{\text{key-use}}$).

Corruption in $\mathcal{P}_{\mathrm{SSH}}$ is modeled as follows: When an instance of M_C or M_S gets activated for the first time (i.e., upon receiving a *session-start* message), it sends a request to the adversary asking whether it is corrupted or not. If the adversary decides to corrupt this instance, then this instance sets its flag **corrupted** to true. The adversary can only corrupt an instance at the beginning, i.e., before the instance starts executing the protocol. A corrupted instance gives complete control to the adversary by forwarding all messages between the adversary and the environment. Note that, by this definition, corrupted instances never send any requests to $\mathcal{F}_{\text{crypto}}$.[9] For an uncorrupted instance, we assume that its own and its intended partner's public/private key is uncorrupted in $\mathcal{F}_{\text{crypto}}$. More precisely, every instance of M_C and M_S always checks that this condition is satisfied (by asking $\mathcal{F}_{\text{crypto}}$ for the corruption status of the public/private keys) and if the condition is not met, then the instance halts.

We now prove that $\mathcal{P}_{\text{SSH}}$ satisfies implicit disjointness. The proof does not need to exploit security of the encryption scheme, the MAC scheme, or the Diffie-Hellman key exchange. Moreover, the proof merely requires syntactic arguments (rather than probabilistic reasoning or reduction arguments) since we can use $\mathcal{F}_{\text{crypto}}$ for the digital signatures. We only need to assume collision resistance for the hash function H, to show that the partnering function is valid (see below).

The partnering function τ_{SSH} for $\mathcal{P}_{\text{SSH}}$ we use is the obvious one: Let ρ be a run of $\mathcal{E}\,|\,\mathcal{P}_{\text{SSH}}\,|\,\mathcal{F}_{\text{crypto}}$ 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 $\tau_{\text{SSH}}(\alpha) := \bot$. Otherwise, if $r = C$ and α contains at least the first four messages of the protocol, then $\tau_{\text{SSH}}(\alpha) := \mathit{sid}$ where $\mathit{sid} = H(N_C, N_S, k_S, g^x, g^y, K)$, N_C is the client's nonce sent in step 1 (of the protocol), N_S is the server's nonce received in step 2, g^x is the value sent in step 3, k_S is the server's public key received in step 4, g^y is the value received in step 4, and $K = g^{xy}$; analogously for the case $r = S$. It is easy to see that τ_{SSH} is valid for $\mathcal{P}_{\text{SSH}}$ if H is collision resistant because the nonces are part of the hashed message.

Proof of Theorem 6. It is easy to see that $\mathcal{P}_{\text{SSH}}$ satisfies implicit disjointness w.r.t. τ_{SSH} because the signed messages from C and S contain sid. Since only uncorrupted instances have access to $\mathcal{F}_{\text{crypto}}$, the only relevant cases are when the client (or the server) perform a verification request to verify the signature of the server (or the client, respectively).

First, we consider the case of the client. So, let us assume that some uncorrupted instance of M_C, say $(p_C, \mathit{lsid}_C, C)$, performed an accepted and ideal digital signature verification request (in some run of $\mathcal{E}\,|\,\mathcal{P}_{\text{SSH}}\,|\,\mathcal{F}_{\text{crypto}}$ for some environment $\mathcal{E}$), say under the public key k_S (which belongs to party p_S). Since the request is accepted and ideal, there exists an instance of M_C or M_S which has created the signature in this request. Since the format of the signed messages in step 4 and step 5 are different and only uncorrupted instances have access to $\mathcal{F}_{\text{crypto}}$, this can only be an uncorrupted instance of M_S, say $(p_S, \mathit{lsid}_S, S)$. Furthermore, since the message over which the signature is computed is $\mathit{sid} = H(N_C, N_S, k_S, g^x, g^y, K)$, the instances $(p_S, \mathit{lsid}_S, S)$ and $(p_C, \mathit{lsid}_C, C)$ agree on the value of sid and, hence, are partners w.r.t. τ_{SSH}.

The case of the server, i.e., where an uncorrupted instance of M_S performs an accepted and ideal digital signature verification request, is analogous. □

A more detailed analysis and a proof that $\mathcal{P}_{\text{SSH}}$ realizes $\mathcal{F}_{\text{key-use}}$ is future work. We note that to prove that $\mathcal{P}_{\text{SSH}}\,|\,\mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{key-use}}$, one would need to do reduction arguments for the Diffie-Hellman key exchange, the hash function H, and the encryption and MAC scheme used for the keys $\mathit{EKCS}, \mathit{EKSC}, \mathit{IKCS}, \mathit{IKSC}$. But, by Theorem 4, it suffices to analyze a single session, i.e., to show that $F_{\text{single}}\,|\,\mathcal{P}_{\text{SSH}}\,|\,\mathcal{F}_{\text{crypto}} \leq^{\tau} F_{\text{single}}\,|\,\mathcal{F}_{\text{key-use}}$, and one can use $\mathcal{F}_{\text{crypto}}$ for digital signatures. If one would extend $\mathcal{F}_{\text{crypto}}$ to support Diffie-Hellman key exchange, as we plan in future work, then this analysis would be greatly simplified because all cryptographic operations could be provided by $\mathcal{F}_{\text{crypto}}$.

D.5 The IKEv2 Protocol (IPsec)

The Internet Key Exchange Protocol Version 2 (IKEv2) [24] is used in the IPsec protocol suite to establish a session key. It uses Diffie-Hellman key exchange, which, e.g., is authenticated by digital signatures. The cryptographic core of the IKEv2 protocol is depicted in Figure 10. In this, I is the initiator role; R is the responder role; c_i for $i \leq 4$ are distinct constants which model fields of the messages, e.g., to negotiate

[9] We could allow the adversary to send requests to $\mathcal{F}_{\text{crypto}}$ in the name of a corrupted instance but then all keys this instance creates using $\mathcal{F}_{\text{crypto}}$ should be corrupted (including the public/private key of the party this instance belongs to). Hence, all keys this instance would have pointers to would be marked known in $\mathcal{F}_{\text{crypto}}$. Since the adversary can perform all operations under known keys herself (outside of $\mathcal{F}_{\text{crypto}}$), allowing directly corrupted instances access to $\mathcal{F}_{\text{crypto}}$ would not give more power to the adversary.

1. $I \to R$: c_1, g^x, N_I
2. $R \to I$: c_2, g^y, N_R
3. $I \to R$: $\{p_I, p_R, \mathrm{sig}_{k_I}(c_1, g^x, N_I, N_R, F(SK_{pi}, p_I)), c_3\}_{SK_{ai}, SK_{ei}}$
4. $R \to I$: $\{p_R, \mathrm{sig}_{k_R}(c_2, g^y, N_R, N_I, F(SK_{pr}, p_R)), c_4\}_{SK_{ar}, SK_{er}}$

Fig. 10. The IKEv2 Protocol.

algorithms, which we model as constants; p is a publicly known large safe prime; g is a publicly known generator for a subgroup of $\mathrm{GF}(p)$; q is the order of the subgroup; the numbers x and y are chosen (by I and R, respectively) uniformly at random from $\{2, \dots, q-1\}$; g^x, g^y, and g^{xy} are computed modulo p; p_I is I's PID; p_R is R's PID; N_I and N_R denote I's and R's, respectively, nonce; k_I is I's public key; k_R is R's public key; F is a pseudo-random function; the session key seed is $SKEYSEED := F(N_I \| N_R, g^{xy})$; the key derivation key SK_d (see below), the MAC keys SK_{ai} (for direction I to R) and SK_{ar} (for direction R to I), the encryption keys SK_{ei} and SK_{er}, and the keys SK_{pi} and SK_{pr} (which are used to generate the authentication payload) are derived from $SKEYSEED$: $SK_d \| SK_{ai} \| SK_{ar} \| SK_{ei} \| SK_{er} \| SK_{pi} \| SK_{pr} := F(SKEYSEED, N_I \| N_R \| c_1 \| c_2)$; $\{m\}_{k_1,k_2}$ denotes encrypt-then-MAC, i.e., $\{m\}_{k_1,k_2} = \{m\}_{k_2}, \mathrm{mac}_{k_1}(\{m\}_{k_2})$. The session key $KEYMAT$ is derived from the key derivation key: $KEYMAT := F(SK_d, N_I \| N_R)$.

Modeling this protocol as a multi-session real protocol $\mathcal{P}_{\mathrm{IKEv2}} = !M_I \mid !M_R$ that uses $\mathcal{F}_{\mathrm{crypto}}$ for digital signatures is straightforward. All other cryptographic operations (i.e., encryption, MAC, key derivation) are carried out by M_I and M_R itself because $\mathcal{F}_{\mathrm{crypto}}$ does not support Diffie-Hellman key exchange and, hence, g^{xy} (and all derived keys) cannot be a key in $\mathcal{F}_{\mathrm{crypto}}$. The protocol $\mathcal{P}_{\mathrm{IKEv2}}$ 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_I and M_R which are encrypted under the session key. As shown in Appendix D.8 this implies that $\mathcal{P}_{\mathrm{IKEv2}}$ can be used to build a secure channel.

Corruption in $\mathcal{P}_{\mathrm{IKEv2}}$ is modeled as follows: When an instance of M_I or M_R gets activated for the first time (i.e., upon receiving a *session-start* message), it sends a request to the adversary asking whether it is corrupted or not. If the adversary decides to corrupt this instance, then this instance sets its flag **corrupted** to true. The adversary can only corrupt an instance at the beginning, i.e., before the instance starts executing the protocol. A corrupted instance gives complete control to the adversary by forwarding all messages between the adversary and the environment. Note that, by this definition, corrupted instances never send any requests to $\mathcal{F}_{\mathrm{crypto}}$.[10] For an uncorrupted instance, we assume that its own and its intended partner's public/private key is uncorrupted in $\mathcal{F}_{\mathrm{crypto}}$. More precisely, every instance of M_I and M_R always checks that this condition is satisfied (by asking $\mathcal{F}_{\mathrm{crypto}}$ for the corruption status of the public/private keys) and if the condition is not met, then the instance halts.

The proof that $\mathcal{P}_{\mathrm{IKEv2}}$ satisfies implicit disjointness, since we use $\mathcal{F}_{\mathrm{crypto}}$ for digital signatures, is completely syntactic (i.e., no reduction arguments and not even reasoning about probabilities). We note that implicit disjointness of $\mathcal{P}_{\mathrm{IKEv2}}$ does not depend on the security of the encryption scheme, the MAC scheme, the Diffie-Hellman key exchange, or the pseudo-random function.

The partnering function τ_{IKEv2} for $\mathcal{P}_{\mathrm{IKEv2}}$ we use is the obvious one: Let ρ be a run of $\mathcal{E} \mid \mathcal{P}_{\mathrm{IKEv2}} \mid \mathcal{F}_{\mathrm{crypto}}$ 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 $r \in \{I, R\}$). If (p, lsid, r) is corrupted, then $\tau_{\mathrm{IKEv2}}(\alpha) := \bot$. Otherwise, if $r = I$ and α contains at least the first two messages of the protocol, then $\tau_{\mathrm{IKEv2}}(\alpha) := (N_I, N_R)$, where N_R is the server's nonce (p, lsid, r) received and N_I is the nonce (p, lsid, r) generated; analogously for the case $r = R$. It is easy to see that τ_{IKEv2} is valid for $\mathcal{P}_{\mathrm{IKEv2}}$ because ideal nonces (i.e., nonces generated using $\mathcal{F}_{\mathrm{crypto}}$) do not collide.

Theorem 7. *$\mathcal{P}_{\mathrm{IKEv2}}$ satisfies implicit disjointness w.r.t. τ_{IKEv2}.*

[10] We could allow the adversary to send requests to $\mathcal{F}_{\mathrm{crypto}}$ in the name of a corrupted instance but then all keys this instance creates using $\mathcal{F}_{\mathrm{crypto}}$ should be corrupted (including the public/private key of the party this instance belongs to). Hence, all keys this instance would have pointers to would be marked known in $\mathcal{F}_{\mathrm{crypto}}$. Since the adversary can perform all operations under known keys herself (outside of $\mathcal{F}_{\mathrm{crypto}}$), allowing directly corrupted instances access to $\mathcal{F}_{\mathrm{crypto}}$ would not give more power to the adversary.

1. $A \to S$: p_A, N_A, c_1
2. $S \to A$: $p_S, N_S, c_2, \text{mac}_{\text{KCK}}(N_S, c_2)$
3. $A \to S$: $p_A, N_A, c_3, \text{mac}_{\text{KCK}}(N_A, c_3)$
4. $S \to A$: $p_S, c_4, \text{mac}_{\text{KCK}}(c_4)$

Fig. 11. The 4-Way Handshake Protocol of IEEE 802.11i.

Proof. It is easy to see that $\mathcal{P}_{\text{IKEv2}}$ satisfies implicit disjointness w.r.t. τ_{IKEv2} because the signed messages from I and R contain N_I, N_R and N_R, N_I, respectively. Since only uncorrupted instances have access to $\mathcal{F}_{\text{crypto}}$, the only relevant cases are when the responder (or the initiator) perform a verification request to verify the signature of the initiator (or the responder, respectively).

First, we consider the case of the responder. So, let us assume that some uncorrupted instance of M_R, say $(p_R, \mathit{lsid}_R, R)$, performed an accepted and ideal digital signature verification request (in some run of $\mathcal{E} \,|\, \mathcal{P}_{\text{IKEv2}} \,|\, \mathcal{F}_{\text{crypto}}$ for some environment $\mathcal{E}$), say under the public key k_I (which belongs to party p_I). Since the request is accepted and ideal, there exists an instance of M_I or M_R which has created the signature in this request. Since $c_1 \neq c_2$ and only uncorrupted instances have access to $\mathcal{F}_{\text{crypto}}$, this can only be an uncorrupted instance of M_I, say $(p_I, \mathit{lsid}_I, I)$. Furthermore, since the message over which the signature is computed contains the nonces (N_I, N_R), the instances $(p_R, \mathit{lsid}_R, R)$ and $(p_I, \mathit{lsid}_I, I)$ agree on the nonces and, hence, are partners w.r.t. τ_{IKEv2}.

The case of the initiator, i.e., where an uncorrupted instance of M_I performs an accepted and ideal digital signature verification request, is analogous. □

A more detailed analysis and a proof that $\mathcal{P}_{\text{IKEv2}}$ realizes $\mathcal{F}_{\text{key-use}}$ is future work. Similar to $\mathcal{P}_{\text{SSH}}$, we note that to prove that $\mathcal{P}_{\text{IKEv2}} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{key-use}}$, one would need to do reduction arguments for the Diffie-Hellman key exchange, the pseudo-random function F, and the encryption and MAC schemes. But, by Theorem 4, it suffices to analyze a single session, i.e., to show that $F_{\text{single}} \,|\, \mathcal{P}_{\text{IKEv2}} \,|\, \mathcal{F}_{\text{crypto}} \leq^{\tau} F_{\text{single}} \,|\, \mathcal{F}_{\text{key-use}}$, and one can use $\mathcal{F}_{\text{crypto}}$ for digital signatures. If one would extend $\mathcal{F}_{\text{crypto}}$ to support Diffie-Hellman key exchange, as we plan in future work, then this analysis would be greatly simplified because all cryptographic operations could be provided by $\mathcal{F}_{\text{crypto}}$.

We note that an analysis of $\mathcal{P}_{\text{IKEv2}}$ using previous joint state composition theorems [13, 28] would be somewhat imprecise because the SID N_I, N_R is not established strictly before the protocol starts but along with first two protocol messages which are also used for algorithm negotiation and Diffie-Hellman key exchange. Furthermore, one of the signed messages does not exactly contain the SID N_I, N_R (as stipulated by these joint state theorems) but contains N_R, N_I instead. We also note that the predecessor of IKEv2, namely IKE, has been analyzed in [12]. In this analysis, although in a game-based setting, pre-established SIDs have been assumed similarly as in the UC model.

D.6 The 4-Way Handshake Protocol of IEEE 802.11i

The 4-Way Handshake (4WHS) protocol, which is part of the WPA2 (IEEE standard 802.11i [22, 23]), is a key exchange protocol where an authenticator A, e.g., an access point, and a supplicant S, e.g., a laptop, use a pre-shared key PSK to establish a fresh session key TK. The cryptographic core of the protocol is depicted in Figure 11. In this, N_A and N_S denote nonces chosen by A and S, respectively, and p_A and p_S denote the PIDs of A and S, respectively. From the PSK, A and S derive a Pairwise Transient Key PTK by computing $\text{PTK} = F(\text{PSK}, \text{“Pairwise key expansion”} \,\|\, \min(p_A, p_S) \,\|\, \max(p_A, p_S) \,\|\, \min(N_A, N_S) \,\|\, \max(N_A, N_S))$ where F is a pseudo-random function. The PTK is then split into the Key Confirmation Key (KCK) and the Temporary Key (TK). The key TK is the established session key. The pairwise different constants $c_1, \ldots, c_4$ are used to indicate different messages.

In [30], it has been shown that the 4WHS protocol realizes an ideal functionality for key exchange (which is similar to our multi-session ideal functionality $\mathcal{F}_{\text{ke}}$ in Section D.1) but their analyzes directly considers the multi-session setting. Furthermore, it has been shown that the CCM Protocol (CCMP), which uses the 4WHS protocol to establish a secure channel (using the key TK for authenticated encryption), realizes an ideal functionality for secure channel (which is similar to our multi-session ideal functionality $\mathcal{F}_{\text{sc}}$ in Section D.1).

In the following, we demonstrate how we could analyze the 4WHS protocol using Theorem 4. The security of CCMP using 4WHS then follows from our results in Appendix D.8. Modeling the 4WHS protocol as a multi-session protocol $\mathcal{P}_{\text{4WHS}} = !M_A \mid !M_S$ that uses $\mathcal{F}_{\text{crypto}}$ is straightforward and has similarly been done in [30]. After the protocol is completed and TK is established, the users are provided with an interface to use the key TK for (ideal) authenticated encryption and decryption. These requests are performed using $\mathcal{F}_{\text{crypto}}$. In other words, $\mathcal{P}_{\text{4WHS}} \mid \mathcal{F}_{\text{crypto}}$ is meant to realize the key usability functionality $\mathcal{F}_{\text{key-use}}$ (see Appendix D.1). Corruption is modeled similar to $\mathcal{P}_{\text{TLS}}$: An instance is corrupted if it is directly corrupted (i.e., by a corrupt message at the beginning) or indirectly corrupted (i.e., the pre-shared key PSK of this instance is corrupted in $\mathcal{F}_{\text{crypto}}$).

The proof that $\mathcal{P}_{\text{4WHS}}$ satisfies implicit disjointness, since we use $\mathcal{F}_{\text{crypto}}$ for all cryptographic operations, is completely syntactic (i.e., no reduction arguments and not even reasoning about probabilities).

The partnering function τ_{4WHS} for $\mathcal{P}_{\text{4WHS}}$ we use is the obvious one: Let ρ be a run of $\mathcal{E} \mid \mathcal{P}_{\text{4WHS}} \mid \mathcal{F}_{\text{crypto}}$ 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 $r \in \{A, S\}$). If (p, lsid, r) is corrupted, then $\tau_{\text{4WHS}}(\alpha) := \bot$. Otherwise, if $r = A$ and α contains at least the first two messages of the protocol, then $\tau_{\text{4WHS}}(\alpha) := (N_A, N_S)$, where N_S is the nonce (p, lsid, r) received and N_A is the nonce (p, lsid, r) generated; analogously for the case $r = S$. It is easy to see that τ_{4WHS} is valid for $\mathcal{P}_{\text{4WHS}}$ because ideal nonces (i.e., nonces generated using $\mathcal{F}_{\text{crypto}}$) do not collide.

Theorem 8. *$\mathcal{P}_{\text{4WHS}}$ satisfies implicit disjointness w.r.t. τ_{4WHS}.*

Proof. Let $\mathcal{E}$ be an environment $\mathcal{E}$ of $\mathcal{P}_{\text{4WHS}} \mid \mathcal{F}_{\text{crypto}}$ and ρ be a run of $\mathcal{E} \mid \mathcal{P}_{\text{4WHS}} \mid \mathcal{F}_{\text{crypto}}$. First, we show that (a) and (b) in Definition 2 hold for PSK (which is an explicitly shared key). This key is never encrypted or retrieved and, hence, it is either corrupted (and always known) or it is unknown, i.e., (a) is satisfied. Furthermore, it is never used to perform a destruction request (which in fact would be impossible because it is a key of type pre-key which can only be used for key derivation) and, hence, (b) is satisfied as well.

The only other keys in the protocol are the derived keys KCK and TK. If they are not explicitly shared (in ρ), then we are done because implicit disjointness only talks about explicitly shared keys. Recall that KCK and TK are derived from PSK using $\min(N_A, N_S) \| \max(N_A, N_S)$ as part of the seed. If they would instead be derived using $N_A \| N_S$ (i.e., with a fixed order of the nonces), we could directly conclude that implicit disjointness is satisfied because KCK and TK would not be explicitly shared. All instances that would derive these keys would have SID (N_A, N_S) (according to τ_{4WHS}) and, hence, they all would belong to the same session. We note that this design decision has another disadvantage, in a particular (unusual) setting, it enables a reflection attack on WPA, see below. Next, we show that KCK and TK satisfy (a) and (b) in Definition 2 if they are explicitly shared.

First, we note that KCK and TK are known (in $\mathcal{F}_{\text{crypto}}$) iff PSK is known because no user ever encrypts or retrieves these keys. Because PSK is known iff it is corrupted (a) in Definition 2 is satisfied for KCK and TK. In the following, we assume that PSK, KCK, and TK are unknown. Assume that there exist two distinct users $(p_1, \mathit{lsid}_1, r_1)$ and $(p_2, \mathit{lsid}_2, r_2)$ such that both users have a pointer to KCK (and TK) and they belong to different sessions, i.e., $\tau_{\text{4WHS}(p_1, \mathit{lsid}_1, r_1)}(\rho) \neq \tau_{\text{4WHS}(p_2, \mathit{lsid}_2, r_2)}(\rho)$.

First, we consider the case that $r_1 = A$ (i.e., the first user plays the role of an authenticator). Let $(N_1, N_1') := \tau_{\text{4WHS}(p_1, \mathit{lsid}_1, r_1)}(\rho)$ (i.e., N_1 is the nonce that $(p_1, \mathit{lsid}_1, r_1)$ has generated and N_1' is the received nonces). Since $\tau_{\text{4WHS}(p_1, \mathit{lsid}_1, r_1)}(\rho) \neq \tau_{\text{4WHS}(p_2, \mathit{lsid}_2, r_2)}(\rho)$ but both users used the same seed, we have that $\tau_{\text{4WHS}(p_2, \mathit{lsid}_2, r_2)}(\rho) = (N_1', N_1)$. Since ideal nonces do not collide, N_1' is to the nonce generated by $(p_2, \mathit{lsid}_2, r_2)$. Hence, $r_2 = A$ (i.e., the second user also is an authenticator). It is easy to see that no other user has a pointer to KCK (and TK) because ideal nonces do not collide. Both $(p_1, \mathit{lsid}_1, r_1)$ and $(p_2, \mathit{lsid}_2, r_2)$ play role A, i.e., in the second protocol message, they expect to receive a MAC $\text{mac}_{\text{KCK}}(N_1', c_2)$ (resp., $\text{mac}_{\text{KCK}}(N_1, c_2)$) but KCK is never used to MAC a message, hence, the verification always fails (by definition of $\mathcal{F}_{\text{crypto}}$). Hence, (b) in Definition 2 is satisfied for KCK and neither $(p_1, \mathit{lsid}_1, r_1)$ nor $(p_2, \mathit{lsid}_2, r_2)$ complete the protocol. So, TK is never used for encryption or decryption and, hence, (b) in Definition 2 is satisfied for TK too.

1. $A \to B$: N_A, p_A
2. $B \to A$: $N_A, N_B, \mathrm{mac}_{AK}(p_B, p_A, N_A, N_B), p_B$
3. $A \to B$: $N_A, \mathrm{mac}_{AK}(p_A, N_B), \{c_1\}_{TEK}$
4. $B \to A$: $N_A, \{c_2\}_{TEK}$

Fig. 12. The EAP-PSK Protocol.

The case where $r_1 = B$ is similar. We can show that $r_2 = B$, i.e., both $(p_1, \mathit{lsid}_1, r_1)$ and $(p_2, \mathit{lsid}_2, r_2)$ play role B, and that no other instance has a pointer to KCK and TK. Now, $(p_1, \mathit{lsid}_1, r_1)$ and $(p_2, \mathit{lsid}_2, r_2)$ expect to receive a MAC $\mathrm{mac}_{\mathrm{KCK}}(N_1', c_3)$ (resp., $\mathrm{mac}_{\mathrm{KCK}}(N_1, c_3)$) but KCK has only been used to MAC messages containing $c_2 \neq c_3$. Hence, we can conclude that (b) in Definition 2 is satisfied for KCK and TK. □

In the (unusual) setting that the same party plays both the role of an authenticator and a supplicant with the same pre-shared key PSK, there exists a simple reflection attack (see, e.g., [20]). This setting is prevented in [30] by assuming disjoint sets of PIDs for authenticators and supplicants. We note that $\mathcal{P}_{\mathrm{4WHS}}$ satisfies implicit disjointness even in the setting where the reflection attack would be possible. (The proof of Theorem 8 does not require this assumption.) But proving security of $\mathcal{P}_{\mathrm{4WHS}}$ in the single-session setting, i.e., $F_{\mathrm{single}} \,|\, \mathcal{P}_{\mathrm{4WHS}} \,|\, \mathcal{F}_{\mathrm{crypto}} \leq^{\tau} F_{\mathrm{single}} \,|\, \mathcal{F}_{\text{key-use}}$, would fail without this assumption.

To prove that $\mathcal{P}_{\mathrm{4WHS}}$ realizes $\mathcal{F}_{\text{key-use}}$ in the multi-session setting, i.e., $\mathcal{P}_{\mathrm{4WHS}} \,|\, \mathcal{F}_{\mathrm{crypto}} \leq \mathcal{F}_{\text{key-use}}$, by Theorem 4, it is only left to show that $F_{\mathrm{single}} \,|\, \mathcal{P}_{\mathrm{4WHS}} \,|\, \mathcal{F}_{\mathrm{crypto}} \leq^{\tau} F_{\mathrm{single}} \,|\, \mathcal{F}_{\text{key-use}}$, which follows from results shown in [30]. Since this analysis is based on $\mathcal{F}_{\mathrm{crypto}}$ it can be done by syntactic arguments only (without reasoning about probabilities at all).

D.7 The EAP-PSK Protocol

The EAP-PSK protocol [6] is an Extensible Authentication Protocol (EAP) which provides mutual authentication and key exchange using a pre-shared key (PSK). It is inspired by AKEP2 [4]. The cryptographic core of the EAP-PSK protocol is depicted in Figure 12. In this, A is the initiator role; B is the responder role; p_A is A's PID; p_B is B's PID; N_A is A's nonce; N_B is B's nonce; c_1 and c_2 are two distinct constants; F is a pseudo-random function for key derivation; A and B have a pre-shared key PSK; from PSK the MAC key $AK := F(\mathit{PSK}, 0)$ and the key derivation key $\mathit{KDK} := F(\mathit{PSK}, 1)$ are derived; the session key $\mathit{TEK} := F(\mathit{KDK}, N_B)$ is derived from KDK using the nonce N_B as a seed. We note that the session key TEK is used in an authenticated encryption scheme for key confirmation in the last two protocol messages.

Modeling this protocol as a multi-session real protocol $\mathcal{P}_{\text{EAP-PSK}} = !M_A \,|\, !M_B$ that uses $\mathcal{F}_{\mathrm{crypto}}$ for all cryptographic operations (i.e., key derivation, MAC, symmetric encryption) is straightforward. The key PSK is a pre-shared key with name (p_A, p_B) (i.e., shared by the parties with PID p_A and p_B) of type pre-key (i.e., for key derivation). The key AK is a key of type mac-key, the key KDK is of type pre-key, and the session key TEK is of type authenc-key, i.e., for authenticated encryption. The protocol $\mathcal{P}_{\text{EAP-PSK}}$ is meant to realize $\mathcal{F}_{\text{key-use}}$, i.e., after the session key TEK is established, the parties can send encryption and decryption requests to M_A and M_B which are encrypted under the session key. Of course, the domain of plaintexts for $\mathcal{F}_{\text{key-use}}$ must not contain c_1 or c_2 because, otherwise, trivially $\mathcal{P}_{\text{EAP-PSK}}$ does not realize $\mathcal{F}_{\text{key-use}}$ (decryption of the ciphertexts that are sent in message 3 and 4 would only succeed in $\mathcal{P}_{\text{EAP-PSK}}$ but not in $\mathcal{F}_{\text{key-use}}$). Similar to $\mathcal{P}_{\mathrm{4WHS}}$, an instance is corrupted if it is directly corrupted (i.e., by a corrupt message at the beginning) or indirectly corrupted (i.e., the pre-shared key PSK is corrupted in $\mathcal{F}_{\mathrm{crypto}}$).

The proof that $\mathcal{P}_{\text{EAP-PSK}}$ satisfies implicit disjointness, since we use $\mathcal{F}_{\mathrm{crypto}}$ for all cryptographic operations, is completely syntactic (i.e., no reduction arguments and not even reasoning about probabilities).

The partnering function $\tau_{\text{EAP-PSK}}$ for $\mathcal{P}_{\text{EAP-PSK}}$ we use is the obvious one: Let ρ be a run of $\mathcal{E} \,|\, \mathcal{P}_{\text{EAP-PSK}} \,|\, \mathcal{F}_{\mathrm{crypto}}$ 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 $r \in \{A, B\}$). If (p, lsid, r) is corrupted, then $\tau_{\text{EAP-PSK}}(\alpha) := \bot$. Otherwise, if $r = A$ and α contains at least the first two messages of the protocol, then $\tau_{\text{EAP-PSK}}(\alpha) := (N_A, N_B)$, where N_B is the nonce (p, lsid, r) received and N_A is the nonce (p, lsid, r) generated; analogously for the case $r = B$. It is easy to see that $\tau_{\text{EAP-PSK}}$ is valid for $\mathcal{P}_{\text{EAP-PSK}}$ because ideal nonces (i.e., nonces generated using $\mathcal{F}_{\mathrm{crypto}}$) do not collide.

Theorem 9. *$\mathcal{P}_{\text{EAP-PSK}}$ satisfies implicit disjointness w.r.t. $\tau_{\text{EAP-PSK}}$.*

Proof. Let $\mathcal{E}$ be an environment for $\mathcal{P}_{\text{EAP-PSK}} \,|\, \mathcal{F}_{\text{crypto}}$ and let ρ be a run of $\mathcal{E} \,|\, \mathcal{P}_{\text{EAP-PSK}} \,|\, \mathcal{F}_{\text{crypto}}$. We now show that implicit disjointness is satisfied for this run, i.e., more precisely, condition (a) or (b) in Definition 2 are satisfied. Let PSK be one pre-shared key in ρ, say for parties p_A and p_B. This key (potentially) is an explicitly shared key. Since PSK is never encrypted or retrieved, it is either corrupted (i.e., always known) or always unknown in ρ. That is, (a) is satisfied for PSK. Furthermore, if PSK is known, then all (directly or indirectly) derived keys, i.e., AK, KDK, and TEK are always known in ρ. So, (a) is satisfied for all these keys. Also, for known keys (b) is trivially satisfied, i.e., in the following we assume that PSK is unknown.

Since PSK is a key derivation key, it cannot be used for destruction requests and, hence, (b) is trivially satisfied for PSK. We note that AK and KDK (potentially) are explicitly shared keys. Just as PSK, they are never encrypted or retrieved in ρ, i.e., they are always unknown in ρ (because they are derived from the unknown key PSK). So, (a) is satisfied for AK and KDK. Trivially, (b) is satisfied for KDK because it is a key derivation key. Next, we show that (b) is also satisfied for AK and that TEK (which is derived from KDK) is *not* explicitly shared. First, we show that (b) is satisfied for AK, i.e., that AK only successfully verifies MACs for messages which have been MACed in the same session (according to $\tau_{\text{EAP-PSK}}$). For the MAC in the second protocol message (i.e., $\text{mac}_{AK}(p_B, p_A, N_A, N_B)$) this is obvious because it contains the nonces N_A, N_B. Now, let (p_B, lsid, B) be an instance of M_B talking to p_A such that $\tau_{\text{EAP-PSK}(p_B,\mathit{lsid},B)}(\rho) = (N_A, N_B)$ and (p_B, lsid, B) has successfully verified the MAC in the third protocol message (i.e., $\text{mac}_{AK}(p_A, N_B)$) in ρ. Such a message is only MACed by an instance (p_A, lsid', A) of M_A (for some lsid') which previously verified a MAC of the form $\text{mac}_{AK}(p_B, p_A, N'_A, N_B)$. Furthermore, only (p_B, lsid, B) can have MACed such a message because it contains N_B and, hence, N'_A in fact is N_A. We conclude that $\tau_{\text{EAP-PSK}(p_A,\mathit{lsid}',A)}(\rho) = \tau_{\text{EAP-PSK}(p_B,\mathit{lsid},B)}(\rho) = (N_A, N_B)$, i.e., (p_B, lsid, B) and (p_A, lsid', A) are partners in ρ. Hence, (b) is satisfied for AK. Now, we show that TEK is *not* explicitly shared. Let (p_B, lsid, B) be an instance of M_B talking to p_A such that $\tau_{\text{EAP-PSK}(p_B,\mathit{lsid},B)}(\rho) = (N_A, N_B)$ and (p_B, lsid, B) has derived $TEK := F(KDK, N_B)$. Because N_B is the seed, it is easy to see that there exists no other instance of M_B which derived this TEK. Assume that there exists an instance (p_A, lsid', A) of M_A that derived $TEK := F(KDK, N_B)$. Exactly as above, it can be shown that $\tau_{\text{EAP-PSK}(p_A,\mathit{lsid}',A)}(\rho) = \tau_{\text{EAP-PSK}(p_B,\mathit{lsid},B)}(\rho) = (N_A, N_B)$ (because (p_A, lsid', A) verified the MAC in the second protocol message before it derived TEK), i.e., (p_B, lsid, B) and (p_A, lsid', A) are partners in ρ. Altogether, this shows that $\mathcal{P}_{\text{EAP-PSK}}$ satisfies implicit disjointness. □

A more detailed analysis and a proof that $\mathcal{P}_{\text{EAP-PSK}}$ realizes $\mathcal{F}_{\text{key-use}}$ is future work. We only note that it should be quite easy now to prove this because, to show that $\mathcal{P}_{\text{EAP-PSK}}$ realizes $\mathcal{F}_{\text{key-use}}$ in the multi-session setting, i.e., $\mathcal{P}_{\text{EAP-PSK}} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{key-use}}$, by Theorem 4, it suffices to prove $F_{\text{single}} \,|\, \mathcal{P}_{\text{EAP-PSK}} \,|\, \mathcal{F}_{\text{crypto}} \leq^{\tau} F_{\text{single}} \,|\, \mathcal{F}_{\text{key-use}}$. Furthermore, since all cryptographic operations are supported by $\mathcal{F}_{\text{crypto}}$, proving this should be possible by syntactic reasoning based on $\mathcal{F}_{\text{crypto}}$ (without reduction arguments or reasoning about probabilities at all).

D.8 Building Secure Channels

In this section, we consider two generic secure channel multi-session real protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}^{+}_{\text{sc}}$ that use the multi-session local-SID ideal functionality $\mathcal{F}_{\text{key-use}}$ (see Figure 5).[11] Then, we show that $\mathcal{P}_{\text{sc}}$ realizes $\mathcal{F}_{\text{sc}}$ and $\mathcal{P}^{+}_{\text{sc}}$ realizes $\mathcal{F}^{+}_{\text{sc}}$, where $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}^{+}_{\text{sc}}$ are the multi-session local-SID ideal functionalities defined in Figure 6. For many real-world secure channel protocols these secure channel protocols can be instantiated appropriately such that $\mathcal{P}_{\text{sc}}$ (or $\mathcal{P}^{+}_{\text{sc}}$) faithfully models the real-world secure channel protocol. For example, the CCM Protocol, which is part of IEEE 802.11i and uses the 4WHS protocol to establish a secure channel, can be modeled by $\mathcal{P}_{\text{sc}}$ as discussed in [30].

[11] We note that (variants of) the protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}^{+}_{\text{sc}}$ have been analyzed in [30] in a different setting. It was shown—directly in the multi-session setting (i.e., without using composition theorems)—that $\mathcal{P}_{\text{sc}}$ (resp., $\mathcal{P}^{+}_{\text{sc}}$), under certain assumptions on the underlying key exchange protocol, realizes a multi-session ideal secure channel functionality which is similar to $\mathcal{F}_{\text{sc}}$ (resp., $\mathcal{F}^{+}_{\text{sc}}$).

In a nutshell, a session of $\mathcal{P}_{\text{sc}}$ (and also $\mathcal{P}^+_{\text{sc}}$) runs a session of $\mathcal{F}_{\text{key-use}}$ to establish a session key. This session key is then used to establish secure channels between the parties of the session, one channel for each pair of parties in that session. For this purpose, before a message is encrypted under the session key, the PIDs of the sender and receiver are added to the plaintexts as well as a counter. While $\mathcal{P}_{\text{sc}}$ tolerates message loss, $\mathcal{P}^+_{\text{sc}}$ does not.

The protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}^+_{\text{sc}}$ are parametrized by what we call a *plaintext construction function* f which takes two PIDs p_1, p_2, a counter v, and a message x and outputs a plaintext $f(p_1, p_2, v, x)$. We require that $f(p_1, p_2, v, x)$ is in the domain D of plaintexts which is a parameter of $\mathcal{F}_{\text{key-use}}$, that f is computable and invertible in polynomial-time, and that f is *length regular*, i.e., $|f(x_1, \ldots, x_4)| = |f(x'_1, \ldots, x'_4)|$ for all bit strings $x_1, x'_1, \ldots, x_4, x'_4$ where $|x_1| = |x'_1|, \ldots, |x_4| = |x'_4|$.

Now, we describe $\mathcal{P}_{\text{sc}}(f) = !M_1 \mid \ldots \mid !M_n$ in more detail. Recall that the machine M_r for role $r \leq n$ has an I/O input and output tape to communicate with the parties and a network input and output tape to communicate with the adversary, modeling the network. Note that $\mathcal{P}_{\text{sc}}$ and $\mathcal{F}_{\text{sc}}$ have the same I/O interface. Similarly to $\mathcal{F}_{\text{sc}}$, $\mathcal{P}_{\text{sc}}$ waits for *session-start* messages from users. For simplicity, we require that the PIDs for the roles in the request are pairwise different. For every such request, say from user (p, lsid, r), a new instance of M_r is created which handles all requests of user (p, lsid, r). Upon receiving a *session-start* message, M_r forwards it to $\mathcal{F}_{\text{key-use}}$ and waits for receiving the message $\mathsf{Established}$ from $\mathcal{F}_{\text{key-use}}$. Then, M_r sends $\mathsf{Established}$ to the user.

Every instance of the M_r maintains counters S_j and R_j for counting messages send to/received from role j, for all $j \leq n$, $j \neq i$. All counters are initialized with 0.

After a session has been established and upon receiving a message (Send, i, x), say from user (p, lsid, r), to send a message x to the party with role j (in the same session), M_r constructs the message $x' = f(p, p', S_j, x)$ where p' is the PID of the receiver, increases S_j by one, encrypts x' using $\mathcal{F}_{\text{key-use}}$ (i.e., M_r sends (Enc, x') to $\mathcal{F}_{\text{key-use}}$), obtains a ciphertext y from $\mathcal{F}_{\text{key-use}}$, and sends (p, p', y) to the adversary (network).

Whenever an instance of M_r, say for user (p, lsid, r), receives a message from the adversary (network) of the form (p', p, y) where p' is the PID of an intended partner of (p, lsid, r) for some role, say $j \neq r$, M_r decrypts y using $\mathcal{F}_{\text{key-use}}$ (i.e., M_r sends (Dec, y) to $\mathcal{F}_{\text{key-use}}$). If the decryption succeeds and the plaintext is of the form $f(p', p, v, x)$ for some number $v \geq R_j$ and some message x, then M_r sets $R_j = v + 1$ and outputs x to the user (p, lsid, r). Otherwise, M_r silently discards the message.

An instance of M_r can directly be corrupted by the adversary in which case the adversary controls the secure channel. However, the corruption needs to occur before M_r has output the message $\mathsf{Established}$. The instance of M_r is also considered corrupted (i.e., it sets its flag **corrupted** to true) if the corresponding local session in $\mathcal{F}_{\text{key-use}}$ is corrupted. The environment may ask whether or not an instance of M_r has been corrupted as described in Section 3.1.

Unlike $\mathcal{P}_{\text{sc}}$, $\mathcal{P}^+_{\text{sc}}$ discards messages from the adversary if, when decrypted, they are not of the form $f(p', p, v, x)$ for $v = R_j$, i.e., message loss is not tolerated by $\mathcal{P}^+_{\text{sc}}$.

The next theorem shows that $\mathcal{P}_{\text{sc}}$ realizes $\mathcal{F}_{\text{sc}}$ and that $\mathcal{P}^+_{\text{sc}}$ realizes $\mathcal{F}^+_{\text{sc}}$. We note that by Theorem 3, in the proof, we only need to consider the single session case.

Theorem 10. *If the leakage algorithm used in $\mathcal{F}_{\text{key-use}}$ leaks only the length of a message (e.g., $L(1^\eta, x) = 0^{|x|}$), then $\mathcal{P}_{\text{sc}} \mid \mathcal{F}_{\text{key-use}} \leq \mathcal{F}_{\text{sc}}$ and $\mathcal{P}^+_{\text{sc}} \mid \mathcal{F}_{\text{key-use}} \leq \mathcal{F}^+_{\text{sc}}$.*

Before we prove the above theorem, we note that, by this theorem, Theorem 1, and the transitivity of $\leq$, it follows that every protocol $\mathcal{P}$ that realizes $\mathcal{F}_{\text{key-use}}$ (e.g., $\mathcal{P} = \mathcal{P}_{\text{4WHS}} \mid \mathcal{F}_{\text{crypto}}$ or $\mathcal{P} = \mathcal{P}_{\text{TLS}} \mid \mathcal{F}_{\text{crypto}}$) can be used as a lower-level protocol for $\mathcal{P}_{\text{sc}}$ (or $\mathcal{P}^+_{\text{sc}}$). More precisely, if $\mathcal{P} \leq \mathcal{F}_{\text{key-use}}$, then $\mathcal{P}_{\text{sc}} \mid \mathcal{P} \leq \mathcal{F}_{\text{sc}}$ and $\mathcal{P}^+_{\text{sc}} \mid \mathcal{P} \leq \mathcal{F}^+_{\text{sc}}$.

Proof of Theorem 10. First, we consider the case of $\mathcal{F}_{\text{sc}}$, see below for the case of $\mathcal{F}^+_{\text{sc}}$. By Theorem 3, it suffices to show that $F_{\text{single}} \mid \mathcal{P}_{\text{sc}} \mid \mathcal{F}_{\text{key-use}} \leq^* F_{\text{single}} \mid \mathcal{F}_{\text{sc}}$.

First, we define a simulator *Sim* for $\mathcal{F}_{\text{sc}}$ which operates in two stages as described in Section 3.3. The simulator is the natural simulator which simply emulates the system $\mathcal{P}_{\text{sc}} \mid \mathcal{F}_{\text{key-use}}$. More precisely, *Sim* operates as follows: In the first stage *Sim* emulates a copy of the system $\mathcal{P}_{\text{sc}} \mid \mathcal{F}_{\text{key-use}}$. Upon corruption or if $\mathcal{F}_{\text{key-use}}$ receives a *session-create* message, *Sim* enters its second stage. In the second stage, *Sim* continues the emulation of $\mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ as follows. If some instance M_r (in the emulated $\mathcal{P}_{\text{sc}}$) with PID p and LSID *lsid* is corrupted, then *Sim* sends $(\mathsf{Corrupt}, (p, \mathit{lsid}, r))$ to F_{session} in $\mathcal{F}_{\text{sc}}$ (i.e., *Sim* corrupts the corresponding user in $\mathcal{F}_{\text{sc}}$). If (the emulated) $\mathcal{F}_{\text{key-use}}$ receives a *session-create* message (note that in this case, by definition of $\mathcal{P}_{\text{sc}}$, all instances of M_r have received a *session-start* message and are uncorrupted), then *Sim* forwards this message to $\mathcal{F}_{\text{sc}}$. If M_r with PID p and LSID *lsid* outputs $\mathsf{Established}$ to its user (i.e., $\mathcal{F}_{\text{key-use}}$ previously received $(\mathsf{Establish}, r)$), then *Sim* sends $(\mathsf{Establish}, r)$ to $\mathcal{F}_{\text{sc}}$. If *Sim* receives $(r, \mathsf{Send}, i, 0^l)$ from $\mathcal{F}_{\text{sc}}$ (because the user in role r wants to send a message of length l to the user in role i), then *Sim* sends $(\mathsf{Send}, i, 0^l)$ to (the emulated) M_r, i.e., as if the user wants to send 0^l. If M_r delivers a message to its user, then *Sim* checks how many messages have been dropped and instructs $\mathcal{F}_{\text{sc}}$ to drop that many messages. More precisely, if the delivered message has count value v and R_j is the corresponding counter, then *Sim* instructs $\mathcal{F}_{\text{sc}}$ to drop $v - R_j$ messages. (Note that, by definition of $\mathcal{P}_{\text{sc}}$, $v \geq R_j$, otherwise, the message would not have been delivered.) Then, *Sim* instructs $\mathcal{F}_{\text{sc}}$ to deliver the next message.

Now, we show that $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ are indistinguishable for every environment of $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$. We only sketch the proof. It is easy to see that upon corruption requests from the environment the systems $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ do not differ, i.e., the view of the environment is the same. Hence, in the following we only need to consider the case where every instance in $\mathcal{P}_{\text{sc}}$ is uncorrupted, the session in $\mathcal{F}_{\text{key-use}}$ is uncorrupted, and the session in $\mathcal{F}_{\text{sc}}$ is uncorrupted. The systems $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ potentially only differ i) upon a message send request for some message, say x, because in $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ the message $f(p', p, v, x)$ gets encrypted while in $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ the message $f(p', p, v, 0^{|x|})$ gets encrypted or ii) upon delivery of messages to the parties because $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ outputs the decrypted received message while $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ drops some messages and outputs the next message in the queue. Next, we show that the systems in fact do not differ.

ad i) By definition of $\mathcal{F}_{\text{key-use}}$ it follows that in $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ not $f(p', p, v, x)$ but its leakage $L(1^\eta, f(p', p, v, x))$ is encrypted. Similar, in $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ not the plaintext $f(p', p, v, 0^{|x|})$ but its leakage $L(1^\eta, f(p', p, v, 0^{|x|}))$ is encrypted. Since $|f(p', p, v, x)| = |f(p', p, v, 0^{|x|})|$ (because f is length regular) and the leakage algorithm leaks only the length of a message, the distribution of the produced ciphertext is the same in both systems.

ad ii) Assume that $\mathcal{E}$ sends a network message that contains a ciphertext y. In this case, both systems $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ and $\mathit{Sim} \,|\, F_{\text{single}} \,|\, \mathcal{F}_{\text{sc}}$ decrypt y using $\mathcal{F}_{\text{key-use}}$ (or the emulated $\mathcal{F}_{\text{key-use}}$ in *Sim*). If the obtained plaintext is not of the form $f(p', p, v, x)$ for some $v \geq R_j$ and some message x then both systems discard this message. Otherwise, $F_{\text{single}} \,|\, \mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ delivers x to the user for party p while *Sim* instructs $\mathcal{F}_{\text{sc}}$ to drop $v - R_j$ messages and to deliver the next message, say x', to p. We need to show that $x = x'$. By definition of $\mathcal{F}_{\text{key-use}}$, only plaintexts are returned that have been previously encrypted using $\mathcal{F}_{\text{key-use}}$. By definition of $\mathcal{P}_{\text{sc}}$, it must have been the user for party p' who produced y when sending the v-th message to p. (Otherwise, the plaintext would not be $f(p', p, v, x)$.) Note that R_j is exactly the number of messages p has already received from p'. Since $v - R_j$ messages are dropped, $\mathcal{F}_{\text{sc}}$ will deliver the $R_j + (v - R_j) = v$-th message that party p' has sent to p. We conclude that $x = x'$.

In the case of $\mathcal{F}^+_{\text{sc}}$ we can use the same simulator *Sim* except that it emulates $\mathcal{P}^+_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$ instead of $\mathcal{P}_{\text{sc}} \,|\, \mathcal{F}_{\text{key-use}}$. Note that *Sim* will never try to instruct $\mathcal{F}^+_{\text{sc}}$ to drop messages because, by definition of $\mathcal{P}^+_{\text{sc}}$, $v = R_j$ if a message is delivered. The rest of the proof is analogously to the case of $\mathcal{F}_{\text{sc}}$. □