Ideal Key Derivation and Encryption in Simulation-based Security

Ralf Küsters and Max Tuengerthal

University of Trier, Germany,
{kuesters,tuengerthal}@uni-trier.de

Abstract. Many real-world protocols, such as SSL/TLS, SSH, IPsec, IEEE 802.11i, DNSSEC, and Kerberos, derive new keys from other keys. To be able to analyze such protocols in a composable way, in this paper we extend an ideal functionality for symmetric and public-key encryption proposed in previous work by a mechanism for key derivation. We also equip this functionality with message authentication codes (MACs) and ideal nonce generation. We show that the resulting ideal functionality can be realized based on standard cryptographic assumptions and constructions, hence, providing a solid foundation for faithful, composable cryptographic analysis of real-world security protocols.
Based on this new functionality, we identify sufficient criteria for protocols to provide universally composable key exchange and secure channels. Since these criteria are based on the new ideal functionality, checking the criteria requires merely information-theoretic or even only syntactical arguments, rather than involved reduction arguments. As a case study, we use our method to analyze two central protocols of the IEEE 802.11i standard, namely the 4-Way Handshake Protocol and the CCM Protocol, proving composable security properties. As to the best of our knowledge, this constitutes the first rigorous cryptographic analysis of these protocols.

1 Introduction

Security protocols employed in practice, such as SSL/TLS, SSH, IPsec, IEEE 802.11i, DNSSEC, and Kerberos, are very complex, and hence, hard to analyze. In order to tame this complexity a viable approach is composable security analysis based on the framework of simulation-based security, in particular universal composability/reactive simulatability [14, 39]: Higher-level components of a protocol are designed and analyzed based on lower-level idealized components, called ideal functionalities. Composition theorems then allow to replace the ideal functionalities by their realizations, altogether resulting in a system without idealized components. Typically, the higher-level components are shown to realize idealized functionalities themselves. By this, they can be used as low-level idealized components in even more complex systems.

This appealing approach has so far, however, only been rarely applied to real-world protocols (see the related work). One crucial obstacle has been the lack of suitable idealized functionalities and corresponding realizations for the most basic cryptographic primitives. While functionalities for public-key encryption and digital signatures have been proposed early on [14, 39, 3, 33], only recently a functionality, which we denote by $\mathcal{F}_{\text{enc}}$ here, for symmetric encryption [35] was proposed. This functionality allows parties to generate symmetric and public/private keys and to use these keys for ideal encryption and decryption. The encrypted messages may contain symmetric keys and parties are given the actual ciphertexts, as bit strings. To bootstrap encryption with symmetric keys, $\mathcal{F}_{\text{enc}}$ also enables parties to generate and use pre-shared keys as well as public/private key pairs.

However, by itself $\mathcal{F}_{\text{enc}}$ is still insufficient for the analysis of many real-world protocols. The main goal of our work is therefore to augment this functionality (and its realization) with further primitives employed in real-word protocols and to develop suitable proof techniques in order to be able to carry out manageable, composable, yet faithful analysis of such protocols.

Contribution of this Paper. The first main contribution of this paper is to extend $\mathcal{F}_{\text{enc}}$ by a mechanism for key derivation, which is employed in virtually every real-word security protocol, as well as by MACs, digital signatures, and nonce generation; we call the new functionality $\mathcal{F}_{\text{crypto}}$. We show that $\mathcal{F}_{\text{crypto}}$ can be realized based on standard cryptographic assumptions and constructions: IND-CCA secure or authenticated encryption, UF-CMA secure MACs and digital signatures, and pseudo-random functions for key derivation, which are common also in implementations of real-world protocols. Our proof requires a non-trivial extension of the hybrid argument in [35]. Since $\mathcal{F}_{\text{crypto}}$ is a rather low-level ideal functionality and its realization is based on standard cryptographic assumptions and constructions, it is

widely applicable (see below and [35, 34]) and allows for a precise modeling of real-word security protocols, including precise modeling of message formats on the bit level.

The second main contribution of our paper are criteria for protocols to provide universally composable key exchange and secure channels. These criteria are based on our ideal functionality $\mathcal{F}_{\text{crypto}}$, and therefore, can be checked merely using information-theoretic arguments, rather than much more involved and harder to manage reduction proofs; often even purely syntactical arguments suffice, without reasoning about probabilities. Indeed, the use of $\mathcal{F}_{\text{crypto}}$ tremendously simplifies proofs in the context of real-world security protocols, as demonstrated by our case study (see below), and in other contexts (see, e.g., [35, 34]). Without $\mathcal{F}_{\text{crypto}}$, such proofs quickly become unmanageable.

The third main contribution of this paper is a case study in which we analyze central components of the wireless networking protocol WPA2, which implements the standard IEEE 802.11i [29, 28]. More precisely, we analyze the pre-shared key mode of WPA2 (WPA2-PSK), which includes the 4-Way Handshake protocol (4WHS) for key exchange and the CCM Protocol (CCMP) for secure channels. Based on $\mathcal{F}_{\text{crypto}}$ and our criteria, we show that 4WHS realizes a universally composable key exchange functionality and that 4WHS with CCMP realizes a universally composable secure channel functionality; we note that 4WHS with TKIP (instead of CCMP) has recently been shown to be insecure [42, 38]. Since we use standard cryptographic assumptions and constructions, our modeling of WPA2-PSK, including the message formats, is quite close to the actual protocol. As to the best of our knowledge, this constitutes the first rigorous cryptographic analysis of these protocols. The framework presented in this paper would also allow us to analyze other real-world security protocols in a similar way, including several modes of Kerberos, SSL/TLS, DNSSEC, and EAP.

Structure of this Paper. In Section 2, we first recall the model for simulation-based security that we use. The functionality $\mathcal{F}_{\text{crypto}}$ and its realization are presented in Section 3. The criteria for secure key exchange and secure channel protocols are established in Section 4. Our case study is presented in Section 5. We conclude with related work in Section 6. Further details and proofs are provided in the appendix.

2 Simulation-based Security

In this section, we briefly recall the IITM model for simulation-based security (see [32] for details). In this model, security notions and composition theorems are formalized based on a relatively simple, but expressive general computational model in which IITMs (inexhaustible interactive Turing machines) and systems of IITMs are defined. While being in the spirit of Canetti's UC model [16], the IITM model has several advantages over the UC model and avoids some technical problems, as demonstrated and discussed in [32, 33, 35, 27].

2.1 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. However, in this extended abstract we omit the details of the definition of IITMs, as these details are not necessary to be able to follow the rest of the paper.

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 machines 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 $\mathcal{S} = M_1 \mid !M_2$ 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 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 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 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 is activated, in this case M_1, and so on. The master IITM is also activated if the currently active IITM does not produce output, i.e., stops in this activation without writing to any output tape. A run stops if the master IITM does not produce output (and hence, does not trigger another machine) or an IITM outputs a message on a tape named decision. Such a message is considered to be the overall output of the system.

We will consider so-called well-formed systems, which satisfy a simple syntactic condition that guarantees polynomial runtime of a system.

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 (on the decision tape) is negligible in the security parameter.

2.2 Notions of Simulation-Based Security

We need the following terminology. For a system $\mathcal{S}$, the input/output tapes of IITMs 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 IITM. We can now define strong simulatability; other equivalent security notions, such as black-box simulatability and (dummy) UC can be defined in a similar way [32].

Definition 1 ([32]). *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 ideal adversary) such that the systems $\mathcal{P}$ and $\mathcal{S} \mid \mathcal{F}$ have the same external interface and for all environmental systems $\mathcal{E}$, connecting only to the external interface of $\mathcal{P}$ (and hence, $\mathcal{S} \mid \mathcal{F}$) it holds that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$.*

2.3 Composition Theorems

We restate the composition theorems from [32]. 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. These theorems can be applied iteratively to construct more and more complex systems.

Theorem 1 ([32]). *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, $\mathcal{P}_1 \mid \mathcal{P}_2$ and $\mathcal{F}_1 \mid \mathcal{F}_2$ are well-formed, and $\mathcal{P}_i \leq \mathcal{F}_i$, for $i \in \{1, 2\}$. Then, $\mathcal{P}_1 \mid \mathcal{P}_2 \leq \mathcal{F}_1 \mid \mathcal{F}_2$.*

In the following theorem, $\underline{\mathcal{F}}$ and $\underline{\mathcal{P}}$ are 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 $\mathcal{P}$.

Theorem 2 ([32]). *Let $\mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{P} \leq \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq !\underline{\mathcal{F}}$.*

3 Our Crypto Functionality

In this section, we describe our ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ and show that it can be realized under standard cryptographic assumptions. For the ideal functionality we provide an informal description and, in Appendix B, a

formal description in pseudo code. While the informal description already contains all details (plus some explanation), the pseudo code is convenient in some proofs.

As mentioned in the introduction, $\mathcal{F}_{\text{crypto}}$ extends $\mathcal{F}_{\text{enc}}$, proposed in [35], by key derivation, MACs, digital signatures, and ideal nonce generation; also pre-shared keys can now be used just as other symmetric keys. More precisely, 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. We also note that the functionality can handle an unbounded number of commands for an unbounded number of parties with the messages, ciphertexts, MACs, etc. being arbitrary bit strings of arbitrary length. We leave it 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. Since users of $\mathcal{F}_{\text{crypto}}$ are provided with actual bit strings, $\mathcal{F}_{\text{crypto}}$ can be combined with other functionalities too, including those of interest for real-word protocols, e.g., certification of public keys (see, e.g., [15]).

3.1 The Ideal Crypto Functionality

The ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ is parameterized by what we call a *leakage algorithm L*, a probabilistic polynomial time algorithm which takes as input a security parameter η and a plaintext 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. We call a leakage algorithm *L length preserving* if it always holds true that $|L(1^\eta, x)| = |x|$ for all η and m. The functionality $\mathcal{F}_{\text{crypto}}$ is also parameterized by a number n which defines the number of roles in a protocol that uses $\mathcal{F}_{\text{crypto}}$ (e.g., $n = 3$ for protocols with initiator, responder, and key distribution server). To address the different roles, every role is associated with an I/O input and output tape.

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, 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. To describe tagging more formally and flexible, we introduce two extra types: tuple to structure messages and data for every thing else, i.e., arbitrary bit strings.

To tag messages, any tagging function tag can be used that maps a type t and bit strings $x_1, \ldots, x_n$ (where n might depend on t) to a tagged bit string $\text{tag}_t(x_1, \ldots, x_n)$ with the following properties: tag is injective, computable and invertible in polynomial-time in the length of the input, and it is *length regular*, i.e., $|\text{tag}_t(x_1, \ldots, x_n)| = |\text{tag}_t(x'_1, \ldots, x'_n)|$ for every type t and bit strings $x_1, x'_1, \ldots, x_n, x'_n$ where $|x_i| = |x'_i|$ for all $i \leq n$. Pointers are tagged with the type of the key they refer to. We say that a bit string x is *well-tagged* if $x = \text{tag}_t(y)$ for some type $t \neq$ tuple and a bit string y or (recursively defined) $x = \text{tag}_{\text{tuple}}(x_1, \ldots, x_n)$ for some $n \geq 1$ and well-tagged bit strings $x_1, \ldots, x_n$. We say that a bit string x has *type t* if it is well-tagged and $x = \text{tag}_t(x_1, \ldots, x_n)$ for some $n \geq 1$ and $x_1, \ldots, x_n$. We will only require that plaintexts to be encrypted are well-tagged; MACs, digital signature, decryption, and key derivation operations take arbitrary bit strings as input.

We note that our tagging policy, which, as mentioned, is only required for plaintexts anyway, is very liberal. We need to distinguish keys and tuples from other data, in order to be able to parse plaintexts. Everything which is not a key or a tuple is considered to be of type data. We leave it up to the protocol how to interpret these bit strings, e.g., as length fields, nonces, ciphertexts, MACs, digital signatures, non-interactive zero-knowledge proofs etc. For real-world

protocols, including those mentioned in the introduction, it is typically possible to find a tagging function such that the message formats used in these protocols are captured precisely on the bit level; see for example Section 5.

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 session ID (LSID), which is chosen and managed by the party itself. In particular, on the tape for role r, $\mathcal{F}_{\text{crypto}}$ expects requests to be prefixed by tuples of the form $(p, lsid)$, and conversely $\mathcal{F}_{\text{crypto}}$ prefixes answers with $(p, lsid)$.

The functionality $\mathcal{F}_{\text{crypto}}$ keeps track of which user has access to which 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 keys stored within the functionality, 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. Every key in $\mathcal{K}$ is of the form (t, k) for $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$ and a bit string k; t is the type of the key and k the actual key. A partial function **key** yields the key $\mathbf{key}(ptr, p, lsid, r) \in \mathcal{K}$ pointer ptr refers to for user $(p, lsid, r)$.

Before any cryptographic operation can be performed, $\mathcal{F}_{\text{crypto}}$ expects to receive (descriptions of) algorithms from the (ideal) adversary, also called simulator, say enc_{authenc}, dec_{authenc} for encryption and decryption with keys of type $\mathsf{authenc\text{-}key}$, $enc_{\text{unauthenc}}$, $dec_{\text{unauthenc}}$ for encryption and decryption with keys of type $\mathsf{unauthenc\text{-}key}$, enc_{pke}, dec_{pke} for encryption and decryption with public/private keys, *mac*, *mac-verify* for creating and verifying MACs, and *sig*, *sig-verify* for creating and verifying digital signatures. These algorithms my fail, i.e., they either return a bit string or the special error symbol $\perp$. Also, $\mathcal{F}_{\text{crypto}}$ expects to receive public/private keys (pk, sk) for encryption/decryption and signing/verification for every party from the adversary. The adversary may decide to statically corrupt a public/private key of a party at the moment she provides it to $\mathcal{F}_{\text{crypto}}$. In this case $\mathcal{F}_{\text{crypto}}$ records the public/private key of this party as corrupted. We do not put any restrictions on these algorithms; all security guarantees that $\mathcal{F}_{\text{crypto}}$ provides are made explicit within $\mathcal{F}_{\text{crypto}}$ in a rather syntactic way, without relying on specific properties of these algorithms. As a result, when using $\mathcal{F}_{\text{crypto}}$ in the analysis of more complex systems, one can abstract from these algorithms entirely. When executing the algorithms, $\mathcal{F}_{\text{crypto}}$ has to do this in polynomial-time, hence, $\mathcal{F}_{\text{crypto}}$ eventually has to abort the execution of the algorithms after a given time. Therefore, $\mathcal{F}_{\text{crypto}}$ additionally is parameterized by a polynomial q and $\mathcal{F}_{\text{crypto}}$ simulates the algorithms for at most $q(l)$ steps where l is the length of the input. If the algorithm would take more than $q(l)$ steps, then the output of the computation is considered to be the error symbol $\perp$. The random coins that might be used by the algorithms are chosen by $\mathcal{F}_{\text{crypto}}$. The algorithms for decryption and verification of MACs and signatures are simulated in a deterministic way. Even if they use random coins, $\mathcal{F}_{\text{crypto}}$ uses the zero bit string.

We now describe the operations that $\mathcal{F}_{\text{crypto}}$ provides in more detail (see Appendix B for a formal specification of $\mathcal{F}_{\text{crypto}}$ in pseudo code).

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 k. The adversary can decide to corrupt k right away, in which case (t, k) is added to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$), and otherwise (t, k) is added to $\mathcal{K}_{\text{unknown}}$ (and $\mathcal{K}$). However, before adding (t, 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 (t, k) is created for user $(p, lsid, r)$, i.e., $\mathbf{key}(ptr, p, lsid, r) := (t, k)$, and ptr is returned to $(p, lsid, r)$. The value of the pointer, i.e., ptr, does not need to be secret. In fact, new pointers are created by increasing a counter. There is a different counter for every user, i.e., a user cannot tell how many keys have been created by other users from observing his pointers. If the adversary decided to corrupt k, then the pointer ptr is recorded as corrupted for user $(p, lsid, r)$. We emphasize that the difference between $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$ is not whether or not the adversary knows the value of a key (she provides these values anyway). The point is that if $(t, k) \in \mathcal{K}_{\text{unknown}}$, cryptographic operations with it are performed ideally, e.g., the leakage of a message is encrypted under the key k instead of the message itself. Conversely, if $(t, k) \in \mathcal{K}_{\text{known}}$, the cryptographic operation is performed as in the real world, e.g., the actual message is encrypted under k. So, no security guarantees are provided in this case. In the realization of $\mathcal{F}_{\text{crypto}}$, however, keys corresponding to keys in $\mathcal{K}_{\text{unknown}}$ will of course not be known to the adversary.

Public key requests $[(\mathsf{GetPubKeyPKE}, p')$ or $(\mathsf{GetPubKeySig}, p')]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to get the public key for party p' for encryption (resp., verification). If $\mathcal{F}_{\text{crypto}}$ has recorded this public key (because the adversary previously provided it, see above), then it is returned to the user. Otherwise, an error is returned to the user. We note that if users request the public key of another party, then this assumes that public keys are distributed somehow, e.g., by some kind of public key infrastructure.

Establishing pre-shared keys [(GetPSK, t, *name*)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to obtain the pre-shared key of type $t \in \{\text{authenc-key}, \text{unauthenc-key}, \text{mac-key}, \text{pre-key}\}$ with name (or identifier) *name*. Then, $\mathcal{F}_{\text{crypto}}$ forwards this request to the adversary who is supposed to provide such a key, say k. Similarly to generating fresh keys, the adversary can decide to corrupt k. However, $\mathcal{F}_{\text{crypto}}$ only accepts k under the following conditions: i) if a key (t, k') is recorded for $(t, name)$, then $k = k'$ and k is not corrupted (i.e., the adversary did not decide to corrupt k), ii) if $(t, name)$ is (recorded as) corrupted, then k is corrupted, iii) if k is corrupted, then $(t, k) \notin \mathcal{K}_{\text{unknown}}$, and iv) if no key (t, k') is recorded for $(t, name)$ and k is not corrupted, then $(t, k) \notin \mathcal{K}$. If $\mathcal{F}_{\text{crypto}}$ accepts k, then it creates a new pointer *ptr* to (t, k) for user $(p, lsid, r)$ and returns *ptr* to the user. Furthermore, if k is corrupted, then $\mathcal{F}_{\text{crypto}}$ adds (t, k) to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$), records $(t, name)$ as corrupted, and records *ptr* as corrupted for user $(p, lsid, r)$. If k is not corrupted, then $\mathcal{F}_{\text{crypto}}$ adds (t, k) to $\mathcal{K}_{\text{unknown}}$ (and $\mathcal{K}$) and records (t, k) for $(t, name)$.

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 $(\text{GetPSK}, t, (p, p'))$ 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 because of condition i). Another example is WPA2 (see Section 5), where requests of suppliers (e.g., laptops) and authenticators (e.g., access points) are of the form (GetPSK, t, *kid*), with *kid* being a key ID (instances of) suppliers and authenticators obtain from the environment (e.g. a system administrator) upon initialization. For corrupted pre-shared keys, i.e., where $(t, name)$ is corrupted, the adversary can choose a different key for every user; condition ii) and iii) guarantee that these keys are corrupted (more precisely, the pointers to corrupted pre-shared keys are corrupted) and do belong to $\mathcal{K}_{\text{unknown}}$. Condition iv) guarantees that new pre-shared keys do not collide with any other key.

Key derivation [(Derive, *ptr*, t', s)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to derive a new key of type $t' \in \{\text{authenc-key}, \text{unauthenc-key}, \text{mac-key}, \text{pre-key}\}$ (using a seed s) from a key $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ pointer *ptr* points to for user $(p, lsid, r)$. The seed s is an arbitrary bit string. It is required that the key from which the new key is derived is of type pre-key, i.e., $t = \text{pre-key}$. Then, $\mathcal{F}_{\text{crypto}}$ forwards this request to the adversary who is supposed to provide such a key, say k'. However, $\mathcal{F}_{\text{crypto}}$ only accepts k' under the following conditions: i) if a key (t', k'') is recorded as being derived from (t, k) with seed s, then $k'' = k'$, ii) if no key (t', k'') has been recorded as derived from (t, k) with seed s and $(t, k) \in \mathcal{K}_{\text{unknown}}$, then $(t', k') \notin \mathcal{K}$, and iii) if no key (t', k'') has been recorded as derived from (t, k) with seed s and $(t, k) \in \mathcal{K}_{\text{known}}$, then $(t', k') \notin \mathcal{K}_{\text{unknown}}$. If $\mathcal{F}_{\text{crypto}}$ accepts k', then it creates a new pointer *ptr* to (t', k') for user $(p, lsid, r)$ and returns *ptr* to the user. Furthermore, if $(t, k) \in \mathcal{K}_{\text{unknown}}$, then $\mathcal{F}_{\text{crypto}}$ adds (t', k') to $\mathcal{K}_{\text{unknown}}$ (and $\mathcal{K}$) and records (t', k') as derived from (t, k) with seed s. If $(t, k) \in \mathcal{K}_{\text{known}}$, then $\mathcal{F}_{\text{crypto}}$ adds (t', k') to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$).

Condition i) guarantees that key derivation is deterministic, i.e., key derivations from the same key with the same seed yield the same key. Similar to generating fresh keys, condition ii) guarantees that new keys derived from unknown keys do not collide with any other key and condition iii) guarantees that new keys derived from known keys at least do not collide with $\mathcal{K}_{\text{unknown}}$ keys. Note that we do not put any restrictions on how the adversary chooses derived keys. All security guarantees that $\mathcal{F}_{\text{crypto}}$ provides do not rely on this. In $\mathcal{F}_{\text{crypto}}$, a key derived from a key marked unknown is treated just like a freshly generated key which is marked unknown. For example, if the derived key is an encryption key, then it is used for ideal encryption, i.e., not the actual message is encrypted but its leakage, see below.

Store [(Store, t, k)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to store some bit string k with some type $t \in \{\text{authenc-key}, \text{unauthenc-key}, \text{mac-key}, \text{pre-key}\}$ as a key. If (t, k) belongs to $\mathcal{K}_{\text{unknown}}$, $\mathcal{F}_{\text{crypto}}$ will return an error message to the user, modeling that unknown keys cannot be guessed. Otherwise, $\mathcal{F}_{\text{crypto}}$ creates a new pointer to (t, k) which is given to the user. The pair (t, k) is added to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$).

Retrieve [(Retrieve, *ptr*)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to retrieve a key $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ pointer *ptr* points to for user $(p, lsid, r)$. Then, $\mathcal{F}_{\text{crypto}}$ returns k to the user and adds (t, k) to $\mathcal{K}_{\text{known}}$.

Equality test [(Equal?, *ptr*, *ptr*′)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ whether two of its pointers *ptr*, *ptr*′ refer to the same key.

Encryption under symmetric keys [(Enc, *ptr*, x)]**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to encrypt a well-tagged message x under a key $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ pointer *ptr* points to for user $(p, lsid, r)$ where $t \in \{\text{authenc-key}, \text{unauthenc-key}\}$. For every pointer $\text{tag}_{t'}(ptr)$ in x (for some $t' \in \{\text{authenc-key}, \text{unauthenc-key}, \text{mac-key}, \text{pre-key}\}$), $\mathcal{F}_{\text{crypto}}$ checks whether *ptr* is a pointer of this user to a key of type t', i.e., whether $\mathbf{key}(ptr, p, lsid, r)$ is defined and yields (t', k') for some k'. If this check fails, $\mathcal{F}_{\text{crypto}}$ returns an error message to the user. If the check succeeds, $\text{tag}_{t'}(ptr)$

in x is replaced by $\text{tag}_{t'}(k')$. This is done for every pointer of the form $\text{tag}_{t'}(ptr)$ in x, resulting in a message x'. We distinguish two cases:

i) If $(t,k) \in \mathcal{K}_{\text{unknown}}$, the *leakage* $\overline{x} = L(1^\eta, x')$ of x' is encrypted under k using either enc_{authenc} or $enc_{\text{unauthenc}}$ (the encryption algorithms provided by the adversary) depending on t. Let y denote the resulting ciphertext. Then, $\mathcal{F}_{\text{crypto}}$ checks if the decryption of y under k using either dec_{authenc} or $dec_{\text{unauthenc}}$ (the decryption algorithms provided by the adversary), depending on t, yields the leakage $\overline{x}$. If this check fails, $\mathcal{F}_{\text{crypto}}$ returns an error message to p.[1] Otherwise, the pair (x', y) is stored for the key (t,k) (for later decryption) and y is given to the user.

ii) If $(t,k) \in \mathcal{K}_{\text{known}}$, all keys in x' are added to $\mathcal{K}_{\text{known}}$, as they are encrypted under a known key. Then, x' is encrypted under k using either enc_{authenc} or $enc_{\text{unauthenc}}$ depending on t. The resulting ciphertext is given to the user.

Decryption under symmetric keys $[(\mathsf{Dec}, ptr, y)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to decrypt a ciphertext y (an arbitrary bit string) under a key $(t,k) = \textbf{key}(ptr, p, lsid, r)$ pointer ptr points to for user $(p, lsid, r)$ where $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}\}$. We distinguish two cases:

i) If $(t,k) \in \mathcal{K}_{\text{unknown}}$, $\mathcal{F}_{\text{crypto}}$ checks whether there exists exactly one x' such that (x', y) is stored for (t,k) (see above). If so, $\mathcal{F}_{\text{crypto}}$ creates new pointers to every key in x' and replaces the keys by the corresponding pointers. The resulting message x is given to the user. If there is more than one x' with (x', y) stored for (t,k), an error is returned to the user, since unique decryption is not possible. If there is no such x', the following is done: If $t = \mathsf{authenc\text{-}key}$, an error is returned, since for authenticated encryption it should not be possible to generate valid ciphertexts outside of the functionality. If $t = \mathsf{unauthenc\text{-}key}$, y is decrypted under k using $dec_{\text{unauthenc}}$ (the decryption algorithm provided by the adversary) and the following is done $(*)$: If the resulting plaintext x' is not well-tagged, an error is returned. Furthermore, if x' contains a key that belongs to $\mathcal{K}_{\text{unknown}}$, an error is returned, modeling that these keys cannot be guessed. Otherwise, $\mathcal{F}_{\text{crypto}}$ adds all keys in x' to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$), creates new pointers to every key in x', and replaces the keys by the corresponding pointers. The resulting message x is given to the user.

ii) If $(t,k) \in \mathcal{K}_{\text{known}}$, $\mathcal{F}_{\text{crypto}}$ decrypts y under k with dec_{authenc} or $dec_{\text{unauthenc}}$, depending on t, and proceeds as in $(*)$ above.

Encryption under public keys $[(\mathsf{PKEnc}, p', pk, x)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to encrypt a well-tagged message x under the public key pk of party p'. Such an encryption request is handled similarly as symmetric encryption requests. First, pointers in x are turned into keys, obtaining x'. Then, if pk is the recorded public encryption key of party p' and it is not corrupted, the encryption is performed ideally, i.e., the leakage $L(1^\eta, x')$ of x' is encrypted under pk using enc_{pke} (the public-key encryption algorithm provided by the adversary) and the pair (x', y) (where y is the ciphertext) is recorded for party p' (for later decryption). If pk is not the recorded public encryption key of party p' or the public encryption key of party p' is corrupted, then all keys in x' are marked known and x' is encrypted under pk using enc_{pke}. The resulting ciphertext is returned to the user.

We note that $\mathcal{F}_{\text{enc}}$ [35] uses an ideal functionality $\mathcal{F}_{\text{pke}}$ for public-key encryption (and decryption) and forwards public-key requests to (instances of) $\mathcal{F}_{\text{pke}}$. While we could have defined $\mathcal{F}_{\text{crypto}}$ such that it is based on $\mathcal{F}_{\text{pke}}$ (nothing would change considering reasoning based on $\mathcal{F}_{\text{crypto}}$), we decided to directly describe public-key encryption in $\mathcal{F}_{\text{crypto}}$, to have everything in one place.

Decryption under private keys $[(\mathsf{PKDec}, y)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to decrypt a ciphertext y (an arbitrary bit string) under its private key (i.e., the private key of party p). Such a decryption request is handled similarly as decryption requests under symmetric keys of type $\mathsf{unauthenc\text{-}key}$. If the private decryption key of p is corrupted or there is no recorded pair (x', y) for party p (for any x'), then y is decrypted under the recorded private decryption key sk of party p using the public-key decryption algorithm dec_{pke} (provided by the adversary). $\mathcal{F}_{\text{crypto}}$ returns an error message to the user if the resulting plaintext, say x', is not well-tagged or there exists a key in x' that belongs to $\mathcal{K}_{\text{unknown}}$, modeling that these keys cannot be guessed. All keys in x' are added to $\mathcal{K}_{\text{known}}$ (and $\mathcal{K}$).

[1] Note that every reasonable encryption scheme satisfies that the decryption of the encryption yields the plaintext again. However, as we do not put any restrictions on the algorithm provided by the adversary, $\mathcal{F}_{\text{crypto}}$ does not know whether they have this property. Because of the test that the decryption of the ciphertext yields the leakage $\overline{x}$, it is guaranteed that the ciphertext contains not only at most the information of $\overline{x}$ but exactly the information of $\overline{x}$, hence, if the leakage algorithm L has high entropy, i.e., collisions between leakages do not occur (except with negligible probability), then ciphertexts do not collide and the adversary cannot guess ciphertexts not known to her (except with negligible probability). This is sometimes useful when reasoning about protocols that use nested encryption, see [35, 34].

Otherwise, i.e., if the private decryption key of p is not corrupted and there exists an x' such that (x', y) is recorded for party p (see above), $\mathcal{F}_{\text{crypto}}$ checks whether there exists exactly one such x'. If this check fails, an error is returned to the user, since unique decryption is not possible.

Finally, $\mathcal{F}_{\text{crypto}}$ creates new pointers to every key in x' and replaces the keys by the corresponding pointers. The resulting plaintext x is given to the user.

MAC $[(\mathsf{Mac}, ptr, x)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to MAC an *arbitrary (uninterpreted) bit string* x under a key $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ pointer ptr points to for user $(p, lsid, r)$ where $t = \mathsf{mac\text{-}key}$. Then, $\mathcal{F}_{\text{crypto}}$ computes the MAC of x under k using *mac* (the MAC algorithm provided by the adversary). Let σ be the resulting MAC. If σ is not a valid MAC for x under k, i.e., it does not verify using *mac-verify* (the MAC verification algorithm provided by the adversary), then $\mathcal{F}_{\text{crypto}}$ returns an error message to the user. Otherwise, it gives σ to the user. If $(t, k) \in \mathcal{K}_{\text{unknown}}$, $\mathcal{F}_{\text{crypto}}$ records x for the key (t, k) (for later verification); we allow an adversary to derive a new MAC from a given one on the same message, which is why $\mathcal{F}_{\text{crypto}}$ does not record σ along with x.

Note that, since we leave the message x uninterpreted, one cannot use $\mathcal{F}_{\text{crypto}}$ to compute the MAC over a message that contains unknown keys. This restriction is motivated by the fact that a MAC itself does not provide confidentiality. We could abandon this restriction by extending our functionality further such that MACs are not directly returned but only pointers to it. But this would make the functionality more complicated while many real-world protocols, including SSL/TLS, SSH, IPsec, IEEE 802.11i, and Kerberos do not require this. We note that MACs that take unknown keys could also be approximated by modeling them as authenticated encryption.

Verify MAC $[(\mathsf{MacVerify}, ptr, x, \sigma)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to verify a MAC σ for some message x under a key $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ pointer ptr points to for user $(p, lsid, r)$ where $t = \mathsf{mac\text{-}key}$. Then, $\mathcal{F}_{\text{crypto}}$ verifies the MAC using *mac-verify* (the MAC verification algorithm provided by the adversary). If the MAC verifies but $(t, k) \in \mathcal{K}_{\text{unknown}}$ and x has not been recorded for (t, k) (see above), $\mathcal{F}_{\text{crypto}}$ returns an error message to the user, preventing forgery. Otherwise, $\mathcal{F}_{\text{crypto}}$ returns the result of the verification to the user.

Sign $[(\mathsf{Sign}, x)]$**.** Similar to MACing, a user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to sign an arbitrary (uninterpreted) bit string x under its private signing key. Then, $\mathcal{F}_{\text{crypto}}$ computes the signature of x under the recorded private signing key sk of party p using *sig* (the signing algorithm provided by the adversary). Let σ be the resulting signature. If σ is not a valid signature for x under the recorded public verification key pk of party p, i.e., it does not verify using *sig-verify* (the signature verification algorithm provided by the adversary), then $\mathcal{F}_{\text{crypto}}$ returns an error message to the user. Otherwise, it gives σ to the user. If the private signing key of party p is not corrupted, then $\mathcal{F}_{\text{crypto}}$ records x for p (for later verification); we allow an adversary to derive a new signature from a given one on the same message, which is why $\mathcal{F}_{\text{crypto}}$ does not record σ along with x.

Note that, since we leave the message x uninterpreted, one cannot use $\mathcal{F}_{\text{crypto}}$ to compute the signature over a message that contains unknown keys. We could extend $\mathcal{F}_{\text{crypto}}$ to allow this, see the remarks for MACs.

Verify signatures $[(\mathsf{SigVerify}, p', pk, x, \sigma)]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to verify a signature σ for some message x under the public key pk for party p'. Then, $\mathcal{F}_{\text{crypto}}$ verifies the signature using pk and *sig-verify* (the signature verification algorithm provided by the adversary). If the signature verifies but pk is the recorded public verification key of party p', pk is uncorrupted, and x has not been recorded for p' (see above), $\mathcal{F}_{\text{crypto}}$ returns an error message to the user, preventing forgery. Otherwise, $\mathcal{F}_{\text{crypto}}$ returns the result of the verification to the user.

Generating fresh nonces $[(\mathsf{NewNonce})]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to generate a fresh nonce. The request is forwarded to the adversary who is supposed to provide a nonce, say x. However, $\mathcal{F}_{\text{crypto}}$ only accepts x if it is not already recorded as a nonce (for anybody), modeling that x is fresh. Then, $\mathcal{F}_{\text{crypto}}$ records x as a nonce and sends x to the user.

We note that nonces are not confidential, their actual bit string is directly given to the user. Hence, $\mathcal{F}_{\text{crypto}}$ only prevents collisions of nonces. It would be easy to extend $\mathcal{F}_{\text{crypto}}$ to model confidential nonces. A nonce would be just a like a symmetric key (of some extra type for nonces). Users could refer to nonces by pointers and $\mathcal{F}_{\text{crypto}}$ would record the known/unknown status of nonces. Of course, nonces could not be used as a key (e.g., for encryption) but they could be part of plaintexts.

Corruption status request. The environment can ask, for a user $(p, lsid, r)$, whether a pointer *ptr* of this user is recorded as corrupted for this user. Similarly, the environment can ask whether the public/private key for encryption/signatures of a party is recorded as corrupted.

This concludes the description of $\mathcal{F}_{\text{crypto}}$. As explained for encryption requests, if a message x is encrypted under a known key (or by a corrupted public key), all keys in x are marked known in $\mathcal{F}_{\text{crypto}}$. Yet, if in an application the ciphertext y for x is encrypted again under an unknown key and y is always kept encrypted under an unknown key, the keys in x might not be revealed from the point of view of the application. While in such a case, $\mathcal{F}_{\text{crypto}}$ would be too pessimistic concerning the known/unknown status of keys, this case does typically not seem to occur: First, ciphertexts are typically sent unencrypted at some point. We are, in fact, not aware of any key exchange or secure channel protocol where this is not the case. Second, if in a session of a protocol symmetric keys known to the adversary are used, typically no security guarantees are provided for that session anyway.

As explained above, corruption is modeled on a per key basis. This allows to model many types of corruption, including corruption of single sessions and of complete parties (Section 5 provides details for our case study).

3.2 A Realization of the Ideal Crypto Functionality

In this section, we provide a realization $\mathcal{P}_{\text{crypto}}$ of the ideal crypto functionality $\mathcal{F}_{\text{crypto}}$.

Let Σ_{authenc}, $\Sigma_{\text{unauthenc}}$, Σ_{pub} be schemes for symmetric and public-key encryption, respectively, Σ_{mac} be a MAC scheme, Σ_{sig} be a digital signature scheme, and $F = \{F_\eta\}_{\eta \in \mathbf{N}}$ be a family of pseudo-random functions with $F_\eta \colon \{0,1\}^\eta \times \{0,1\}^* \to \{0,1\}^\eta$ for all $\eta \in \mathbf{N}$. For simplicity of presentation, we assume all symmetric keys to be chosen uniformly at random from $\{0,1\}^\eta$ (where η is the security parameter). Considering the family of pseudo-random functions F, we assume that the function $F_\eta(k, \cdot)$ for a randomly chosen $k \in \{0,1\}^\eta$ is indistinguishable from a random function, see Appendix A.5 for the precise definition of pseudo-randomness. These schemes induce a realization $\mathcal{P}_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$ in the obvious way where key derivation is realized by the family F of pseudo-random functions. To setup public/private keys for encryption/signatures, the adversary can send requests to $\mathcal{P}_{\text{crypto}}$ to generate these keys for a party p. In this case $\mathcal{P}_{\text{crypto}}$ will generate public/private keys using the key generation algorithms in Σ_{pub} or Σ_{sig}, respectively, and stores them for party p. Instead, the adversary might decide to corrupt the public/private key for encryption/signatures of the party p. In this case, the public/private keys are provided by the adversary and recorded as corrupted by $\mathcal{P}_{\text{crypto}}$. The realization $\mathcal{P}_{\text{crypto}}$ maintains symmetric keys and pointers to these keys in the same way as $\mathcal{F}_{\text{crypto}}$ does, but it does not maintain the sets $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$. $\mathcal{P}_{\text{crypto}}$ answers requests similarly to $\mathcal{F}_{\text{crypto}}$ as follows:

Generating fresh, symmetric keys $[(\mathsf{New}, t)]$**.** Upon generation of fresh, symmetric keys, the adversary is asked whether she wants to corrupt the key, in which case she provides the key and the pointer to this key is then marked corrupted in $\mathcal{P}_{\text{crypto}}$. Otherwise, $\mathcal{P}_{\text{crypto}}$ chooses the key uniformly at random from $\{0,1\}^\eta$.

Public key requests $[(\mathsf{GetPubKeyPKE}, p')$ or $(\mathsf{GetPubKeySig}, p')]$**.** A user $(p, lsid, r)$ can ask $\mathcal{F}_{\text{crypto}}$ to get the public key for party p' for encryption (resp., verification). If $\mathcal{F}_{\text{crypto}}$ has recorded this public key (because it has been generated upon request of the adversary, see above), then it is returned to the user. Otherwise, an error is returned to the user. We note that if users request the public key of another party, then this assumes that public keys are distributed somehow, e.g., by some kind of public key infrastructure.

Establishing pre-shared keys $[(\mathsf{GetPSK}, t, name)]$**.** Upon a request by a user to establish a pre-shared key of type t and with name *name*, $\mathcal{P}_{\text{crypto}}$ checks whether a key (t, k) is recorded for $(t, name)$ and $(t, name)$ is not corrupted. In this case, $\mathcal{P}_{\text{crypto}}$ creates a new pointer to (t, k) and returns it to the user. (We note that such a setup assumption for pre-shared keys is often made, e.g., it models manual distribution of keys by a system administrator.) Otherwise, i.e., no key is recorded for $(t, name)$ or $(t, name)$ is corrupted, we distinguish the following cases:

If $(t, name)$ is corrupted, the request is forwarded to the adversary who is supposed to provide a key k. Then, $\mathcal{P}_{\text{crypto}}$ creates a new pointer to (t, k), records this pointer as corrupted, and returns it to the user.

If $(t, name)$ is not corrupted but no key is recorded for $(t, name)$, the request is forwarded to the adversary who is asked whether the key is corrupted. In the case the key is corrupted the adversary provides the key k and $\mathcal{P}_{\text{crypto}}$ proceeds as in the case above. If the key is not corrupted, $\mathcal{P}_{\text{crypto}}$ chooses the key k uniformly at random from $\{0,1\}^\eta$ and records (t, k) for $(t, name)$. Then, $\mathcal{P}_{\text{crypto}}$ creates a new pointer to (t, k) and returns it to the user.

Key derivation [(Derive, ptr, t', s)]**.** Upon a request by a user to derive a new key of type t' from a key k of type pre-key with seed s, $\mathcal{P}_{\text{crypto}}$ computes $k' = F_\eta(k, \text{tag}_{t'}(s))$, creates a new pointer ptr to (t', k') for this user, and gives ptr to the user. Note that F_η is used with the seed $\text{tag}_{t'}(s)$, instead of just s. This guarantees that keys of different types are derived with different seeds. This kind of tagging is common also in real-world protocols in order to ensure that the same (derived) key is not used for different cryptographic operations.

Store [(Store, t, k)]**, retrieve** [(Retrieve, ptr)]**, equality test** [(Equal?, ptr, ptr')]**.** These requests are handled as in $\mathcal{F}_{\text{crypto}}$ except that $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$ are not maintained.

Encryption under symmetric and public keys [(Enc, ptr, m), (PKEnc, p', pk, m)]**.** Upon a request by a user to encrypt a well-tagged message m under a symmetric key k of type $t \in \{\text{authenc-key}, \text{unauthenc-key}\}$, all pointers in m are replaced by the keys they refer to (just as in $\mathcal{F}_{\text{crypto}}$). Unlike $\mathcal{F}_{\text{crypto}}$, the resulting message, say m', is then encrypted under k by running the encryption algorithm of Σ_{authenc} or $\Sigma_{\text{unauthenc}}$, depending on t, and the resulting ciphertext c is returned to the user. (Note that no extra randomness or tagging is added.) Requests for encryption under public keys are handled similarly using the encryption algorithm of Σ_{pub} and the public key pk contained in the request.

Decryption under symmetric and private keys [(Dec, ptr, c), (PKDec, c)]**.** Requests for the decryption of a ciphertext c under a symmetric key k of type $t \in \{\text{authenc-key}, \text{unauthenc-key}\}$ are answered by running decryption algorithm of Σ_{authenc} or $\Sigma_{\text{unauthenc}}$, depending on t, on the inputs k and c. If the decryption is successful and returns a well-tagged message, say m', then all keys in m' are replaced by new pointers (just as in $\mathcal{F}_{\text{crypto}}$) and the resulting message m is returned to the party. Otherwise, an error is returned. Requests for decryption under private keys are handled similarly using the decryption algorithm of Σ_{pub} and the private key recorded for this party.

Computing and verifying MACs [(Mac, ptr, m), (MacVerify, ptr, m, σ)]**.** Upon a request to MAC a message m under a key k of type mac-key, $\mathcal{P}_{\text{crypto}}$ simply returns the MAC computed using the MAC algorithm of Σ_{mac}. Upon a MAC verification request, $\mathcal{P}_{\text{crypto}}$ simply returns the result of the MAC verification algorithm of Σ_{mac}.

Computing and verifying signatures [(Sign, x), (SigVerify, p', pk, x, σ)]**.** Upon a request to sign a message x, $\mathcal{P}_{\text{crypto}}$ simply returns the signature computed using the signing algorithm of Σ_{sig} and the private key recorded for party p. Upon signature verification request, $\mathcal{P}_{\text{crypto}}$ simply returns the result of the signature verification algorithm of Σ_{sig} using the public key pk contained in the request.

Generating fresh nonces [(NewNonce)]**.** Upon generation of fresh nonces, $\mathcal{P}_{\text{crypto}}$ chooses the nonce uniformly at random from $\{0,1\}^\eta$ and gives it to the user.

Corruption status request. Similarly to $\mathcal{F}_{\text{crypto}}$, the environment can ask whether a pointer of a user or a public/private key of a party is corrupted.

3.3 Proving that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$

In this section, we prove that $\mathcal{P}_{\text{crypto}}$ is in fact a realization of $\mathcal{F}_{\text{crypto}}$.

As discussed in [35] for $\mathcal{F}_{\text{enc}}$, one cannot prove that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$ (in the presence of arbitrary environments) for standard assumptions about the symmetric encryption schemes Σ_{authenc} and $\Sigma_{\text{unauthenc}}$, namely authenticated encryption (IND-CPA and INT-CTXT security) and IND-CCA security, respectively, because it is easy to see that such a theorem does not hold in the presence of environments that may produce key cycles or cause the commitment problem: It is well-known that standard assumptions about symmetric encryption schemes are too weak to deal with key cycles [12, 5]. In the context of symmetric encryption, the commitment problem occurs if a key is revealed after it was used to encrypt a message. Before the key is revealed, messages encrypted under this key are encrypted ideally, i.e., the leakage of the message is encrypted. After the key has been revealed, the functionality would have to come up with a key such that the ciphertexts produced so far decrypt to the original messages. However, this is typically not possible (see, e.g., [4]). Therefore, as in [35] and similarly to [4], we restrict the class of environments that we consider basically to those environments that do not produce key cycles or cause the commitment problem, where, unlike [35] and [4], we now, when defining the class of environments, also need to be concerned about key derivation.

To formulate such a class of environments that captures what is typically encountered in applications, we observe, as was first pointed out in [4], that once a key has been used in a protocol to encrypt a message, this key is typically not

encrypted anymore in the rest of the protocol. Let us call these protocols *standard*. This observation can be generalized to *used-order respecting environments*, which we formulate based on $\mathcal{F}_{\text{crypto}}$: In what follows, we say that an unknown key k of type authenc-key, unauthenc-key, or pre-key has been *used (for encryption or key derivation)*, if $\mathcal{F}_{\text{crypto}}$ has been instructed to encrypt a message under k or to derive a new key from k. If k is of type unauthenc-key, we also consider k as used if it has successfully been used to decrypt a message. Now, an environment $\mathcal{E}$ (for $\mathcal{F}_{\text{crypto}}$) is called *used-order respecting* if it happens only with negligible probability that, in a run of $\mathcal{E}\,|\,\mathcal{F}_{\text{crypto}}$, an unknown key k (i.e., k is marked unknown in $\mathcal{F}_{\text{crypto}}$) that has been used at some point is encrypted itself by an unknown key k' used for the first time later than k. Clearly, such environments do not produce key cycles among unknown keys, with overwhelming probability. (We do not need to prevent key cycles among known keys.) Note that MAC keys are not problematic because they cannot be used to MAC other keys.

We say that an environment $\mathcal{E}$ *does not cause the commitment problem (is non-committing)*, if it happens only with negligible probability that, in a run of $\mathcal{E}\,|\,\mathcal{F}_{\text{crypto}}$, after an unknown key k has been used to encrypt a message or to derive a new key, k becomes known later on in the run, i.e., is marked known by $\mathcal{F}_{\text{crypto}}$. It is easy to see that for standard protocols, as introduced above, the commitment problem does not occur; see Section 5 for an example.

We can now state the theorem which shows that, given that F is a pseudo-random function family, realizing $\mathcal{F}_{\text{crypto}}$ by $\mathcal{P}_{\text{crypto}}$ is *equivalent* to the encryption and MAC schemes being IND-CCA, authenticated (IND-CPA and INT-CTXT), and UF-CMA secure, respectively, where we use the standard definitions of these security notions. (See Appendix A for a precise definition of these cryptographic security definitions.) In other words, $\mathcal{F}_{\text{crypto}}$ *exactly* captures IND-CCA security, authenticated encryption, and UF-CMA security.

In the theorem, stated below, instead of explicitly restricting the class of environments described above, we introduce a functionality $\mathcal{F}^*$ that provides exactly the same I/O interface as $\mathcal{F}_{\text{crypto}}$ (and hence, $\mathcal{P}_{\text{crypto}}$), but before forwarding requests to $\mathcal{F}_{\text{crypto}}$/$\mathcal{P}_{\text{crypto}}$ checks whether the used-order is still respected and the commitment problem is not caused. Otherwise, $\mathcal{F}^*$ raises an error flag and from then on blocks all messages, i.e., effectively stops the run. It is easy to see that all information needed to perform these checks can be obtained from observing the I/O interface of $\mathcal{F}_{\text{crypto}}$.

Theorem 3. *Let $\Sigma_{\text{authenc}}, \Sigma_{\text{unauthenc}}, \Sigma_{\text{pub}}$ be encryption schemes as above, where the domain of plaintexts is the set of well-tagged bit strings. Let Σ_{mac} be a MAC scheme, Σ_{sig} be a signature scheme, and F be a pseudo-random function family as above. Let L be a leakage algorithm which leaks exactly the length of a message. Then,*

$$\mathcal{F}^*\,|\,\mathcal{P}_{\text{crypto}} \leq \mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$$

if and only if $\Sigma_{\text{unauthenc}}$ and Σ_{pub} are IND-CCA, Σ_{authenc} is IND-CPA and INT-CTXT, and Σ_{mac} and Σ_{sig} are UF-CMA secure. (The direction from right to left holds for any plaintext domains of the encryption schemes.)

Since derived keys can be encrypted and used as encryption keys, the security of encryption depends on the security of key derivation and vice versa. Therefore, we need to carry out a single hybrid argument, intertwining both encryption and key derivation. Next, we present a proof sketch (see Appendix C for a full proof).

Proof sketch. The direction from left to right is easy to prove by standard cryptographic reductions and can be found in the appendix. To prove the direction from right to left, we proceed as follows. First, we replace public-key encryption and digital signatures in $\mathcal{P}_{\text{crypto}}$ by the ideal functionalities $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{sig}}$ for public-key encryption and digital signatures, respectively, presented in [33]. We note that similar ideal functionalities for public-key encryption and digital signatures have been presented previously, e.g., [16, 19, 15]. Using results proved in [33] (and that Σ_{pub} is IND-CCA and Σ_{sig} is UF-CMA secure), we then show for the resulting system $\mathcal{P}'_{\text{crypto}}$ that $\mathcal{F}^*\,|\,\mathcal{P}_{\text{crypto}} \leq \mathcal{F}^*\,|\,\mathcal{P}'_{\text{crypto}}$. It remains to prove that $\mathcal{F}^*\,|\,\mathcal{P}'_{\text{crypto}} \leq \mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$. Roughly speaking, the simulator Sim_{crypto} we use for that answers requests from $\mathcal{F}_{\text{crypto}}$ in such a way that they match the behavior of $\mathcal{P}'_{\text{crypto}}$ except that Sim_{crypto} generates a fresh key upon key derivation from an unknown key (instead of using the pseudo-random function for key derivation). The rest of the proof proceeds by a hybrid argument:

We define hybrid systems $\mathcal{F}^{(r)}_{\text{crypto}}$ for every $r \in \mathbf{N}$. The r-th hybrid $\mathcal{F}^{(r)}_{\text{crypto}}$ behaves like $\mathcal{F}_{\text{crypto}}$ except for the following. $\mathcal{F}^{(r)}_{\text{crypto}}$ keeps track of the order in which unknown keys are first used for encryption, key derivation, or decryption (in case of unauthenticated encryption). The first key used for one of these operations has *used-order* 1, the second has *used-order* 2, and so on. Now, all keys that have used-order less than r are treated *ideal*, i.e., as in

$Sim_{\text{crypto}} \,|\, \mathcal{F}_{\text{crypto}}$, while the others are treated *real*, i.e., as in $\mathcal{P}'_{\text{crypto}}$. All MAC keys are treated real; we replace real MACs by ideal MACs in a second step. Then, we show that the 0-th hybrid is indistinguishable from the real system and that the $p(\eta)$-th hybrid (where η is the security parameter and p is a polynomial that bounds the runtime of the environment) is indistinguishable from the ideal system. The latter requires a hybrid argument itself to replace real MACs by ideal MACs. Finally, we show that the r-th hybrid is indistinguishable from the $(r+1)$-th hybrid: The two hybrids only differ in the handling of the r-th key, say k, which is treated ideal and real, respectively. If k was obtained by key derivation, it was derived by one of the keys with used-order $< r$, and hence, it was derived in an ideal way, distributed just like a fresh key. Moreover, k is at most encrypted by keys with used-order $< r$, and hence, encrypted ideally. This now allows the reduction to the indistinguishability games for encryption (if k is of type authenc-key or unauthenc-key) and key derivation (if k is of type pre-key). □

Theorem 3, together with the composition theorems, yields the following corollary, which gets rid of the functionality $\mathcal{F}^*$, assuming that $\mathcal{F}_{\text{crypto}}$ is used by what we call a non-committing, used-order respecting protocol. A protocol system $\mathcal{P}$ that uses $\mathcal{F}_{\text{crypto}}$ is called *non-committing, used-order respecting* if the probability that in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$ the functionality $\mathcal{F}^*$ raises the error flag, is negligible for any environment $\mathcal{E}$, connecting to both I/O and network interfaces. As mentioned above, most protocols have this property and this can typically be easily checked by inspection of the protocol. For example, standard protocols (see above) are non-committing and used-order respecting because unknown keys are never encrypted (by other keys) after they have been used. In particular, since corruption of keys is static, if the key is unknown at the moment it is first used, it will remain unknown. We note that corruption of whole parties (or users) where the adversary controls the party can be defined in such a way that a corrupted party cannot obtain a pointer to a key marked unknown. Thus, even for such a modeling of corruption, a protocol can be standard, and hence, non-committing and used-order respecting; see Section 5 and [35] for examples.

Corollary 1. *Let Σ_{authenc}, $\Sigma_{\text{unauthenc}}$, Σ_{pub}, Σ_{mac}, Σ_{sig}, F, and L be given as in Theorem 3. Let $\mathcal{P}$ be a non-committing, used-order respecting protocol system. Then,*

$$\mathcal{P} \,|\, \mathcal{P}_{\text{crypto}} \leq \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$$

if $\Sigma_{\text{unauthenc}}$ and Σ_{pub} are IND-CCA, Σ_{authenc} is IND-CPA and INT-CTXT, and Σ_{mac} and Σ_{sig} are UF-CMA secure.

As demonstrated in the following sections, with Theorem 3 and Corollary 1 protocols can first be analyzed based on $\mathcal{F}_{\text{crypto}}$ and then $\mathcal{F}_{\text{crypto}}$ can be replaced by its realization $\mathcal{P}_{\text{crypto}}$.

We note that the joint state composition theorems for public-key encryption and symmetric encryption under pre-shared keys in [35] carry over to $\mathcal{F}_{\text{crypto}}$. That is, we can prove that a—so called—*joint state realization* of $\mathcal{F}_{\text{crypto}}$ realizes the multi-session version of $\mathcal{F}_{\text{crypto}}$. However, as explained in Section 4, we do not use composition with joint state in this paper.

4 Applications to Key Exchange and Secure Channels

In this section, we consider a general class of key exchange and secure channel protocols which use the functionality $\mathcal{F}_{\text{crypto}}$ (or its realization $\mathcal{P}_{\text{crypto}}$) and develop criteria to prove universally composable security for such protocols. Since our criteria are based on $\mathcal{F}_{\text{crypto}}$, proving the criteria merely requires information-theoretic arguments or purely syntactical arguments (without reasoning about probabilities), rather than involved cryptographic reduction proofs.

Our criteria are formulated w.r.t. multiple protocol sessions. Alternatively, we could formulate them for single sessions and then extend them to the multiple session case by joint state theorems [19, 33, 35]. However, in order for our models to be very close to the actual (real-world) protocols, in this paper, we avoid these theorems: First, they rely on the setup assumption that the parties participating in a session already "magically" agreed upon a unique session identifier (SID). Real-world protocols do not rely on this assumption. Second, in joint state realizations, SIDs are explicitly added to messages before encryption, signing, and MACing, i.e., in a session with SID *sid*, instead of the actual message, say m, the message (sid, m) is encrypted, signed, or MACed. While this is a good design principle, it modifies the actual protocols.

4.1 A Criterion for Universally Composable Key Exchange

We define an ideal functionality for (multi-session) key exchange $\mathcal{F}_{\text{ke}}$, formulate a general class of key exchange protocols that use $\mathcal{F}_{\text{crypto}}$ for cryptographic operations, and present a criterion which allows us to prove that a key exchange protocol in this class realizes $\mathcal{F}_{\text{ke}}$.

The Ideal Key Exchange Functionality. The basic idea of an ideal functionality for key exchange $\mathcal{F}_{\text{ke}}$, see, e.g., [16], is that parties can send requests to $\mathcal{F}_{\text{ke}}$ to exchange a key with other parties and then, in response, receive a session key which is generated by $\mathcal{F}_{\text{ke}}$ and guaranteed to be i) the same for every party in the same session and ii) only known to these parties. As mentioned above and unlike other formulations, our functionality directly allows to handle an unbounded number of sessions between arbitrary parties.

More precisely, similarly to $\mathcal{F}_{\text{crypto}}$, our ideal key exchange functionality $\mathcal{F}_{\text{ke}}$ is parameterized by a number n and has n I/O input and output tapes, one pair for each role, e.g., $n = 2$ in case of a two-party key exchange protocol. To address multiple sessions of a party, the parties identify themselves to $\mathcal{F}_{\text{ke}}$ as a *user* (similarly to $\mathcal{F}_{\text{crypto}}$), represented by a tuple $(p, lsid, r)$, where p is the PID of the party, $lsid$ a local session ID chosen and managed by the party itself, and the role $r \in \{1, \ldots, n\}$. On the tape for role r, $\mathcal{F}_{\text{ke}}$ expects requests to be prefixed by tuples of the form $(p, lsid)$, and conversely $\mathcal{F}_{\text{ke}}$ prefixes answers with $(p, lsid)$. For every user a corresponding *local session* is managed in $\mathcal{F}_{\text{ke}}$, which contains the state of the key exchange for this user. To initiate a key exchange, a user, say $(p, lsid, r)$, can send a *session-start* message of the form $(\mathsf{Start}, p_1, \ldots, p_n)$, with $p = p_r$, to $\mathcal{F}_{\text{ke}}$, where the PIDs $p_1, \ldots, p_n$ are the desired partners of p in the n roles for the key exchange. Upon such a request, $\mathcal{F}_{\text{ke}}$ records this *session-start* message as a local session for user $(p, lsid, r)$ and informs the (ideal) adversary about this request by forwarding it to her. The adversary determines (at some point) to which *global* session local sessions belong, by sending a *session-create* message of the form $(\mathsf{Create}, (p_1, lsid_1, 1), \ldots, (p_n, lsid_n, n))$ to $\mathcal{F}_{\text{ke}}$, containing one local session for every role. The functionality $\mathcal{F}_{\text{ke}}$ only accepts such a message if it is consistent with the local sessions: The mentioned local sessions all exist, are uncorrupted (see below) and are not already part of another global session, and the desired partners in the local sessions correspond to each other. For a global session, $\mathcal{F}_{\text{ke}}$ creates a fresh key—called the *session key*—according to some probability distribution. For a local session $(p, lsid, r)$ which is part of a global session in $\mathcal{F}_{\text{ke}}$, the adversary can send a *session-finish* message of the form $(\mathsf{Finish}, (p, lsid, r))$ to $\mathcal{F}_{\text{ke}}$, upon which $\mathcal{F}_{\text{ke}}$ sends a *session-key-output* message of the form $(\mathsf{SessionKey}, k)$ to the user $(p, lsid, r)$, which contains the session key k for this session.

The adversary can corrupt a local session $(p, lsid, r)$ which is not already part of a global session by sending a *corrupt* message of the form $(\mathsf{Corrupt}, (p, lsid, r))$ to $\mathcal{F}_{\text{ke}}$. For a corrupted local session, the adversary may determine the session key by sending a *session-finish* message of the form $(\mathsf{Finish}, (p, lsid, r), k)$ to $\mathcal{F}_{\text{ke}}$, upon which $\mathcal{F}_{\text{ke}}$ sends a *session-key-output* message of the form $(\mathsf{SessionKey}, k)$ to the user $(p, lsid, r)$, which contains the session key k chosen by the adversary. As usual, the environment can ask whether a local session is corrupted or not.

Key Exchange Protocols. An *$\mathcal{F}_{\text{crypto}}$-key exchange protocol ($\mathcal{F}_{\text{crypto}}$-KE protocol)* $\mathcal{P}$ is of the form $\mathcal{P} = !M_1 \mid \ldots \mid !M_n \mid \mathcal{F}_1 \mid \ldots \mid \mathcal{F}_l$ for some n and machines (IITMs) $M_1, \ldots, M_n$ and ideal functionalities (formally, protocol systems) $\mathcal{F}_1, \ldots, \mathcal{F}_l$. Each machine (IITM) M_r represents one role in the protocol and there can be multiple instances of each machine, namely, one instance of M_r for each local session $(p, lsid, r)$; this can be ensured by the $\mathsf{CheckAddress}$ mode of M_r. An instance may arbitrarily communicate with the adversary (the network) and may use $\mathcal{F}_{\text{crypto}}$ and the ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_l$; we note that the environment cannot directly access the I/O interfaces of $\mathcal{F}_1, \ldots, \mathcal{F}_l, \mathcal{F}_{\text{crypto}}$, only via the IITMs $M_1, \ldots, M_n$. The functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_l$ may provide additional cryptographic operations such as public-key certification. Analogously to $\mathcal{F}_{\text{ke}}$, a user $(p, lsid, r)$ initiates a key exchange by sending a *session-start* message to M_r (on the I/O interface), which creates a new instance of M_r. At some point, every instance of M_r may return a *session-key-pointer-output* message of the form $(\mathsf{SessionKeyPointer}, ptr)$ to the user $(p, lsid, r)$ which contains a pointer ptr, called the *session key pointer*, to the actual session key stored in $\mathcal{F}_{\text{crypto}}$; so, unlike $\mathcal{F}_{\text{ke}}$, only a pointer to the session key, rather than the actual key, is output (see below for a variant of $\mathcal{P}$ which is closer to $\mathcal{F}_{\text{ke}}$). This instance then provides the user $(p, lsid, r)$ with an interface to $\mathcal{F}_{\text{crypto}}$ where initially only the session key pointer ptr may be used (but subsequently other pointers can be generated). More precisely, the user $(p, lsid, r)$ may send any request for $\mathcal{F}_{\text{crypto}}$ to M_r, such as encryption, decryption, and key derivation requests. Upon such a request, M_r forwards this request to $\mathcal{F}_{\text{crypto}}$ and waits for receiving an answer from $\mathcal{F}_{\text{crypto}}$ which is forwarded to the user $(p, lsid, r)$. However, we require that all pointers in such a request have been output by M_r to this user before and that the session key pointer is never encrypted or explicitly revealed by a retrieve command (see below for an example). Before forwarding requests

to $\mathcal{F}_{\text{crypto}}$, M_r checks whether this requirement is satisfied; if the check fails, M_r returns an error message to the user $(p, lsid, r)$.

For example, after having received $(\mathsf{SessionKeyPointer}, ptr)$ from M_r, the user $(p, lsid, r)$ might send the request (New, t) to M_r upon which M_r will forward it to $\mathcal{F}_{\text{crypto}}$. Then, $\mathcal{F}_{\text{crypto}}$ will return a new pointer ptr' to M_r which is forwarded by M_r to the user $(p, lsid, r)$. To encrypt a message m which contains the pointer ptr' (and no other pointer, say) under the session key pointer ptr, $(p, lsid, r)$ sends the request (Enc, ptr, m) to M_r. Then, M_r will forward this message to $\mathcal{F}_{\text{crypto}}$ because all pointers in this request, i.e., ptr and ptr', have been output to this user before. Finally, the ciphertext returned by $\mathcal{F}_{\text{crypto}}$ is forwarded to the user $(p, lsid, r)$.

We do not fix a special form of corruption but leave the modeling of corruption to the definition of the protocol $\mathcal{P}$, up to the following conditions: i) the environment can ask about the corruption status of local sessions (as in $\mathcal{F}_{\text{ke}}$), ii) once a local session is corrupted, it stays corrupted, and iii) a local session cannot be corrupted after it has returned a *session-key-pointer-output* message. (See our case study in Section 5 for an example.)

We also consider a variant $\widehat{\mathcal{P}}$ of an $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ defined as follows: Instead of sending *session-key-pointer-output* messages, $\widehat{\mathcal{P}}$ sends *session-key-output* messages (as $\mathcal{F}_{\text{ke}}$) which contain the actual key the session key pointer refers to. This key is obtained using the retrieve command $(\mathsf{Retrieve}, ptr)$ of $\mathcal{F}_{\text{crypto}}$. Furthermore, in contrast to $\mathcal{P}$, $\widehat{\mathcal{P}}$ does not provide the environment with an interface to $\mathcal{F}_{\text{crypto}}$, i.e., $\widehat{\mathcal{P}}$ does not forward requests to $\mathcal{F}_{\text{crypto}}$. Note that the protocol $\widehat{\mathcal{P}}$ has the same I/O interface as the ideal functionality $\mathcal{F}_{\text{ke}}$; it is in fact meant to realize $\mathcal{F}_{\text{ke}}$ (see below). The advantage of $\mathcal{P}$ over $\widehat{\mathcal{P}}$ is that a session key pointer can still be used for *ideal* cryptographic operations, e.g., ideal encryption or even to establish an ideal secure channel (see below).

Criterion for Secure Key Exchange Protocols. We now present a sufficient criterion for an $\mathcal{F}_{\text{crypto}}$-KE protocol to realize $\mathcal{F}_{\text{ke}}$, and hence, to provide universally composable key exchange. The criterion is based on partnering functions.

A *partnering function* τ for an $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ is a polynomial-time computable function that maps a sequence of configurations of $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ to a set of tuples of the form $(s_1, \ldots, s_n)$, where s_r is of the form $(p, lsid, r)$, i.e., s_r refers to an instance of M_r, for all $r \leq n$. We say that the instances $s_1, \ldots, s_n$ *form a (global) session according to* τ. We call τ *valid for* $\mathcal{P}$ if for any environment $\mathcal{E}$ for $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ and any run of $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ the following holds, where τ operates on the projection of the runs to configurations of $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$: i) All instances occur in at most one session (according to τ). ii) Instances in one session agree on the PIDs of the desired partners. iii) τ is monotonic, i.e., once a session has been established according to τ, it continues to exist. Now, we are ready to state our criterion.

Definition 2. *We say that an $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ is* strongly $\mathcal{F}_{\text{crypto}}$-secure (with type t_0 of a key) w.r.t. a partnering function τ and an environment $\mathcal{E}$ of $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ *if for runs of $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ the following holds with overwhelming probability: For every uncorrupted instance of M_r, say $(p, lsid, r)$, which has output a session key pointer to say the key k in $\mathcal{F}_{\text{crypto}}$ it holds that:*

i) The local session $(p, lsid, r)$ belongs to some global session (according to τ) which contains only uncorrupted local sessions.
ii) The key k is of type t_0 and marked unknown in $\mathcal{F}_{\text{crypto}}$.
iii) The key k has never been used in $\mathcal{F}_{\text{crypto}}$ as a key for encryption, key derivation, or to compute a MAC by any user, except through the interface to $\mathcal{F}_{\text{crypto}}$ provided to the environment after a session-key-pointer-output message.
iv) Session key pointers (if any) of other instances point to the same key k if and only if they belong to the same session as $(p, lsid, r)$ (according to τ).

An $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ is strongly $\mathcal{F}_{\text{crypto}}$-secure *if there exists a valid partnering function τ for $\mathcal{P}$ and $\mathcal{P}$ is strongly $\mathcal{F}_{\text{crypto}}$-secure w.r.t. τ and $\mathcal{E}$ for every environment $\mathcal{E}$ for $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$.*

The following theorem states that this criterion is indeed sufficient for an $\mathcal{F}_{\text{crypto}}$-KE protocol to realize the ideal key exchange functionality $\mathcal{F}_{\text{ke}}$.

Theorem 4. *Let $\mathcal{P}$ be an $\mathcal{F}_{\text{crypto}}$-KE protocol. If $\mathcal{P}$ is strongly $\mathcal{F}_{\text{crypto}}$-secure and $\widehat{\mathcal{P}}$ is used-order respecting and non-committing, then $\widehat{\mathcal{P}} \mid \mathcal{P}_{\text{crypto}} \leq \mathcal{F}_{\text{ke}}$.*

Proof. Let $\mathcal{S} = \widehat{\mathcal{P}} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}_{\text{crypto}}$, where Sim_{crypto} is the simulator used to prove Theorem 3. We note that, by Corollary 1, it holds that $\mathcal{E} \,|\, \widehat{\mathcal{P}} \,|\, \mathcal{P}_{\text{crypto}} \equiv \mathcal{E} \,|\, \mathcal{S}$ for every environment $\mathcal{E}$. Next, we define a simulator *Sim* for $\mathcal{F}_{\text{ke}}$ and show that $\mathcal{E} \,|\, \mathcal{S} \equiv \mathcal{E} \,|\, Sim \,|\, \mathcal{F}_{\text{ke}}$ for every environment $\mathcal{E}$, which completes the proof.

Since $\mathcal{P}$ is strongly $\mathcal{F}_{\text{crypto}}$-secure, there exists a valid partnering function τ. The simulator *Sim* is defined as follows: It basically emulates $\mathcal{S}$. More precisely, if $\mathcal{F}_{\text{ke}}$ receives a *session-start* message, $\mathcal{F}_{\text{ke}}$ forwards it to *Sim* which forwards it to (the emulated) $\mathcal{S}$. Hence, every local session, say $(p, lsid, r)$, in $\mathcal{F}_{\text{ke}}$ corresponds to the instance $(p, lsid, r)$ in $\mathcal{S}$. If an instance in $\mathcal{S}$ gets corrupted, i.e., would return *corrupted* upon a corruption request from the environment, *Sim* corrupts the corresponding local session in $\mathcal{F}_{\text{ke}}$. If a corrupted instance outputs a session key k, *Sim* instructs the corresponding local session in $\mathcal{F}_{\text{ke}}$ to output k. If an uncorrupted instance outputs a session key, *Sim* determines the session, i.e., an instance for every role, by using the partnering function τ. Note that by i) of Definition 2, all instances in this session are uncorrupted. Then, *Sim* sends two messages to $\mathcal{F}_{\text{ke}}$, first, a *session-create* message which contains the determined session and, second, a *session-finish* message for the corresponding local session. Note that since τ is valid and all instances in this session are uncorrupted, $\mathcal{F}_{\text{ke}}$ accepts all these messages. Network communication is forwarded directly by *Sim* between the environment and the emulated $\mathcal{S}$.

It is now possible to define a mapping from every run ρ of the real system $\mathcal{E} \,|\, \mathcal{S}$, excluding the (negligible set of) runs for which one of the conditions in Definition 2 is not satisfied, to a set S_ρ of runs of the ideal system $\mathcal{E} \,|\, Sim \,|\, \mathcal{F}_{\text{ke}}$ such that the probability of ρ is the same as the one for S_ρ and the overall output of ρ (on tape decision) is the same as the overall output of every run in S_ρ. Such a mapping implies that $\mathcal{E} \,|\, \mathcal{S} \equiv \mathcal{E} \,|\, Sim \,|\, \mathcal{F}_{\text{ke}}$. The mapping is defined as follows: Let ρ be a run of $\mathcal{E} \,|\, \mathcal{S}$ which satisfies the conditions in Definition 2. We now define S_ρ to be the set of runs of $\mathcal{E} \,|\, Sim \,|\, \mathcal{F}_{\text{ke}}$ where

i) $\mathcal{E}$ uses the same random coins as $\mathcal{E}$ in ρ,
ii) the random coins of $\mathcal{F}_{\text{ke}}$ are defined such that the keys that are generated and output as session keys by $\mathcal{F}_{\text{ke}}$ correspond to the keys generated by $\mathcal{S}$ and output as session keys by $\mathcal{S}$ to $\mathcal{E}$ in ρ (this is to make sure that the session keys output in the ideal and real system coincide), and
iii) the random coins of *Sim* are defined such that the emulated $\mathcal{S}$ (within *Sim*) uses the same random coins as $\mathcal{S}$ in ρ, except that we use fresh keys for keys generated by $\mathcal{S}$ and output as session keys by $\mathcal{S}$ to $\mathcal{E}$ for uncorrupted local sessions in ρ. Every such fresh key induces one run in S_ρ. Note that for corrupted local sessions $\mathcal{E}$ determines the session keys, and hence, in such a case the ideal and real system behave in the same way.

Note that this mapping is well-defined because ρ satisfies the conditions in Definition 2. By construction, the probabilities of ρ and S_ρ are equal. Based on the following observations, one can easily show, by induction on the length of runs, that the view of $\mathcal{E}$ in ρ is the same as the view of $\mathcal{E}$ in every run ρ' in S_ρ, and hence, the overall output is the same: The only difference between ρ and ρ' is that for every key that is output as a session key in an uncorrupted local session in ρ', *Sim* does not use this session key in its simulation, but a freshly generated key. By definition of the randomness of ρ', the corresponding session in $\mathcal{F}_{\text{ke}}$ contains the actual session key. For this reason, output at the I/O interface of ρ and ρ' coincide: The session keys output by uncorrupted local sessions coincide in both runs, ρ and ρ'. As for the network interface, even though ρ' does not use the actual session key, $\mathcal{E}$ cannot observe this: By assumption, ρ satisfies the conditions in Definition 2. Thus, keys output in ρ as session keys in uncorrupted local sessions are marked unknown, and hence, they have always been encrypted ideally. Moreover, these keys were never used as keys for encryption, MACing, or key derivation. So, $\mathcal{E}$ cannot distinguish whether the actual session keys or freshly generated keys have been encrypted. □

We note that the concept of partnering functions has been used in game-based security definitions which led to discussions whether the obtained security definitions are reasonable, see, e.g., [9, 10, 8, 17, 20, 31]. Here, we use partnering functions as part of our criterion but not as part of the security definition; security means realizing $\mathcal{F}_{\text{ke}}$.

4.2 Applications to Secure Channels

A secure channel, see, e.g., [18], between two parties provides confidentiality and authenticity of the messages sent over the channel and prevents rearrangement and replay of messages. Some secure channels also prevent message loss.

In this section, we first define two ideal functionalities for secure channels $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}_{\text{sc}}^{+}$, where, unlike $\mathcal{F}_{\text{sc}}$, $\mathcal{F}_{\text{sc}}^{+}$ prevents message loss. Just as $\mathcal{F}_{\text{ke}}$ and in contrast to previous formulations, our functionalities directly allow to handle an unbounded number of sessions between arbitrary parties.

We consider two generic realizations of $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}_{\text{sc}}^{+}$, namely $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}_{\text{sc}}^{+}$, respectively, which use an $\mathcal{F}_{\text{crypto}}$-key exchange protocol $\mathcal{P}$ as a sub-protocol. Every session of $\mathcal{P}_{\text{sc}}$ (analogously for $\mathcal{P}_{\text{sc}}^{+}$) runs a session of $\mathcal{P}$ to exchange 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.

Finally, we provide a criterion for $\mathcal{F}_{\text{crypto}}$-KE protocols and show that $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}_{\text{sc}}^{+}$ realize $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}_{\text{sc}}^{+}$, respectively, if the underlying $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ satisfies this criterion. We could use the criterion "strongly $\mathcal{F}_{\text{crypto}}$-secure", but in fact a weaker variant suffices, which we call α-$\mathcal{F}_{\text{crypto}}$-*secure*. Unlike strong $\mathcal{F}_{\text{crypto}}$-security, α-$\mathcal{F}_{\text{crypto}}$-security allows that session keys are used in the key exchange protocol (e.g., for key confirmation). But then, messages encrypted under these keys in the key exchange protocol should not interfere with messages sent over the secure channel. We therefore consider a set of messages α which contains at least all possible plaintexts that potentially can occur in the key exchange protocol encrypted under the session key. Usually, (an over-approximation of) α can be described based on the different message formats in the protocols.

The Ideal Secure Channel Functionalities. We first describe $\mathcal{F}_{\text{sc}}$. It has the same I/O and network interface as $\mathcal{F}_{\text{ke}}$ and is parameterized by a number n which defines the number of roles in the protocol. Just as $\mathcal{F}_{\text{ke}}$, $\mathcal{F}_{\text{sc}}$ manages local sessions of users and (global) sessions: A user $(p, lsid, r)$ can send *session-start* messages which create local sessions. The (ideal) adversary groups local sessions to form (global) sessions by sending *session-create* messages and completes local sessions by sending *session-finish* messages.

Upon a *session-finish* message, in contrast to $\mathcal{F}_{\text{ke}}$, which sends a *session-key-output* message to the user, $\mathcal{F}_{\text{sc}}$ sends a *session-established* message to the user notifying it that the secure channel has been established successfully. Now, this user, say $(p, lsid, r)$, can send messages to its partners in the same session by sending *send* requests of the form (Send, m) to $\mathcal{F}_{\text{sc}}$ where m is the message that is supposed to be sent. Upon receiving such a request, $\mathcal{F}_{\text{sc}}$ adds m to a queue for this local session and the intended recipient, i.e., one of the partners in the local session. Then, $\mathcal{F}_{\text{sc}}$ forwards this request to the adversary but replaces m by $0^{|m|}$, revealing the length of m but nothing more. The adversary can now instruct $\mathcal{F}_{\text{sc}}$ to deliver or drop messages. If the adversary sends a *deliver* message for a local session to $\mathcal{F}_{\text{sc}}$, $\mathcal{F}_{\text{sc}}$ removes the first message from the queue for this local session and intended recipient and sends it to the intended recipient, i.e., to the user corresponding to it. If the adversary sends a *drop* message for a local session to $\mathcal{F}_{\text{sc}}$, then $\mathcal{F}_{\text{sc}}$ only removes the first message from the queue but does not send it to the intended recipient.

Similarly to $\mathcal{F}_{\text{ke}}$, the adversary has the ability to corrupt a local session of a party if it is not already part of a session. If a corrupted local session receives a *send* request, this request is forwarded to the adversary unaltered, i.e., the message is revealed to the adversary. Furthermore, the adversary can instruct $\mathcal{F}_{\text{sc}}$ to deliver arbitrary messages to the user of a corrupted local session. (See below for the discussion of arbitrary network input.)

The variant $\mathcal{F}_{\text{sc}}^{+}$ coincides with $\mathcal{F}_{\text{sc}}$, except that the adversary does not have the ability to drop messages, i.e., $\mathcal{F}_{\text{sc}}^{+}$ does not accept *drop* messages.

We note that, for an uncorrupted session both $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}_{\text{sc}}^{+}$ guarantee secrecy and authenticity of messages and that messages arrive in the right order, without replay. But $\mathcal{F}_{\text{sc}}$ allows message loss while $\mathcal{F}_{\text{sc}}^{+}$ prevents this. Clearly, if a protocol realizes $\mathcal{F}_{\text{sc}}^{+}$, it also realizes $\mathcal{F}_{\text{sc}}$, because we have $\mathcal{F}_{\text{sc}}^{+} \leq \mathcal{F}_{\text{sc}}$.

Generic Secure Channel Protocols. We define two generic secure channel protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}_{\text{sc}}^{+}$ of the form $!M_1 \mid \ldots \mid !M_n$ which use an $\mathcal{F}_{\text{crypto}}$-KE protocol, say $\mathcal{P} = !M'_1 \mid \ldots \mid !M'_n \mid \mathcal{F}_1 \mid \ldots \mid \mathcal{F}_l$, as a sub-protocol. In a nutshell, a session of $\mathcal{P}_{\text{sc}}$ (and also $\mathcal{P}_{\text{sc}}^{+}$) runs a session of $\mathcal{P}$ to exchange 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 parameterized by what we call a *plaintext construction function* f which takes two PIDs p_1, p_2, a counter v, and a message m and outputs a plaintext $f(p_1, p_2, v, m)$. We require that $f(p_1, p_2, v, m)$ is a well-tagged bit string which does not contain any pointers, 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. Just as for $\mathcal{F}_{\text{crypto}}$-KE protocols, there is one machine M_r for every role $i \leq n$ which 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 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 M'_r (the key exchange protocol) and waits for receiving a *session-key-pointer-output* message from M'_r. Then, M_r sends a *session-established* message 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 $j \leq n$, $j \neq r$. All counters are initialized with 0.

After a session has been established and upon receiving a *send* request triggering M_r, say for user $(p, lsid, r)$, to send a message m to the party with role j (in the same session), M_r constructs the message $m' = f(p, p', S_j, m)$ where p' is the PID of the receiver, increases S_j by one, encrypts m' with the session key pointer received from M'_r (using $\mathcal{F}_{\text{crypto}}$ through M'_r), obtains a ciphertext c, and sends (p, p', c) 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, c) where p' is the PID of an intended partner of $(p, lsid, r)$ for some role, say $j \neq r$, M_r decrypts c using the session key pointer ptr. If the decryption succeeds and the plaintext is of the form $f(p', p, v, m)$ for some number $v \geq R_j$ and some message m, then M_r sets $R_j = v + 1$ and outputs m 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 *session-established* message. The instance of M_r is also considered *corrupted* if the corresponding local session of the key exchange protocol $\mathcal{P}$ is corrupted. The environment may ask whether or not an instance of M_r has been corrupted.

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

Note that the messages received by M_r from the network are again output (at least parts of these messages) by M_r. Because the network input tapes are consuming, the length of these message has to be bound by a polynomial in the security parameter plus the length of messages received from the user (environment) so far. (Recall that the messages received from the user are received on enriching tapes.) This aspect has not been discussed in the description above. We can allow the environment to send resources to M_r. This way it is possible that arbitrary many network inputs of arbitrary length can be processed. This mechanism has been first used in [33].

Criterion for Secure Channel Protocols. We now provide a criterion for $\mathcal{F}_{\text{crypto}}$-KE protocols and prove that the generic secure channel protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}^+_{\text{sc}}$ realize $\mathcal{F}_{\text{sc}}$ and $\mathcal{F}^+_{\text{sc}}$, respectively, if the underlying $\mathcal{F}_{\text{crypto}}$-KE protocol satisfies this criterion. We could use the criterion "strongly $\mathcal{F}_{\text{crypto}}$-secure", but in fact a weaker variant suffices where the session key may have been used by the key exchange protocol, e.g., for key confirmation.

However, in order to be able to prove the desired property with this weaker criterion, we need to require that messages sent over the secure channel cannot be confused with messages exchanged in the key exchange protocol. This is typically the case because of different message formats used in the key exchange and secure channel protocol, e.g., messages may be tagged with the protocol names. To capture this formally, we consider a set of messages α which contains at least all possible plaintexts that potentially can occur in the key exchange protocol encrypted under the session key. As mentioned, (an over-approximation of) α can typically be described based on the different message formats in the protocols.

Definition 3. *Let α be a set of bit strings such that α can be decided in polynomial time. Furthermore, let t_0 be some type of a key. We say that an $\mathcal{F}_{\text{crypto}}$-KE protocol $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-*secure (with type t_0) *if it is strongly $\mathcal{F}_{\text{crypto}}$-secure, except that we replace condition iii) for strong $\mathcal{F}_{\text{crypto}}$-security by the following condition, with $(p, lsid, r)$ and (t, k) as in Definition 2:*

iii) The key (t, k) has only been used in $\mathcal{F}_{\text{crypto}}$ as a key by users that belong to the same session as $(p, lsid, r)$ (according to τ) to encrypt messages that belong to α, except through the interface to $\mathcal{F}_{\text{crypto}}$ provided for the environment after a session-key-pointer-output message.

We note that strongly $\mathcal{F}_{\text{crypto}}$-secure protocols are α-$\mathcal{F}_{\text{crypto}}$-secure for all α, in particular for $\alpha = \emptyset$.

The following theorem states that if the plaintexts constructed by the secure channel protocols $\mathcal{P}_{\text{sc}}$ and $\mathcal{P}^{+}_{\text{sc}}$ do not interfere with the plaintexts encrypted by the key exchange protocol, then our criterion is sufficient for the secure channel protocols to realize the ideal secure channel functionalities.

Theorem 5. *Let $\mathcal{P}$ be an $\mathcal{F}_{\text{crypto}}$-KE protocol, α be a set of messages as above, and f be a message construction function such that $f(p_1, p_2, v, m) \notin \alpha$ for all bit strings p_1, p_2, v, m. If $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-secure with type* authenc-key*, then $\mathcal{P}_{\text{sc}}(f) \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$ and $\mathcal{P}^{+}_{\text{sc}}(f) \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}^{+}_{\text{sc}}$. (The leakage algorithm used in $\mathcal{F}_{\text{crypto}}$ is assumed to leak only the length of a message.)*

Proof. First, we consider $\mathcal{F}_{\text{sc}}$, see below for $\mathcal{F}^{+}_{\text{sc}}$. We define a simulator *Sim* for $\mathcal{F}_{\text{sc}}$ which is similar to the one defined in the proof of Theorem 4. Given a partnering function τ which exists because $\mathcal{P}$ is $\mathcal{F}_{\text{crypto}}$-secure *Sim* emulates a local copy of the system $\mathcal{S} = \mathcal{P}_{\text{sc}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ and establishes sessions using the partnering function τ. Message *send* request are forwarded to (the emulated) $\mathcal{S}$ and if $\mathcal{S}$ delivers a message, 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. As in the proof of Theorem 4, $\mathcal{F}_{\text{sc}}$ and $\mathcal{S}$ correspond on local sessions/instances, sessions, and corruption.

Upon corruption requests from the environment the systems $\mathcal{S}$ and $Sim \,|\, \mathcal{F}_{\text{sc}}$ do not differ, i.e., the view of the environment is the same. The systems $\mathcal{S}$ and $Sim \,|\, \mathcal{F}_{\text{sc}}$ potentially only differ i) upon a message *send* request for some message, say m, because in $\mathcal{S}$ the message $f(p', p, v, m)$ gets encrypted while in $Sim \,|\, \mathcal{F}_{\text{sc}}$ the message $f(p', p, v, 0^{|m|})$ gets encrypted or ii) upon delivery of messages to the parties because $\mathcal{S}$ outputs the decrypted received message while $Sim \,|\, \mathcal{F}_{\text{sc}}$ drops some messages and outputs the next message in the queue. Next, we show that the systems in fact do not differ.

ad i) Since $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-secure, every (uncorrupted) local session of a session uses the session key and this is marked unknown (in $\mathcal{F}_{\text{crypto}}$). Then, from the definition of $\mathcal{F}_{\text{crypto}}$ it follows that in $\mathcal{S}$ not $f(p', p, v, m)$ but its leakage is encrypted. Similar, in $Sim \,|\, \mathcal{F}_{\text{sc}}$ not $f(p', p, v, 0^{|m|})$ but its leakage is encrypted. Since these messages have the same length (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 c. Then both systems $\mathcal{P}$ and $Sim \,|\, \mathcal{F}_{\text{sc}}$ decrypt c using the session key pointer for the receiving local session. If the obtained plaintext is not of the form $f(p', p, v, m)$ for some $v \geq R_j$ and some message m then both systems discard this message. Otherwise, $\mathcal{S}$ delivers m 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 m', to p. We need to show that $m = m'$. Since $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-secure, the used session key is marked unknown (in $\mathcal{F}_{\text{crypto}}$) and of type authenc-key. Then, by definition of $\mathcal{F}_{\text{crypto}}$, only plaintexts are returned that have been previously encrypted under the same key using $\mathcal{F}_{\text{crypto}}$. Since $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-secure and $f(p', p, v, m) \notin \alpha$ for any r, some local session of this session must have encrypted this plaintext. They are all uncorrupted and, hence, it must have been the user for party p' when sending the v-th message to p. Note that R_j is exactly the number of messages p has already received from p' (in this session). 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 $m = m'$.

In the case of $\mathcal{F}^{+}_{\text{sc}}$ we can use the same simulator *Sim* except that it emulates $\mathcal{P}^{+}_{\text{sc}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$ instead of $\mathcal{P}_{\text{sc}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}}$. 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 $\mathcal{F}_{\text{sc}}$. □

5 Security Analysis of IEEE 802.11i

Using our results and methods developed in the previous sections, we now analyze two central protocols of WPA2-PSK (IEEE 802.11i) [28, 29], namely the 4-Way Handshake (4WHS) protocol and the CCM Protocol (CCMP). We

1. $A \to S$: $p_A, n_A, c_1[, \text{PMKID}]$
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. 1. The 4-Way Handshake protocol.

prove that 4WHS provides universally composable key exchange and that 4WHS with CCMP provides universally composable secure channels. Without $\mathcal{F}_{\text{crypto}}$, our modular approach, and our criteria, the proof would be considerably more complex and would involve non-trivial reduction proofs. In particular, due to $\mathcal{F}_{\text{crypto}}$, our proofs only require syntactic arguments and they illustrate that $\mathcal{F}_{\text{crypto}}$ can be used in an intuitive and easy way for the analysis of real-world security protocols.

5.1 The 4-Way Handshake Protocol

Description of the 4WHS Protocol. The 4-Way Handshake (4WHS) protocol consists of two roles, an authenticator A, e.g., an access point, and a supplicant S, e.g., a laptop, which share a Pairwise Master Key (PMK). The authenticator may communicate with several supplicants using the same PMK, which in WPA2-PSK is a pre-shared key (PSK). On an abstract level, the message exchange between an authenticator A and a supplicant S is shown in Figure 1, where p_A and p_S are the names (Media Access Control (MAC) addresses) of A and S, respectively, n_A and n_S are nonces generated by A and S, respectively, and $c_1, \ldots, c_4$ are pairwise distinct constants used to indicate different messages. From the PMK, A and S derive a Pairwise Transient Key PTK by computing $\text{PTK} = F(\text{PMK}, \text{“Pairwise key expansion”} \,\|\, \min(p_A, p_S) \,\|\, \max(p_A, p_S) \,\|\, \min(n_A, n_S) \,\|\, \max(n_A, n_S)))$, where F is an HMAC, which according to the IEEE 802.11i standard is assumed to be pseudo-random. The PTK is then split into the Key Confirmation Key (KCK), the Key Encryption Key (KEK), and the Temporary Key (TK), where TK is used in CCMP to establish a secure channel between A and S (see below). By $\text{MAC}_{\text{KCK}}(m)$ we denote the MAC of the message m under the key KCK. The first message of the 4WHS optionally includes $\text{PMKID} = F(\text{PMK}, \text{“PMK Name”} \| p_A \| p_S)$ to indicate the corresponding PMK.

Modeling the 4WHS Protocol. Modeling the 4WHS protocol as an $\mathcal{F}_{\text{crypto}}$-KE protocol is straightforward. We emphasize that, since $\mathcal{F}_{\text{crypto}}$ provides a low-level interface to basic cryptographic primitives with a very liberal use of tagging, our modeling of the 4WHS protocol, including message formats, the use of cryptographic primitives, and cryptographic assumptions, is quite close to the actual standard. We note that in our modeling of 4WHS parties may *not* play both the role of an authenticator and a supplicant with the same pre-shared key. Otherwise, 4WHS would be insecure. Indeed, a reflection attack would be possible [25], and our security proofs would fail.

The $\mathcal{F}_{\text{crypto}}$-KE protocol 4WHS consists of two roles: $\text{4WHS} = !M_A \mid !M_S$. As defined in Section 4.1, there can be multiple instances of M_A and M_S and each instance is addressed by a PID and a LSID.

A natural way to model the pre-shared key PSK would be the following: At first, every instance (of M_A or M_S) establishes a pre-shared key (in $\mathcal{F}_{\text{crypto}}$) using the PID of the authenticator as the name of the pre-shared key. This way, all supplicants talking to the same authenticator use the same pre-shared key. But this modeling yields that an authenticator, say A, uses the same PSK in every session. For example, A cannot change the PSK. Also, different authenticator would necessarily use different pre-shared keys. Therefore, we model the pre-shared key PSK as follows: We allow the environment to decide which instance uses which pre-shared key. To do so, we require that every LSID is a pair $(lsid', psk\text{-}name)$ of the actual LSID $lsid'$ and the name of the pre-shared key *psk-name* this instance uses. Whenever an instance (of M_A or M_S) is first activated, it checks if its LSID is of the form $(lsid', psk\text{-}name)$ and obtains a pointer to the pre-shared key with name *psk-name* using $\mathcal{F}_{\text{crypto}}$. (If this check fails, the instance terminates.)

After having received a *session-start* message and established the PSK (as described above), every instance of M_A and M_S executes the 4WHS protocol as expected using $\mathcal{F}_{\text{crypto}}$ to generate nonces, derive keys (the key KCK is of type mac-key and the key TK is of type authenc-key), and compute and verify MACs. Finally, the pointer to TK (later used to establish a secure channel) is output as the session key pointer.

As mentioned above, a simple reflection attack on the 4WHS protocol is possible if there is a party playing both the role of an authenticator and a supplicant using the same pre-shared key, see [25] and our proof of Theorem 6. To

prevent such an unusual environment, we use a (polynomial-time computable) predicate for the PIDs which separates PIDs for authenticators from PIDs for supplicants. Every instance of M_A and M_S first checks (using the predicate) whether its own PID is allowed for the role of an authenticator or supplicant, respectively. (If this check fails, the instance terminates.)

Recall that in the 4WHS protocol, the keys KCK, KEK, and TK are derived from PMK by first deriving PTK and then splitting it, as described above. We do not model this directly because we do not model bit string operations like splitting of keys in $\mathcal{F}_{\text{crypto}}$. Instead of deriving KCK||KEK||TK = PTK = F(PMK, *seed*) we derive every key separately KCK = F(PMK, *seed*||0), KEK = F(PMK, *seed*||1), and TK = F(PMK, *seed*||2). Note that it is easy to show that F is a pseudo-random function if and only if F' is a pseudo-random function where F' is defined by $F'(k, s||i) = \pi_i(F(k, s))$ for all k, s and $i \in \{0, 1, 2\}$ where π_0, π_1, π_2 partitions the input into disjoint parts. Hence, our modeling is faithful.

We model corruption as follows. The adversary can corrupt an instance of M_A and M_S by sending a special corrupt message to it. This has to be the first message this instance receives from the adversary. A corrupted instance forwards all messages from the user to the adversary and vice versa. (See Section 4.2 for the discussion of arbitrary network input.) Furthermore, it allows the adversary to send requests to $\mathcal{F}_{\text{crypto}}$ (the responses are forwarded to the adversary) in the name of the user.

If the instance is corrupted, all pointers of this user to pre-shared keys and other symmetric keys this user (on demand of the adversary) has created in $\mathcal{F}_{\text{crypto}}$ should be corrupted (in $\mathcal{F}_{\text{crypto}}$) as well. We therefore require that the adversary corrupts all keys a corrupted instance creates using $\mathcal{F}_{\text{crypto}}$. A corrupted instance always checks (by asking $\mathcal{F}_{\text{crypto}}$) if its keys created in $\mathcal{F}_{\text{crypto}}$ indeed have been corrupted by the adversary and terminates if they have not been corrupted. Note that since keys in $\mathcal{F}_{\text{crypto}}$ of a corrupted instance are known, it is not a problem if the adversary generates key cycles or causes the commitment problem with those keys. Conversely, uncorrupted instances always check that the pre-shared key PSK, and the nonce, n_A or n_S, they have created using $\mathcal{F}_{\text{crypto}}$ are uncorrupted at the time of their creation. We note that this modeling implies that a corrupted local session never obtains a pointer to an unknown key. Hence, the adversary can use $\mathcal{F}_{\text{crypto}}$ (through the corrupted local session) only with known keys. Of course, since the adversary provides all algorithms and keys used in $\mathcal{F}_{\text{crypto}}$, she can encrypt and decrypt messages on her own, i.e., outside of $\mathcal{F}_{\text{crypto}}$.

In the literature, (static) corruption is often modeled on a per party basis, i.e., if a party is corrupted, then all its keys are corrupted and the adversary is in full control of that party. We note that this is a special case of our modeling of corruption because the adversary can decide to corrupt all keys and local sessions of a corrupted party.

The messages $c_1, \ldots, c_4$ consist of several fields. For simplicity of the analysis, our modeling ignores some of these fields or fixes them to be constants, as described next (conceptually, it would not be a problem to model these fields precisely): The messages $c_1, \ldots, c_4$ contain counters used for re-keying, which we ignore. The information contained in c_2 and c_3 for negotiating cipher suites and avoiding version rollback attacks is modeled as constants. Finally, we ignore an optional field in c_3 for multicast communication.

Next, we provide more insight into how message formats of the 4WHS protocol are modeled on the bit level. All four 4WHS messages are *EAPOL-Key frames*, see Figure 2, which are defined in the IEEE 802.11 standard, e.g., the field *Key Nonce* contains the nonces of the authenticator or supplicant, respectively, and the field *Key MIC* contains the MACs. Since every instance (of M_A and M_S) knows the bit strings of all parts of an EAPOL-Key frame, e.g., the nonces and MACs, it can easily construct precisely these EAPOL-Key frames. We note that, since our cryptographic model is asymptotic, the length of e.g. keys and nonces depends on a security parameter η, and hence, we need to generalize EAPOL-Key frames so that the length of some fields depends on η.

Security Analysis. We first show that 4WHS is strongly $\mathcal{F}_{\text{crypto}}$-secure.

Theorem 6. *The protocol* 4WHS *is strongly* $\mathcal{F}_{\text{crypto}}$*-secure with type* authenc-key.

Proof. First, we define a partnering function τ for 4WHS: Two instances are defined to form a session if a) they have different roles, namely A and S, respectively, b) they are both uncorrupted, c) the party names of the desired partner correspond to each other, d) they use the same pre-shared key, e) the values of the nonces correspond to each other, and f) one of them has already output a session key pointer. Because $\mathcal{F}_{\text{crypto}}$ guarantees that (uncorrupted) nonces are unique for every instance, there are at most two such instances, and hence, it is easy to see that τ is a valid partnering function for 4WHS.

Protocol Version – 1 octet	Packet Type – 1 octet	Packet Body Length – 2 octets
Descriptor Type – 1 octet		
Key Information – 2 octets		Key Length – 2 octets
Key Replay Counter – 8 octets		
Key Nonce – 32 octets		
EAPOL-Key IV – 16 octets		
Key RSC – 8 octets		
Reserved – 8 octets		
Key MIC – 16 octets		
Key Data Length – 2 octets		Key Data – n octets

Fig. 2. EAPOL-Key frame [28, Figure 8-23]

It remains to show that 4WHS is strongly $\mathcal{F}_{\text{crypto}}$-secure w.r.t. τ and every environment $\mathcal{E}$ of $4\text{WHS} \mid \mathcal{F}_{\text{crypto}}$: Let ρ be a run of $\mathcal{E} \mid 4\text{WHS} \mid \mathcal{F}_{\text{crypto}}$ and let $(p, lsid, r)$ be some uncorrupted instance (i.e., an instance of M_r) in ρ which has output a session key pointer to a key, say k, in $\mathcal{F}_{\text{crypto}}$, and which established the pre-shared key PSK and derived KCK and TK from it in $\mathcal{F}_{\text{crypto}}$.

First, we observe that, by our corruption model, since $(p, lsid, r)$ is uncorrupted, PSK is uncorrupted (in $\mathcal{F}_{\text{crypto}}$). Also, every other instance that established PSK must be uncorrupted as well since keys created by corrupted instances are required to be corrupted. In uncorrupted instances, PSK is only used to derive keys, hence, PSK is always marked unknown in $\mathcal{F}_{\text{crypto}}$. In particular, no corrupted local session has a pointer to PSK. Now, by definition of $\mathcal{F}_{\text{crypto}}$, KCK and TK can only be derived by instances that have a pointer to PSK, leaving only uncorrupted instances. Moreover, again by $\mathcal{F}_{\text{crypto}}$, these uncorrupted instances have to use the same seed s as $(p, lsid, r)$, which contains the party names, p and p' say, and two nonces. Since uncorrupted nonces generated by $\mathcal{F}_{\text{crypto}}$ are guaranteed to be unique, by the construction of s, it follows that besides $(p, lsid, r)$ at most one other (uncorrupted) instance $(p', lsid', r')$, for some p', $lsid'$, and r', uses s, and hence, has a pointer to KCK and TK by derivation. By the definition of the protocol, uncorrupted instances only use KCK for MACing and TK is at most used after being output in a *session-key-pointer-output* message, but then TK may not be encrypted or retrieved. By definition of $\mathcal{F}_{\text{crypto}}$, it follows that KCK and TK are always marked unknown in $\mathcal{F}_{\text{crypto}}$ and only $(p, lsid, r)$ and, if present, $(p', lsid', r')$ have pointers to KCK and TK.

We now show that $(p', lsid', r')$ exists and that $(p, lsid, r)$ and $(p', lsid', r')$ belong to the same session (according to τ), which implies i) of Definition 2: We assume that $r = A$; the proof for $r = S$ is similar. The instance $(p, lsid, r)$ verified a MAC in a message of the form $p', n'', c_2, \text{MAC}_{\text{KCK}}(n'', c_2)$. Since $r = A$ and the constants c_2 and c_3 are distinct, $(p, lsid, r)$ has not created such a MAC. By definition of $\mathcal{F}_{\text{crypto}}$, $\text{MAC}_{\text{KCK}}(n'', c_2)$ can only have been created by some instance that has a pointer to KCK, which must be the (uncorrupted) instance $(p', lsid', r')$ from above. It follows that $r' = S$ since an uncorrupted instance with $r' = A$ would not create a MAC of such a form. By our assumption that a party does not play both the role of A and S with the same pre-shared key, it follows that $p' \neq p$. (Our assumption, and the implied fact, $p' \neq p$, is crucial; without it the proof would fail and in fact a reflection attack would be possible [25].) We can now show that $(p, lsid, r)$ and $(p', lsid', r')$ belong to the same session according to τ: We already know that conditions a), b), d), and f) for τ (as defined above) are satisfied. Since $p \neq p'$, it follows that the intended partner of $(p', lsid', r')$ is p, since, by definition of $\mathcal{F}_{\text{crypto}}$ and KCK, otherwise $(p', lsid', r')$ could not have derived KCK. So c) is satisfied. (Without our assumption mentioned above, this could not be concluded.) Similarly, condition e) is satisfied since otherwise the two instances would not have derived the same KCK.

We already know that TK (= k) is marked unknown in $\mathcal{F}_{\text{crypto}}$. It is of type authenc-key since, by definition of the protocol, it has been derived as a key of this type. So ii) of Definition 2 follows.

We also know that only $(p, lsid, r)$ and $(p', lsid', r')$ have a pointer to TK in $\mathcal{F}_{\text{crypto}}$. Hence, iv) of Definition 2 follows. Since both instances are uncorrupted, by the definition of the protocol, iii) follows as well. □

Trivially, $\widehat{4\text{WHS}}$ (recall that $\widehat{4\text{WHS}}$ outputs the session key instead of a pointer to it) is a standard protocol (as defined in Section 3) because it never encrypts an unknown key after it has been first used. We note that, by our modeling

of corruption, corrupted local sessions do not have pointers to unknown keys and, hence, even upon corruption, no unknown key is encrypted after it has been first used. In fact, no key is encrypted at all by uncorrupted local sessions. It is easy to see that $4\widehat{\text{WHS}}$ is used-order respecting and non-committing. Using Theorem 4 and 6 we immediately obtain that $4\widehat{\text{WHS}}$ is a universally composable key exchange protocol.

Corollary 2. $4\widehat{\text{WHS}} \,|\, \mathcal{P}_{\text{crypto}} \leq \mathcal{F}_{\text{ke}}$.

5.2 The CCM Protocol

Description of the CCM Protocol. WPA2-PSK uses CCMP with the Temporal Key (TK) which has been exchanged by 4WHS to establish a secure channel between the authenticator and the supplicant. In CCMP, both parties maintain and use counters for the messages they send and received just like $\mathcal{P}_{\text{sc}}$. Encryption is done using the block cipher AES (with the key TK) in the CCM (Counter with CBC-MAC) encryption mode.

Modeling the CCM Protocol. We model CCMP as the protocol CCMP $= \mathcal{P}_{\text{sc}}(f)$ using a message construction function f that closely models the message formats of CCMP. More precisely, we define the function f such that the plaintexts $f(p, p', v, m)$ (recall that p/p' is the PID of the sender/receiver, v is the value of a counter, and m the actual message that is sent over the secure channel) are well-tagged with type data. Furthermore, we define tagging in $\mathcal{F}_{\text{crypto}}$ such that $f(p, p', v, m)$ precisely models the format of plaintexts in the CCMP protocol, see [28, Section 8.3.3] for a definition of these message formats. The CCM encryption mode is a secure authenticated encryption mode [30] and, hence, can be modeled as an authenticated encryption scheme. Altogether, the protocol CCMP $|$ 4WHS $|\, \mathcal{P}_{\text{crypto}}$ is quite close to the actual cryptographic implementation of CCMP using 4WHS for key exchange.

Security Analysis. By Theorem 5 and 6 we obtain that CCMP using 4WHS and the ideal crypto functionality is a secure channel protocol.

Corollary 3. CCMP $|$ 4WHS $|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$.

It is easy to see that CCMP $|$ 4WHS is a standard protocol (as defined in Section 3), hence, it is used-order respecting and non-committing. From Corollary 1 and 3 (and the composition theorems) we obtain that WPA2-PSK modeled as the protocol CCMP $|$ 4WHS $|\, \mathcal{P}_{\text{crypto}}$ is a secure channel protocol.

Corollary 4. CCMP $|$ 4WHS $|\, \mathcal{P}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$.

As mentioned in the introduction, it has been recently shown that 4WHS with TKIP (instead of CCMP) is insecure [42, 38]. The attacks exploit that TKIP uses the stream cipher RC4 in an encryption mode which does not yield a secure authenticated encryption scheme.

6 Related Work

Backes et al. (see, e.g., [4]) proposed a Dolev-Yao style cryptographic library. The main purpose of the library is to provide a Dolev-Yao style abstraction to the user, in the spirit of computational soundness results [36, 22, 2, 34]. In contrast, our functionality provides a much lower-level idealization, aiming at wide applicability and faithful treatment of cryptographic primitives. More specifically, unlike $\mathcal{F}_{\text{crypto}}$, based on the Dolev-Yao library only those protocols can be analyzed which merely use operations provided by the library (since the user, except for payload data, only gets his/her hand on pointers to Dolev-Yao terms in the library, rather than on the actual bit strings, internally everything is represented as terms too) and these protocols can only be shown to be secure w.r.t. non-standard encryption schemes (since, e.g., extra randomness and tagging with key identifiers is assumed for encryption schemes) and assuming specific message formats (all types of messages—nonces, ciphertexts, pairs of messages etc.—, are tagged in the realization). While the Dolev-Yao library considers symmetric encryption (key derivation is not considered at all) [4], it is an open problem whether there is a reasonable realization; the original proof of the realization of the crypto library in [4] is flawed, as examples presented in [21] illustrate (see also [35]).

Our criteria for secure key exchange and secure channel protocols presented in Section 4 are related to the concept of secretive protocols proposed by Roy et al. [40, 41] (see also [35]). However, unlike our criteria, which can be checked based on information-theoretic/syntactical arguments, checking whether a protocol is secretive requires involved cryptographic reduction proofs. Also, Roy et al. do not prove implications for *composable* security and they do not consider secure channels.

The only work we are aware of that attempts to perform a cryptographic analysis of the 4-Way Handshake protocol of IEEE 802.11i is [43]; secure channels are not considered. However, this work is quite preliminary: The security assumptions and theorems are not formulated precisely and no security proofs or proof sketches are available. In He et al. [26], the first *symbolic* analysis of IEEE 802.11i has been presented, based on their Protocol Composition Logic (PCL). There are only a few other papers on the analysis of real-world protocols that involve key derivation: (fragments of) TLS were analyzed in [23, 37, 11], assuming session identifiers in ciphertexts [23] or the random oracle for key derivation [37, 11]. Cryptographic analysis of Kerberos was carried out in [1, 13], where in [13] key derivation is modeled by pseudo-random functions within CryptoVerif. However, this analysis considers more abstract message formats and does not yield composable security guarantees.

References

1. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically Sound Security Proofs for Basic and Public-Key Kerberos. In D. Gollmann, J. Meier, and A. Sabelfeld, editors, *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security*, volume 4189 of *Lecture Notes in Computer Science*, pages 362–383. Springer, 2006.
2. M. Backes, M. Dürmuth, and R. Küsters. On Simulatability Soundness and Mapping Soundness of Symbolic Cryptography. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2007)*, volume 4855 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2007.
3. M. Backes and D. Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. In *Information Security, 7th International Conference, ISC 2004, Proceedings*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004.
4. M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *17th IEEE Computer Security Foundations Workshop (CSFW-17 2004)*, pages 204–218. IEEE Computer Society, 2004.
5. M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent Message Security under Active Attacks - BRSIM/UC-Soundness of Symbolic Encryption with Key Cycles. *Journal of Computer Security (JCS)*, 2008.
6. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In H. Krawczyk, editor, *Advances in Cryptology, 18th Annual International Cryptology Conference (CRYPTO 1998)*, volume 1462 of *Lecture Notes in Computer Science*, pages 549–570. Springer, 1998.
7. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.
8. 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.
9. 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.
10. 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.
11. K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Cryptographically Verified Implementations for TLS. In P. Ning, P. F. Syverson, and S. Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security (CCS 2008)*, pages 459–468. ACM, 2008.
12. J. Black, P. Rogaway, and T. Shrimpton. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In K. Nyberg and H. M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.
13. B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally Sound Mechanized Proofs for Basic and Public-key Kerberos. In M. Abe and V. D. Gligor, editors, *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security (ASIACCS 2008)*, pages 87–99. ACM, 2008.

14. 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.
15. R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW-17 2004)*, pages 219–233. IEEE Computer Society, 2004.
16. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical report, Cryptology ePrint Archive, December 2005. Online available at `http://eprint.iacr.org/2000/067.ps`.
17. 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.
18. R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.
19. 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.
20. 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.
21. H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. Technical Report INRIA Research Report RR-6508, INRIA, 2008. Available from `http://www.loria.fr/~cortier/Papiers/CCS08-report.pdf`.
22. V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2006.
23. 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.
24. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
25. 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.
26. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A Modular Correctness Proof of IEEE 802.11i and TLS. In V. Atluri, C. Meadows, and A. Juels, editors, *12th ACM Conference on Computer and Communications Security (CCS 2005)*, pages 2–15. ACM, 2005.
27. 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/`.
28. 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.
29. 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.
30. J. Jonsson. On the Security of CTR + CBC-MAC. In K. Nyberg and H. M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop (SAC 2002)*, volume 2595 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 2002.
31. 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.
32. 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.
33. 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.
34. R. Küsters and M. Tuengerthal. Computational Soundness for Key Exchange Protocols with Symmetric Encryption. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 91–100. ACM Press, 2009.

35. 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.
36. D. Micciancio and B. Warinschi. Soundness of Formal Encryption in the Presence of Active Adversaries. In M. Naor, editor, *First Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.
37. P. Morrissey, N. P. Smart, and B. Warinschi. A Modular Security Analysis of the TLS Handshake Protocol. In J. Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security*, volume 5350 of *Lecture Notes in Computer Science*, pages 55–73. Springer, 2008.
38. T. Ohigashi and M. Morii. A Practical Message Falsification Attack on WPA. In *Proceedings of the Fourth Joint Workshop on Information Security (JWIS 2009)*, 2009.
39. B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Security and Privacy*, pages 184–201. IEEE Computer Society Press, 2001.
40. A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Proofs of Computational Secrecy. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium On Research In Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2007.
41. A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Trace Properties Imply Computational Security. In *Proceedings of the 7th International Workshop on Issues in the Theory of Security (WITS'07)*, 2007.
42. 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.
43. F. Zhang, J. Ma, and S. Moon. The Security Proof of a 4-Way Handshake Protocol in IEEE 802.11i. In Y. Hao, J. Liu, Y. Wang, Y. ming Cheung, H. Yin, L. Jiao, J. Ma, and Y.-C. Jiao, editors, *Computational Intelligence and Security, International Conference (CIS 2005)*, volume 3802 of *Lecture Notes in Computer Science*, pages 488–493. Springer, 2005.

A Security Definitions for Cryptographic Primitives

In this section, we present the standard cryptographic schemes and security notions which we use to realize our ideal crypto functionality.

A.1 Symmetric Encryption

In this section, we recall the notion of [6, 7] for symmetric encryption schemes.

Definition 4. *A* symmetric encryption scheme $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ *with plaintext domain* $\mathrm{dom}(\Sigma) \subseteq \{0,1\}^*$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter* η *and returns a key* $\mathrm{gen}(1^\eta)$*. The probabilistic encryption algorithm* enc *expects a key* k *and a plaintext* m *and returns a ciphertext* $\mathrm{enc}(k,m) \in \{0,1\}^*$ *or* $\mathrm{enc}(k,m) = \perp$ *(where* $\perp \notin \{0,1\}^*$ *is a special error symbol) if encryption fails. The deterministic decryption algorithm* dec *expects a key* k *and a ciphertext* $c \in \{0,1\}^*$ *and returns the plaintext* $\mathrm{dec}(k,c) \in \{0,1\}^*$ *or* $\mathrm{dec}(k,c) = \perp$ *if decryption fails.*

It is required that for every security parameter η *and key* k *generated by* $\mathrm{gen}(1^\eta)$ *it holds that i)* $\mathrm{enc}(k,m) = \perp$ *if and only if* $m \notin \mathrm{dom}(\Sigma)$ *and ii)* $\mathrm{dec}(k, \mathrm{enc}(k,m)) = m$ *for every plaintext* $m \in \mathrm{dom}(\Sigma)$*.*

We assume that every encryption scheme is associated with a polynomial q that bounds the runtime of the algorithms and the length of there description in some standard encoding. We say that Σ is *q-bounded*. For all symmetric encryption schemes considered in this paper, we assume that the key generation algorithm chooses keys uniformly at random from $\{0,1\}^\eta$.

We define $\mathrm{LR}(m_0, m_1, b) = m_b$ for every $b \in \{0,1\}$ and $m_0, m_1 \in \{0,1\}^*$ of same length. If m_0 and m_1 are not of same length, we define $\mathrm{LR}(m_0, m_1, b) = \perp$.

Definition 5 (IND-CPA secure). *A symmetric encryption scheme* Σ *is called* IND-CPA secure *if for every probabilistic polynomial-time algorithm* $A^{O(\cdot,\cdot)}$ *with access to an oracle* O*, the* IND-CPA advantage *of* A *with respect to* Σ

$$\mathrm{Adv}^{\text{ind-cpa}}_{A,\Sigma}(1^\eta, a) = \Pr[k \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,1))}(1^\eta, a) = 1] - \Pr[k \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,0))}(1^\eta, a) = 1]$$

is negligible as a function in η *and* a*.*[2]

[2] A function $f\colon \{1\}^* \times \{0,1\}^* \to \mathbf{R}_{\geq 0}$ is called *negligible* if for all polynomials p and q there exists $\eta_0 \in \mathbf{N}$ such that for all $\eta > \eta_0$ and all bit strings $a \in \{0,1\}^*$ with length $|a| \leq q(\eta)$ we have that $f(1^\eta, a) \leq 1/p(\eta)$.

Definition 6 (IND-CCA secure). *A symmetric encryption scheme Σ is called* IND-CCA secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot,\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2 which never queries O_2 with a bit string returned by O_1, the* IND-CCA advantage *of A with respect to Σ*

$$\mathrm{Adv}^{\text{ind-cca}}_{A,\Sigma}(1^\eta, a) = \Pr[k \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,1)),\mathrm{dec}(k,\cdot)}(1^\eta, a) = 1] - \Pr[k \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,0)),\mathrm{dec}(k,\cdot)}(1^\eta, a) = 1]$$

is negligible as a function in η and a.

Definition 7 (INT-CTXT secure). *A symmetric encryption scheme Σ is* INT-CTXT secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2, the* INT-CTXT advantage *of A with respect to Σ*

$$\mathrm{Adv}^{\text{int-ctxt}}_{A,\Sigma}(1^\eta, a) = \Pr[k \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k,\cdot),\mathrm{dec}(k,\cdot)}(1^\eta, a) \text{ makes a query } c \text{ to } \mathrm{dec}(k,\cdot) \text{ such that } \mathrm{dec}(k,c) \neq \perp \text{ and } c \text{ was not previously returned by } \mathrm{enc}(k,\cdot)]$$

is negligible as a function in η and a.

A.2 Public-Key Encryption

In this section, we recall the notion of [6] for public-key encryption schemes.

Definition 8. *A* public-key encryption scheme $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ *with plaintext domain* $\mathrm{dom}(\Sigma) \subseteq \{0,1\}^*$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter η and returns a pair of keys (k_d, k_e), the secret (or decryption) key k_d and the public (or encryption) key k_e. The probabilistic encryption algorithm* enc *expects a public key k_e and a plaintext m and returns a ciphertext* $\mathrm{enc}(k_e, m) \in \{0,1\}^*$ *or* $\mathrm{enc}(k_e, m) = \perp$ *(where $\perp \notin \{0,1\}^*$ is a special error symbol) if encryption fails. The deterministic decryption algorithm* dec *expects a private key k_d and a ciphertext $c \in \{0,1\}^*$ and returns the plaintext* $\mathrm{dec}(k_d, c) \in \{0,1\}^*$ *or* $\mathrm{dec}(k_d, c) = \perp$ *if decryption fails.*

It is required that for every security parameter η and key pair (k_e, k_d) generated by $\mathrm{gen}(1^\eta)$ *the following holds: i)* $\mathrm{enc}(k_e, m) = \perp$ *if and only if $m \notin \mathrm{dom}(\Sigma)$ and ii)* $\mathrm{dec}(k_d, \mathrm{enc}(k_e, m)) = m$ *for every plaintext $m \in \mathrm{dom}(\Sigma)$.*

We assume that every encryption scheme is associated with a polynomial q that bounds the runtime of the algorithms and the length of there description in some standard encoding. We say that Σ is *q-bounded*.

As above, we define $\mathrm{LR}(m_0, m_1, b) = m_b$ for every $b \in \{0,1\}$ and $m_0, m_1 \in \{0,1\}^*$ of same length. If m_0 and m_1 are not of same length, we define $\mathrm{LR}(m_0, m_1, b) = \perp$.

Definition 9 (IND-CCA secure). *A public-key encryption scheme Σ is called* IND-CCA secure *if for every probabilistic polynomial-time algorithm $A^{O_1(\cdot,\cdot),O_2(\cdot)}$ with access to two oracles O_1, O_2 which never queries O_2 with a bit string returned by O_1, the* IND-CCA advantage *of A with respect to Σ*

$$\mathrm{Adv}^{\text{ind-cca}}_{A,\Sigma}(1^\eta, a) = \Pr[(k_e, k_d) \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k_e,\mathrm{LR}(\cdot,\cdot,1)),\mathrm{dec}(k_d,\cdot)}(1^\eta, a, k_e) = 1] - \Pr[(k_e, k_d) \leftarrow \mathrm{gen}(1^\eta) : A^{\mathrm{enc}(k_e,\mathrm{LR}(\cdot,\cdot,0)),\mathrm{dec}(k_d,\cdot)}(1^\eta, a, k_e) = 1]$$

is negligible as a function in η and a.

A.3 Message Authentication Codes

In this section, we recall the notions of [24] for message authentication codes.

Definition 10. *A* message authentication code (MAC) scheme $\Sigma = (\text{gen}, \text{mac}, \text{mac-verify})$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter* η *and returns a key* $\text{gen}(1^\eta)$*. The (possibly) probabilistic MAC algorithm* mac *expects a key* k *and a message* m *and returns a message authentication code* $\text{mac}(k, m)$*. The deterministic verification algorithm* mac-verify *expects a key* k*, a message* m*, and a message authentication code* σ *and returns* $\text{mac-verify}(k, m, \sigma) \in \{\mathsf{false}, \mathsf{true}\}$.

It is required that for every $\eta \in \mathbf{N}$*, key* k *generated by* $\text{gen}(1^\eta)$*, and message* $m \in \{0,1\}^*$ *it holds that* $\text{mac-verify}(k, m, \text{mac}(k, m)) = \mathsf{true}$.

For all MAC schemes considered in this paper, we assume that the key generation algorithm chooses keys uniformly at random from $\{0,1\}^\eta$.

Definition 11. *A MAC scheme* Σ *is called* existentially unforgeable under adaptive chosen-message attacks (UF-CMA secure) *if for every probabilistic polynomial-time algorithm* $A^{O_1(\cdot),O_2(\cdot,\cdot)}$ *with access to two oracles* O_1, O_2*, the* advantage *of* A *with respect to* Σ

$$\text{Adv}^{\text{mac}}_{A,\Sigma}(1^\eta, a) = \Pr[k \leftarrow \text{gen}(1^\eta), (m, \sigma) \leftarrow A^{\text{mac}(k,\cdot),\text{mac-verify}(k,\cdot,\cdot)}(1^\eta, a) : \text{mac-verify}(k, m, \sigma) = \mathsf{true} \text{ and } A \text{ has not previously called } \text{mac}(k, m)]$$

is negligible as a function in η *and* a.

A.4 Digital Signatures

In this section, we recall the notions of [24] for digital signature schemes.

Definition 12. *A* (digital) signature scheme $\Sigma = (\text{gen}, \text{sig}, \text{sig-verify})$ *consists of three polynomial-time algorithms. The probabilistic key generation algorithm* gen *expects a security parameter* η *and returns a pair of keys* (k_s, k_v)*, the secret (or signing) key* k_s *and the public (or verification) key* k_v*. The (possibly) probabilistic signing algorithm* sig *expects a private key* k_s *and a message* $m \in \{0,1\}^*$ *and returns a signature* $\text{sig}(k_s, m)$*. The deterministic verification algorithm* sig-verify *expects a public key* k_v*, a message* $m \in \{0,1\}^*$*, and a signature* $\sigma \in \{0,1\}^*$ *and returns* $\text{sig-verify}(k_v, m, \sigma) \in \{\mathsf{false}, \mathsf{true}\}$.

It is required that for every security parameter $\eta \in \mathbf{N}$*, key pair* (k_s, k_v) *generated by* $\text{gen}(1^\eta)$*, and message* $m \in \{0,1\}^*$ *it holds that* $\text{sig-verify}(k_v, m, \text{sig}(k_s, m)) = \mathsf{true}$.

Definition 13. *A digital signature scheme* Σ *is called* existentially unforgeable under adaptive chosen-message attacks (UF-CMA secure) *if for every probabilistic polynomial-time algorithm* $A^{O(\cdot)}$ *with access to a signing oracle* O*, the* advantage *of* A *with respect to* Σ

$$\text{Adv}^{\text{sig}}_{A,\Sigma}(1^\eta, a) = \Pr[(k_s, k_v) \leftarrow \text{gen}(1^\eta), (m, \sigma) \leftarrow A^{\text{sig}(k_s,\cdot)}(1^\eta, a, k_v) : \text{sig-verify}(k_v, m, \sigma) = \mathsf{true} \text{ and } A \text{ has not previously called } \text{sig}(k_s, m)]$$

is negligible as a function in η *and* a.

A.5 Pseudo-Random Functions

Let $h\colon \{0,1\}^* \to \{0,1\}^\eta$ be the following probabilistic, stateful algorithm. It maintains a set H which is initially empty. Upon input $s \in \{0,1\}^*$, h returns x if there exists an x such that $(x, s) \in H$. Otherwise, h chooses x uniformly at random from $\{0,1\}^\eta$, adds (x, s) to H, and returns x.

Definition 14. *A family of functions* $F = \{F_\eta\}_{\eta \in \mathbf{N}}$ *with* $F_\eta\colon \{0,1\}^\eta \times \{0,1\}^* \to \{0,1\}^\eta$ *for all* $\eta \in \mathbf{N}$ *is called a* pseudo-random function family *if it is efficiently computable and for any probabilistic polynomial-time adversary* $A^{O(\cdot)}$ *with access to an oracle* O*, the* advantage *of* A *with respect to* F

$$\text{Adv}^{\text{prf}}_{A,F}(1^\eta, a) = \Pr\left[k \xleftarrow{R} \{0,1\}^\eta : A^{F_\eta(k,\cdot)}(1^\eta, a) = 1\right] - \Pr\left[A^{h(\cdot)}(1^\eta, a) = 1\right]$$

is negligible as a function in η *and* a.

Oracle$(b, \Sigma_{\text{authenc}}, \Sigma_{\text{unauthenc}}, \Sigma_{\text{mac}}, F, L)$

Tapes: enriching I/O input tape $T^{\text{in}}_{\text{oracle}}$, I/O output tape $T^{\text{out}}_{\text{oracle}}$
State: The state of Oracle consists of the following variables: **state** $\in$ {init, authenc, unauthenc, mac, prf} (initially init), $\mathbf{k} \in \{0,1\}^* \cup \{\bot\}$ (initially $\bot$), **Table** $\subseteq \{0,1\}^*$ (initially $\emptyset$)
CheckAddress: Every message on every tape is always accepted.
Compute:

1. Upon receiving (Init, t) from $T^{\text{in}}_{\text{oracle}}$ where **state** = init and $t \in$ {authenc, unauthenc, mac, prf} do: Set **state** := t. Choose $k \leftarrow^{\text{R}} \{0,1\}^\eta$ and set $\mathbf{k} := k$. Send (Ack) to $T^{\text{out}}_{\text{oracle}}$.
2. Upon receiving (Enc, x) from $T^{\text{in}}_{\text{oracle}}$ where **state** $\in$ {authenc, unauthenc} and $m \in \text{dom}(L)_\eta$ do: If b = real, then compute $y \leftarrow \text{enc}(k, x)$ (where enc is the encryption algorithm of Σ_{authenc} or $\Sigma_{\text{unauthenc}}$ depending on t) and send y to T_{oracle}. Otherwise, compute $\overline{x} \leftarrow L(1^\eta, x)$ and $z \leftarrow \text{enc}(k, \overline{x})$. Then, add (x, y) to **Table** and send y to $T^{\text{out}}_{\text{oracle}}$.
3. Upon receiving (Dec, y) from $T^{\text{in}}_{\text{oracle}}$ where **state** $\in$ {authenc, unauthenc} and $y \in \{0,1\}^*$ do: If b = real, then compute $x := \text{dec}(k, y)$ (where dec is the decryption algorithm of Σ_{authenc} or $\Sigma_{\text{unauthenc}}$ depending on t) and send x to $T^{\text{out}}_{\text{oracle}}$. Otherwise, compute
$$x := \begin{cases} x' & \text{if } \exists! x' \colon (m', c) \in \mathit{Table} \quad \text{(exists unique } x') \\ \text{dec}(k, y) & \text{if } \textbf{state} = \text{unauthenc and } \forall x' \colon (x', y) \notin \textbf{Table} \\ \bot & \text{otherwise} \end{cases}$$
and send x to $T^{\text{out}}_{\text{oracle}}$.
4. Upon receiving (Mac, x) from $T^{\text{in}}_{\text{oracle}}$ where **state** = mac do: Compute $\sigma \leftarrow \text{mac}(k, x)$, add x to **Table**, and send σ to $T^{\text{out}}_{\text{oracle}}$.
5. Upon receiving (MacVerify, x, σ) from $T^{\text{in}}_{\text{oracle}}$ where **state** = mac do: If b = ideal, mac-verify(k, x, σ) = true, and $x \notin$ **Table**, then send $\bot$ to $T^{\text{out}}_{\text{oracle}}$. (This indicates forgery.) Otherwise, send the result of mac-verify(k, x, σ) to $T^{\text{out}}_{\text{oracle}}$.
6. Upon receiving (PRF, s) from $T^{\text{in}}_{\text{oracle}}$ where **state** = prf and $s \in \{0,1\}^*$ do: If b = real, then compute $x := F_\eta(k, s)$ and send x to $T^{\text{out}}_{\text{oracle}}$. Otherwise, if $\exists x \colon (s, x) \in$ **Table**, then send x to $T^{\text{out}}_{\text{oracle}}$. (Note that if such an x exists, it is unique.) Otherwise, choose $x \leftarrow^{\text{R}} \{0,1\}^\eta$, add (s, x) to **Table**, and send x to $T^{\text{out}}_{\text{oracle}}$.

Fig. 3. The IITM Oracle is parameterized by a variable $b \in$ {real, ideal}, two symmetric encryption schemes Σ_{authenc} and $\Sigma_{\text{unauthenc}}$, a message authentication scheme Σ_{mac}, a pseudo-random function family F, and a leakage algorithm L.

A.6 Oracle Functionality

To link the game based definitions and our ideal functionality more easily, we define the IITM Oracle (Figure 3) which is parametrized by a variable $b \in$ {real, ideal}, two symmetric encryption schemes Σ_{authenc} and $\Sigma_{\text{unauthenc}}$, and a message authentication scheme Σ_{mac}, a pseudo-random function family F, and a leakage algorithm L. The machine Oracle is used in the proof of Theorem 3. The variable b specifies the behavior; real or ideal behavior. The IITM has an enriching I/O input tape $T^{\text{in}}_{\text{oracle}}$ and an I/O output tape $T^{\text{out}}_{\text{oracle}}$. First, Oracle waits for receiving a initialization message which contains a type t for which Oracle then generates a key. Then, the environment of Oracle can use it to perform the chosen cryptographic operation with it.

If Σ_{authenc}, $\Sigma_{\text{unauthenc}}$, Σ_{mac}, F, and L are clear from the context, Oracle$(b, \Sigma_{\text{authenc}}, \Sigma_{\text{unauthenc}}, \Sigma_{\text{pub}}, \Sigma_{\text{mac}}, F, L)$ is abbreviated by Oracle(b), for $b \in$ {real, ideal}.

The following lemma can be proven by standard cryptographic reductions, see [35] for the case of symmetric encryption.

Lemma 1. *Let $\Sigma_{\text{authenc}}, \Sigma_{\text{unauthenc}}$ be encryption schemes as above. Let Σ_{mac} be a MAC scheme and F be a function family as above. Let L be a leakage algorithm which leaks exactly the length of a message. Then,*

$$\mathcal{E} \mid \text{Oracle}(\mathit{real}) \equiv \mathcal{E} \mid \text{Oracle}(\mathit{ideal})$$

for every environment $\mathcal{E}$ for Oracle *if and only if $\Sigma_{\text{unauthenc}}$ is IND-CCA, Σ_{authenc} is IND-CPA and INT-CTXT, and Σ_{mac} is UF-CMA secure, and F is a pseudo-random function family. (The direction from right to left holds for any length preserving leakage algorithm L.)*

B Formal Specification of the Ideal Crypto Functionality

Our ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ is formally defined as pseudo code in Figure 4 to 6. As mentioned, the informal description provided above contains all necessary details to understand how $\mathcal{F}_{\text{crypto}}$ works. However, the pseudo code is convenient in some proofs. Next, we describe some notation and terminology which is used in these figures.

The description of $\mathcal{F}_{\text{crypto}}$ is divided into three parts: *Tapes*, *State*, and *Compute*. The first part is used to describe the input and output tapes. The second part is used to describe the variables that describe the state of $\mathcal{F}_{\text{crypto}}$ and also the initial state while the last describes the behavior of $\mathcal{F}_{\text{crypto}}$ in mode Compute. (In mode CheckAddress, $\mathcal{F}_{\text{crypto}}$ accepts all messages.) The description in mode Compute, consists of a sequence of blocks where every block is of the form $\langle condition \rangle : \langle actions \rangle$. Upon activation, the conditions of the blocks are checked one after the other. If a condition is satisfied the corresponding actions are carried out.

A condition is often of the form "receive m on t" for a message m and a tape t. This condition is satisfied if a message is received on tape t and the message is of the form m.

In the description of actions we often write "send m on t". This means that $\mathcal{F}_{\text{crypto}}$ outputs message m on tape t and stops for this activation. In the next activation the IITM will not proceed at the point where it stopped, but again go through the list of conditions, starting with the first one, as explained above. However, if we write "send m on t and wait for receiving m' on t'", then $\mathcal{F}_{\text{crypto}}$ does the following: It outputs m on tape t and stops for this activation. In the next activation with a message for the same user, it will check whether it received a message on input tape t' and check whether this message matches with m'. If it does, the computation continues. Otherwise, $\mathcal{F}_{\text{crypto}}$ stops for this activation without producing output. In the next activation, it will again check whether it received a message on input tape t' and whether this message matches with m' and behaves as before, and so on, until it receives the expected message on t'. However, if $\mathcal{F}_{\text{crypto}}$ receives a message for another user, then $\mathcal{F}_{\text{crypto}}$ will start where this users activation last stopped, i.e., either with the first condition or at some "wait for receiving m' on t'" statement. That is, $\mathcal{F}_{\text{crypto}}$ maintains one "thread" for every user. For example, if a user $(p, lsid, r)$ sends a request to generate a fresh symmetric key to $\mathcal{F}_{\text{crypto}}$, then $\mathcal{F}_{\text{crypto}}$ forwards this request to the adversary and waits for a response from the adversary. Now, if another user $(p', lsid', r')$ sends a request to $\mathcal{F}_{\text{crypto}}$ to also generate a fresh symmetric key before $\mathcal{F}_{\text{crypto}}$ received the response for user $(p, lsid, r)$ from the adversary, then $\mathcal{F}_{\text{crypto}}$ will also $\mathcal{F}_{\text{crypto}}$ forwards this request to the adversary and waits for a response from the adversary. That is, now $\mathcal{F}_{\text{crypto}}$ is waiting for two responses from the adversary, one for user $(p, lsid, r)$ and one for user $(p', lsid', r')$.

Given a configuration of $\mathcal{F}_{\text{crypto}}$, by *"create a new pointer ptr (for $(p, lsid, r)$) to $(t, k) \in \mathcal{K}$"* we denote the algorithm that finds the lexicographically smallest bit string ptr such that $\mathbf{key}(ptr, p, lsid, r)$ is not defined and sets $\mathbf{key}(ptr, p, lsid, r) := (t, k)$.

A bit string x is called a *valid user plaintext* (for party p in role r with LSID $lsid$) if x is well-tagged and for every bit string $\text{tag}_t(ptr)$ contained in x where $t \in \{\text{authenc-key}, \text{unauthenc-key}, \text{mac-key}, \text{pre-key}\}$ it holds that $\mathbf{key}(ptr, p, lsid, r)$ is defined and $\mathbf{key}(ptr, p, lsid, r) = (t, k)$ for some k.

C Proof of Theorem 3

The direction from left to right is easy to prove by standard cryptographic reductions. Because the domain of plaintexts is the set of well-tagged bit strings, using the store and retrieve commands, $\mathcal{F}_{\text{crypto}}/\mathcal{P}_{\text{crypto}}$ can be used to encrypt any plaintext. Now, it is easy to see that $\mathcal{F}_{\text{crypto}}$ offers at least the possibilities as Oracle, (defined in Appendix A.6), $\mathcal{F}_{\text{pke}}$, and $\mathcal{F}_{\text{sig}}$. By Lemma 1 and the results in [33], we can conclude that if $\mathcal{F}^* \,|\, \mathcal{P}_{\text{crypto}} \leq \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$, then $\Sigma_{\text{unauthenc}}$ and Σ_{pub} are IND-CCA secure, Σ_{authenc} is IND-CPA and INT-CTXT secure, and Σ_{mac} and Σ_{sig} are UF-CMA secure.

To prove the direction from right to left we use a hybrid argument, see Section 3.3 for a sketch of this proof. As mentioned in the proof sketch, we first introduce an intermediate system $\mathcal{P}'_{\text{crypto}}$ where we replace public-key encryption with Σ_{pub} and digital signatures with Σ_{sig} by the use of the ideal functionalities $\mathcal{F}_{\text{pke}}$ for public-key encryption and $\mathcal{F}_{\text{sig}}$ for digital signatures presented in [33]. More precisely, $\mathcal{P}'_{\text{crypto}}$ uses (instances of) $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{sig}}$ as a sub-protocol. There will be one instance of $\mathcal{F}_{\text{pke}}$ and one of $\mathcal{F}_{\text{sig}}$ for every party which handles all requests for public/private keys of this party. The addressing can be done using the session version $\overline{\mathcal{F}_{\text{pke}}}$ (resp., $\overline{\mathcal{F}_{\text{sig}}}$) of $\mathcal{F}_{\text{pke}}$ (resp., $\mathcal{F}_{\text{sig}}$) where the PID is used as a session ID to address the different instances of $\mathcal{F}_{\text{pke}}$ (resp., $\mathcal{F}_{\text{sig}}$); see [33] for details. In [33], it has been shown that realizing $\mathcal{F}_{\text{pke}}$ by a public-key encryption scheme is equivalent to the public-key encryption scheme

$\mathcal{F}_{\text{crypto}}(q, n, L)$

Tapes: input: T_r^{in} for all $r \in \{1, \ldots, n\}$ (enriching I/O tapes), $T_{\text{adv}}^{\text{in}}$ (network tape)
output: T_r^{out} for all $r \in \{1, \ldots, n\}$ (I/O tapes), $T_{\text{adv}}^{\text{out}}$ (network tape)

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

State: The state of $\mathcal{F}_{\text{crypto}}$ consists of the following variables:

- $\mathbf{enc}_{\text{authenc}}, \mathbf{dec}_{\text{authenc}}, \mathbf{enc}_{\text{unauthenc}}, \mathbf{dec}_{\text{unauthenc}}, \mathbf{enc}_{\text{pke}}, \mathbf{dec}_{\text{pke}}\mathbf{mac}, \mathbf{mac\text{-}verify}, \mathbf{sig}, \mathbf{sig\text{-}verify} \in \{0,1\}^* \cup \{\perp\}$ (initially $\perp$; interpreted as algorithms)
- $\mathcal{K} \subseteq \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\} \times \{0,1\}^*$ (initially $\emptyset$)
- $\mathcal{K}_{\text{known}} \subseteq \mathcal{K}$ (initially $\emptyset$; $\mathcal{K}_{\text{unknown}} = \mathcal{K} \setminus \mathcal{K}_{\text{known}}$)
- $\mathbf{key}\colon \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \times \mathbf{N} \to \mathcal{K} \cup \{\perp\}$ (initially $\mathbf{key}(ptr, p, lsid, r) := \perp$ for all $ptr, p, lsid, r$)
- $\mathbf{pk}_{\text{pke}}, \mathbf{sk}_{\text{pke}}, \mathbf{pk}_{\text{sig}}, \mathbf{sk}_{\text{sig}}\colon \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ (initially $p \mapsto \perp$ for all p)
- $\mathcal{N} \subseteq \{0,1\}^*$ (initially $\emptyset$)
- $\mathbf{Corrupt} \subseteq \{0,1\}^*$ (initially $\emptyset$)

Furthermore, $\mathcal{F}_{\text{crypto}}$ keeps track of several operations that have been performed.

For all variables that are interpreted as algorithms (e.g., $\mathbf{enc}_{\text{authenc}}$), $\mathcal{F}_{\text{crypto}}$ simulates its execution as follows: 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 $\perp$. Similarly, by "$y := A(x)$" we denote (enforced) deterministic execution of A.

CheckAddress: Every message on every tape is always accepted.

Compute:

Network input: Upon receiving a message m from T_{adv} do:

1. If $m = (\mathsf{Algorithms}, e, d, e', d', mac, mac\text{-}verify, sig, sig\text{-}verify)$ and $\mathbf{enc}_{\text{authenc}} = \perp$, then: Set $\mathbf{enc}_{\text{authenc}} := e$, $\mathbf{dec}_{\text{authenc}} := d$, $\mathbf{enc}_{\text{unauthenc}} := e'$, $\mathbf{dec}_{\text{unauthenc}} := d'$, $\mathbf{mac} := mac$, $\mathbf{mac\text{-}verify} := mac\text{-}verify$, $\mathbf{sig} := sig$, $\mathbf{sig\text{-}verify} := sig\text{-}verify$, and send (Ack) to T_{adv}.
2. If $m = (\mathsf{KeysPKE}, p, pk, sk, corrupt)$ such that $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ and $\mathbf{pk}_{\text{pke}}(p) = \perp$, then: Set $\mathbf{pk}_{\text{pke}}(p) := pk$ and $\mathbf{sk}_{\text{pke}}(p) := sk$. If $corrupt = \mathsf{true}$, then add (pke, p) to $\mathbf{Corrupt}$. Send (Ack) to T_{adv}.
3. If $m = (\mathsf{KeysSig}, p, pk, sk, corrupt)$ such that $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ and $\mathbf{pk}_{\text{sig}}(p) = \perp$, then: Set $\mathbf{pk}_{\text{sig}}(p) := pk$ and $\mathbf{sk}_{\text{sig}}(p) := sk$. If $corrupt = \mathsf{true}$, then add (sig, p) to $\mathbf{Corrupt}$. Send (Ack) to T_{adv}.

I/O input: Upon receiving a message m from $(p, lsid, r)$ do:

4. **Symmetric key generation:** If $m = (\mathsf{New}, t)$ for some $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$, then:
   (a) Send $(p, lsid, r, \mathsf{New}, t)$ to T_{adv} and wait for receiving $(p, lsid, r, \mathsf{Continue}, corrupt, k)$ from T_{adv} where $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ and $((t,k) \notin \mathcal{K}$ or $(corrupt = \mathsf{true}$ and $(t,k) \in \mathcal{K}_{\text{known}}))$.
   (b) Add (t,k) to $\mathcal{K}$ (if $(t,k) \notin \mathcal{K}$). Create a new pointer ptr for $(p, lsid, r)$ to (t,k).
   (c) If $corrupt = \mathsf{true}$, then add $(\mathsf{ptr}, (p, lsid, r), ptr)$ to $\mathbf{Corrupt}$ and add (t,k) to $\mathcal{K}_{\text{known}}$.
   (d) Send $(\mathsf{Pointer}, ptr)$ to $(p, lsid, r)$.
5. **Public key requests:** If $m = (\mathsf{GetPubKeyPKE}, p')$ (resp., $m = (\mathsf{GetPubKeySig}, p')$), then send (PubKey, pk) to $(p, lsid, r)$ where $pk = \mathbf{pk}_{\text{pke}}(p')$ (resp., $pk = \mathbf{pk}_{\text{sig}}(p')$).
6. **Nonce generation:** If $m = (\mathsf{NewNonce})$, then: Send $(p, lsid, r, \mathsf{NewNonce})$ to T_{adv} and wait for receiving $(p, lsid, r, \mathsf{Continue}, corrupt, x)$ from T_{adv} where $x \notin \mathcal{N}$. Add x to $\mathcal{N}$ (if $x \notin \mathcal{N}$). Send (Nonce, x) to $(p, lsid, r)$.
7. **Pre-shared keys:** If $m = (\mathsf{GetPSK}, t, name)$ for some $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$ and $name \in \{0,1\}^*$, then:
   (a) Send $(p, lsid, r, \mathsf{GetPSK}, t, name)$ to T_{adv} and wait for receiving $(p, lsid, r, \mathsf{Continue}, corrupt, k)$ from T_{adv} where $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ and the following is satisfied: i) if a key (t, k') is recorded for $(t, name)$, then $k = k'$ and $corrupt = \mathsf{false}$, ii) if $(\mathsf{psk}, t, name) \in \mathbf{Corrupt}$, then $corrupt = \mathsf{true}$, iii) if $corrupt = \mathsf{true}$, then $(t,k) \notin \mathcal{K}_{\text{unknown}}$, and iv) if no key (t, k') is recorded for $(t, name)$ and $corrupt = \mathsf{false}$, then $(t,k) \notin \mathcal{K}$.
   (b) Add (t,k) to $\mathcal{K}$ (if $(t,k) \notin \mathcal{K}$). Create a new pointer ptr for $(p, lsid, r)$ to (t,k).
   (c) If $corrupt = \mathsf{true}$, then add $(\mathsf{psk}, t, name)$ and $(\mathsf{ptr}, (p, lsid, r), ptr)$ to $\mathbf{Corrupt}$ and add (t,k) to $\mathcal{K}_{\text{known}}$. Otherwise, record (t,k) as the pre-shared key $name$.
   (d) Send $(\mathsf{Pointer}, ptr)$ to $(p, lsid, r)$.

Fig. 4. Ideal Crypto Functionality $\mathcal{F}_{\text{crypto}}$; parameterized by a polynomial q, a number $n \in \mathbf{N}$ which defines the I/O interface, and a leakage algorithm L.

$\mathcal{F}_{\text{crypto}}(q, n, L)$

8. **Encryption under symmetric keys:** If $m = (\mathsf{Enc}, ptr, x)$ for some ptr, x where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}\}$ and some k, x is a valid user plaintext, and $\mathbf{enc}_t \neq \bot$ (i.e., the encryption algorithm is defined), then:
   (a) In the following, let *ideal* = true if $(t, k) \in \mathcal{K}_{\text{unknown}}$, otherwise, let *ideal* = false.
   (b) *Translate pointers in x to keys, obtain x′:* Obtain x' from x by replacing every pointer $\text{tag}_{t'}(ptr')$ in x (i.e., where $t' \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$) by $\text{tag}_{t'}(k')$ where $(t', k') = \mathbf{key}(ptr', p, lsid, r)$.
   (c) *Encrypt x′, obtain ciphertext y:*
   if $x' \notin \text{dom}(L)_\eta$ **then**
   $y := \bot$
   else if *ideal* = true **then**
   $\overline{x} \leftarrow L(1^\eta, x')$, $y \leftarrow \mathbf{enc}_t(k, \overline{x})$, $\overline{x}' := \mathbf{dec}_t(k, y)$
   if $\overline{x}' = \overline{x}$ **then** record (x', y) for key (t, k) (for later decryption) **else** $y := \bot$
   else {*ideal* = false}
   $y \leftarrow \mathbf{enc}_t(k, x')$
   (d) *Update $\mathcal{K}_{\text{known}}$:* If $y \neq \bot$ and *ideal* = false, then: For every key $\text{tag}_{t'}(k')$ in x' (i.e., where $t' \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$), add (t', k') to $\mathcal{K}_{\text{known}}$.
   (e) *Return ciphertext:* Send $(\mathsf{Ciphertext}, y)$ to $(p, lsid, r)$.
9. **Encryption under public keys:** If $m = (\mathsf{PKEnc}, p', pk, x)$ for some p', pk, x where x is a valid user plaintext, $\mathbf{enc}_{\text{pke}} \neq \bot$, and $\mathbf{pk}_{\text{pke}}(p') \neq \bot$ then perform the encryption as the encryption under symmetric keys, i.e., 8. (a) to (e), expect that: i) *ideal* = false iff $(\mathsf{pke}, p') \in$ **Corrupt** or $\mathbf{pk}_{\text{pke}}(p') \neq pk$, ii) $\mathbf{enc}_t(k, \cdot)$ is replaced by $\mathbf{enc}_{\text{pke}}(pk, \cdot)$, iii) $\mathbf{dec}_t(k, y)$ is replaced by $\mathbf{dec}_{\text{pke}}(\mathbf{sk}_{\text{pke}}(\mathbf{p}'), y)$, and iv) "record (x', y) for key (t, k)" is replaced by "record (x', y) for p'".
10. **Decryption under symmetric keys:** If $m = (\mathsf{Dec}, ptr, y)$ for some ptr, y where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}\}$ and some k, $y \in \{0, 1\}^*$, and $\mathbf{enc}_t \neq \bot$, then:
   (a) In the following, let *ideal* = false if $(t, k) \in \mathcal{K}_{\text{known}}$ or ($t =$ unauthenc-key and there exists no x' such that (upon encryption) (x', y) has been recorded for (t, k)), otherwise, let *ideal* = true.
   (b) *Decrypt y, obtain plaintext x′:*
   if *ideal* = true **then**
   if $\exists x_1, x_2$: $x_1 \neq x_2$ and both (x_1, y) and (x_2, y) have been recorded for (t, k) **then**
   $x' := \bot$
   else if $\exists x''$: (x'', y) has been recorded for (t, k) **then**
   $x' := x''$
   else {Note that in case $t =$ unauthenc-key, this cannot occur by definition of *ideal*}
   $x' := \bot$
   else {*ideal* = false}
   $x' := \mathbf{dec}_t(k, y)$
   if x' is not well-tagged **then** $x' := \bot$
   (c) *Prevent guessing:* If $x' \neq \bot$, *ideal* = false, and there exists a key $\text{tag}_{t'}(k')$ in x' such that $(t', k') \in \mathcal{K}_{\text{unknown}}$, then set $x' := \bot$.
   (d) *Update $\mathcal{K}_{\text{known}}$:* If $x' \neq \bot$ and *ideal* = false, then for every key $\text{tag}_{t'}(k')$ in x', add (t', k') to $\mathcal{K}$ and $\mathcal{K}_{\text{known}}$.
   (e) *Translate keys in x′ to pointers, obtain x:* Let x be the bit string obtained from x' by doing the following for every key $\text{tag}_{t'}(k')$ in x': Create a new pointer ptr' for $(p, lsid, r)$ to (t', k'), and replace $\text{tag}_{t'}(k')$ by $\text{tag}_{t'}(ptr')$ in x'.
   (f) *Return plaintext:* Send $(\mathsf{Plaintext}, x)$ to $(p, lsid, r)$.
11. **Decryption under private keys:** If $m = (\mathsf{PKDec}, y)$ for some $y \in \{0, 1\}^*$, $\mathbf{dec}_{\text{pke}} \neq \bot$, and $\mathbf{sk}_{\text{pke}}(p) \neq \bot$, then perform the decryption as the decryption under symmetric keys of type $t =$ unauthenc-key, i.e., 10. (a) to (e), expect that: i) *ideal* = false iff $(\mathsf{pke}, p) \in$ **Corrupt** or there exists no x' such that (upon encryption) (x', y) has been recorded for p, ii) $\mathbf{dec}_t(k, y)$ is replaced by $\mathbf{dec}_{\text{pke}}(\mathbf{sk}_{\text{pke}}(\mathbf{p}), y)$, iii) "... recorded for (t, k)" is replaced by "... recorded for p".

Fig. 5. Ideal Crypto Functionality $\mathcal{F}_{\text{crypto}}$; parameterized by a polynomial q, a number $n \in \mathbf{N}$ which defines the I/O interface, and a leakage algorithm L. (continued)

$\mathcal{F}_{\text{crypto}}(q, n, L)$

12. **Key derivation:** If $m = (\mathsf{Derive}, ptr, t', s)$ for some ptr where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some k and $t = \mathsf{pre\text{-}key}$, $t' \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$, and seed $s \in \{0,1\}^*$, then:
    (a) Send $(p, lsid, r, \mathsf{Derive}, t, k, t', s, known)$ to T_{adv} where $known = \mathsf{true}$ iff $(t, k) \in \mathcal{K}_{\text{known}}$ and wait for receiving $(p, lsid, r, \mathsf{Continue}, k')$ from T_{adv} where the following condition is satisfied:
    if exists k'' such that (t', k'') has been derived from (t, k) with seed s **then** $k' = k''$
    else if $(t, k) \in \mathcal{K}_{\text{unknown}}$ **then** $(t', k') \notin \mathcal{K}$
    else $(t', k') \notin \mathcal{K}_{\text{unknown}}$
    (b) If $(t, k) \in \mathcal{K}_{\text{known}}$, then add (t', k') to $\mathcal{K}_{\text{known}}$. Add (t', k') to $\mathcal{K}$. Record (t', k') as derived from (t, k) with seed s. Create a new pointer ptr' for $(p, lsid, r)$ to (t', k'). Send $(\mathsf{Pointer}, ptr')$ to $(p, lsid, r)$.
13. **MAC:** If $m = (\mathsf{Mac}, ptr, x)$ for some ptr, x where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some k and $t = \mathsf{mac\text{-}key}$, $x \in \{0,1\}^*$, and $\mathbf{mac} \neq \bot$, then:
    (a) *Compute MAC:* Compute $\sigma \leftarrow \mathbf{mac}(k, x)$ and $b := \mathbf{mac\text{-}verify}(k, x, \sigma)$. If $b \neq \mathsf{true}$, then set $\sigma := \bot$. If $(t, k) \in \mathcal{K}_{\text{unknown}}$ and $\sigma \neq \bot$, then record x for (t, k) (for later verification).
    (b) *Return MAC:* Send (Mac, σ) to $(p, lsid, r)$.
14. **Verify MAC:** If $m = (\mathsf{MacVerify}, ptr, x, \sigma)$ for some ptr, x, σ where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some k and $t = \mathsf{mac\text{-}key}$, $x, \sigma \in \{0,1\}^*$, and $\mathbf{mac} \neq \bot$, then:
    (a) *Verify MAC:* Compute $b := \mathbf{mac\text{-}verify}(k, x, \sigma)$.
    (b) *Prevent forgery:* If $(t, k) \in \mathcal{K}_{\text{unknown}}$, $b = \mathsf{true}$, and (upon MACing) x has not been recorded for (t, k), then set $b := \bot$.
    (c) *Return verification result:* Send $(\mathsf{MacVerify}, b)$ to $(p, lsid, r)$.
15. **Sign:** If $m = (\mathsf{Sign}, x)$ for some $x \in \{0,1\}^*$, $\mathbf{sig} \neq \bot$, and $\mathbf{sk}_{\text{sig}}(p) \neq \bot$, then:
    (a) *Compute signature:* Compute $\sigma \leftarrow \mathbf{sig}(\mathbf{sk}_{\text{sig}}(p), x)$ and $b := \mathbf{sig\text{-}verify}(\mathbf{pk}_{\text{sig}}(p), x, \sigma)$. If $b \neq \mathsf{true}$, then set $\sigma := \bot$. If $(\mathsf{sig}, p) \notin \mathbf{Corrupt}$, then record x for p (for later verification).
    (b) *Return signature:* Send (Sign, σ) to $(p, lsid, r)$.
16. **Verify signature:** If $m = (\mathsf{SigVerify}, p', pk, x, \sigma)$ for some p', pk, x, σ, $\mathbf{sig} \neq \bot$, and $\mathbf{pk}_{\text{sig}}(p') \neq \bot$ then:
    (a) *Verify signature:* Compute $b := \mathbf{sig\text{-}verify}(pk, x, \sigma)$.
    (b) *Prevent forgery:* If $b = \mathsf{true}$, $pk = \mathbf{pk}_{\text{sig}}(p')$, $(\mathsf{sig}, p') \notin \mathbf{Corrupt}$, and (upon signing) x has not been recorded for p', then set $b := \bot$.
    (c) *Return verification result:* Send $(\mathsf{SigVerify}, b)$ to $(p, lsid, r)$.
17. **Store:** If $m = (\mathsf{Store}, t, k)$ for some $t \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$ and $k \in \{0,1\}^*$, then: If $(t, k) \in \mathcal{K}_{\text{unknown}}$, then send $(\mathsf{Store}, \bot)$ to $(p, lsid, r)$. Otherwise, add (t, k) to $\mathcal{K}$ and $\mathcal{K}_{\text{known}}$, create a new pointer ptr for $(p, lsid, r)$ to (t, k), and send $(\mathsf{Pointer}, ptr)$ to $(p, lsid, r)$.
18. **Retrieve:** If $m = (\mathsf{Retrieve}, ptr)$ for some ptr where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ for some t, k, then: Add (t, k) to $\mathcal{K}_{\text{known}}$ and send (Key, k) to $(p, lsid, r)$.
19. **Equality test:** If $m = (\mathsf{Equal?}, ptr, ptr')$ for some ptr, ptr' where $(t, k) = \mathbf{key}(ptr, p, lsid, r)$ and $(t', k') = \mathbf{key}(ptr', p, lsid, r)$ for some t, k, t', k', then: If $t = t'$ and $k = k'$, send $(\mathsf{Equal}, \mathsf{true})$ to $(p, lsid, r)$, otherwise, send $(\mathsf{Equal}, \mathsf{false})$ to $(p, lsid, r)$.
20. **Corruption request for a symmetric key:** If $m = (\mathsf{Corrupted?}, ptr)$, then send $(\mathsf{Corrupted}, b)$ to $(p, lsid, r)$ where $b = \mathsf{true}$ if $(\mathsf{ptr}, (p, lsid, r), ptr) \in \mathbf{Corrupt}$ and, otherwise, $b = \mathsf{false}$.
21. **Corruption request for a public/private key:** If $m = (\mathsf{CorruptedPKE?}, p')$ (resp., $m = (\mathsf{CorruptedSig?}, p')$), then send $(\mathsf{Corrupted}, b)$ to $(p, lsid, r)$ where $b = \mathsf{true}$ if $(\mathsf{pke}, p') \in \mathbf{Corrupt}$ (resp., $(\mathsf{sig}, p') \in \mathbf{Corrupt}$) and, otherwise, $b = \mathsf{false}$.

Fig. 6. Ideal Crypto Functionality $\mathcal{F}_{\text{crypto}}$; parameterized by a polynomial q, a number $n \in \mathbf{N}$ which defines the I/O interface, and a leakage algorithm L. (continued)

being IND-CCA secure. Furthermore, in [33], it has been shown that realizing $\mathcal{F}_{\text{sig}}$ by a digital signature scheme is equivalent to the digital signature scheme being UF-CMA secure. From this, we obtain that $\mathcal{P}_{\text{crypto}} \leq \mathcal{P}'_{\text{crypto}}$.

Next, we show that $\mathcal{F}^* \,|\, \mathcal{P}'_{\text{crypto}} \leq \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$ which completes the proof. Note that public-key encryption and digital signatures in $\mathcal{P}'_{\text{crypto}}$ and $\mathcal{F}_{\text{crypto}}$ is already done identically in an ideal way. In the following, we do not consider keys of type unauthenc-key. For such keys the presentation of the proof is slightly more complicated because a key of type unauthenc-key which is marked unknown could be used for decryption before it is used for encryption. Now, if this is the r-th key than this decryption in the hybrid $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ will be handled by Oracle. But then, the r-th key might become known ($\mathcal{F}^*$ does not prevent this because it is not a commitment problem) and $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ needs to have the ability to extract the key from Oracle; see [35] for details.

Formulation of the Simulator. The simulator Sim_{crypto} is defined as follows: On the first activation it provides the encryption, decryption, MAC, and signature algorithms to $\mathcal{F}_{\text{crypto}}$. Upon generation of fresh keys, pre-shared keys, and nonces, Sim_{crypto} chooses keys and nonces as $\mathcal{P}'_{\text{crypto}}$. Upon key derivation from an unknown key, Sim_{crypto} chooses a fresh key uniformly at random from $\{0,1\}^\eta$. Otherwise, it uses the pseudo-random function F as $\mathcal{P}'_{\text{crypto}}$ does. Furthermore, Sim_{crypto} simulates the network interface of (instances of) $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{sig}}$ as it exists in $\mathcal{P}'_{\text{crypto}}$.

Formulation of the Hybrid Systems. We define the hybrid systems $\mathcal{F}^{(r)}_{\text{crypto}}$ and $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ for all $r \in \mathbf{N}$. The system $\mathcal{F}^{(r)}_{\text{crypto}}$ behaves like $\mathcal{F}_{\text{crypto}}$ except that the order in which unknown keys are used is tracked, as in $\mathcal{F}^*$. All keys with order $< r$ are treated ideal (as in $\mathcal{F}_{\text{crypto}}$) but keys with order $\geq r$ are treated real (as in the realization $\mathcal{P}'_{\text{crypto}}$). In $\mathcal{F}_{\text{crypto}}$ the adversary was not able to insert keys (upon key generation, store, decryption, or key derivation with corrupted or known keys) that collide with unknown objects (guessing of objects that are ideally not known). Here, this is only guaranteed for keys of order $\leq r$ or as long as there are no keys of order $> r$.

More formally: The system $\mathcal{F}^{(r)}_{\text{crypto}}$ has an additional variable $nextused \in \mathbf{N}$ (initially 1) and maintains a partial function $used$ from keys ($\mathcal{K}$) to numbers ($\mathbf{N}$) to keep track of the order in which unknown keys are used. Preventing key guessing is relaxed as follows:

1. For symmetric key generation $[(\mathsf{New}, t)]$, when receiving a key from the adversary, the condition "$(t,k) \notin \mathcal{K}$ or ($corrupt$ = true and $(t,k) \in \mathcal{K}_{\text{known}}$)" is replaced by "$(t,k) \notin \mathcal{K}$ or ($corrupt$ = true and $\text{Guess}^{(r)}(t,k)$ = false)" where $\text{Guess}^{(r)}(t,k) :=$ true iff $(t,k) \in \mathcal{K}_{\text{unknown}}$ and ($\bot \neq used(t,k) \leq r$ or $nextused \leq r$).
   We note that for every pair (t,k) we have that $\text{Guess}^{(p)}(t,k)$ = true iff $(t,k) \in \mathcal{K}_{\text{unknown}}$ where p bounds the runtime of the environment that uses $\mathcal{F}_{\text{crypto}}$. Furthermore, $\text{Guess}^{(0)}(t,k)$ = false for every pair (t,k).
2. For storing keys $[(\mathsf{Store}, t, k)]$, the statement "If $(t,k) \in \mathcal{K}_{\text{unknown}}$, then send (Store, $\bot$) to $(p, lsid, r)$." is replaced by "If $\text{Guess}^{(r)}(t,k)$ = true, then send (Store, $\bot$) to $(p, lsid, r)$."
3. For symmetric and public-key decryption $[(\mathsf{Dec}, ptr, y)$ and $(\mathsf{PKDec}, y)]$, in *Preventing guessing*, we replace the condition "$(t', k') \in \mathcal{K}_{\text{unknown}}$" by "$\text{Guess}^{(r)}(t', k')$ = true".

Furthermore, for encryption and decryption under symmetric keys $[(\mathsf{Enc}, ptr, x)$ and $(\mathsf{Dec}, ptr, y)]$ the definition of *ideal* (which determines whether encryption/decryption is performed ideal or real) is modified:

1. For symmetric encryption:
   if $(t,k) \in \mathcal{K}_{\text{unknown}}$ **then**
   if $used(t,k) = \bot$ **then** $used(t,k) := nextused{+}{+}$
   if $used(t,k) < r$ **then**
   $ideal :=$ true
   else
   $ideal :=$ false
   else
   $ideal :=$ false
2. For symmetric decryption:
   if $(t,k) \in \mathcal{K}_{\text{unknown}}$ **then**
   if $\bot \neq used(t,k) < r$ or $nextused \leq r$ **then**
   $ideal :=$ true
   else
   $ideal :=$ false

else
ideal := false

Finally, also key derivation $[(\mathsf{Derive}, ptr, t', s)]$ changes:

Key Derivation: If $m = (\mathsf{Derive}, ptr, t', s)$ for some ptr where $(t,k) = \mathbf{key}(ptr, p, lsid, r)$ for some k and $t = \mathsf{pre\text{-}key}$, $t' \in \{\mathsf{authenc\text{-}key}, \mathsf{unauthenc\text{-}key}, \mathsf{mac\text{-}key}, \mathsf{pre\text{-}key}\}$, and seed $s \in \{0,1\}^*$, then:

(a) If there exists a bit string k' such that (t', k') has been derived from (t,k) with seed s, then create a new pointer ptr' for $(p, lsid, r)$ to (t', k') and send $(\mathsf{Pointer}, ptr')$ to $(p, lsid, r)$. Otherwise, continue.

(b) If $(t,k) \in \mathcal{K}_{\text{known}}$, then set $k' := F_\eta(k, \text{tag}_t(s))$.
Otherwise, if $used(t,k)$ is undefined (i.e., $used(t,k) = \perp$), then $used(t,k) := nextused{+}{+}$. If $used(t,k) < r$, then choose $k' \leftarrow^{\text{R}} \{0,1\}^\eta$. Otherwise, set $k' := F_\eta(k, \text{tag}_{t'}(s))$.

(c) If $(t,k) \in \mathcal{K}_{\text{known}}$ and $\text{Guess}^{(r)}(t,k) = \mathsf{true}$, then produce empty output in this activation.
Otherwise, add (t', k') to $\mathcal{K}$ and record (t', k') as derived from (t,k) with seed s. If $(t,k) \in \mathcal{K}_{\text{known}}$, then add (t', k') to $\mathcal{K}_{\text{known}}$. Create a new pointer ptr' for $(p, lsid, r)$ to (t', k'). Send $(\mathsf{Pointer}, ptr')$ to $(p, lsid, r)$.

The system $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ behaves like $\mathcal{F}^{(r)}_{\text{crypto}}$ except that it connects to Oracle and the key with order r is relayed out and handled by calls to Oracle. More formally: $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ has the additional I/O output tape $T^{\text{in}}_{\text{oracle}}$ and I/O input tape $T^{\text{out}}_{\text{oracle}}$. When a key gets assigned order r, i.e., $used(t,k) := r$ is computed (either upon encryption or key derivation), then Oracle is initialized, i.e., $(\mathsf{Init}, \mathsf{authenc})$ or $(\mathsf{Init}, \mathsf{prf})$, respectively, is sent to Oracle. (Recall that we do not consider keys of type $\mathsf{unauthenc\text{-}key}$ in this proof, see above.) Now, upon encryption $[(\mathsf{Enc}, ptr, x)]$ every thing is exactly as in $\mathcal{F}^{(r)}_{\text{crypto}}$ except that if $(t,k) \in \mathcal{K}_{\text{unknown}}$ and $used(t,k) = r$, then the ciphertext y is obtained by sending (Enc, x') to Oracle and waiting to receive y from Oracle. Similarly, upon decryption $[(\mathsf{Dec}, ptr, y)]$ every thing is exactly as in $\mathcal{F}^{(r)}_{\text{crypto}}$ except that if $(t,k) \in \mathcal{K}_{\text{unknown}}$ and $used(t,k) = r$, then the plaintext x' is obtained by sending (Dec, y) to Oracle and waiting to receive x' from Oracle. Likewise, upon key derivation $[(\mathsf{Derive}, ptr, t', s)]$ every thing is exactly as in $\mathcal{F}^{(r)}_{\text{crypto}}$ except that if $(t,k) \in \mathcal{K}_{\text{unknown}}$ and $used(t,k) = r$, then the key k' is obtained by sending $(\mathsf{PRF}, \text{tag}_{t'}(s))$ to Oracle and waiting to receive k' from Oracle.

Completing the Proof of Theorem 3. Let $\mathcal{E}$ be an environment for $\mathcal{F}^* \,|\, \mathcal{P}'_{\text{crypto}}$ and $p_{\mathcal{E}}$ be a polynomial such that the overall length of all messages output by $\mathcal{E}$ in any run of $\mathcal{E} \,|\, \mathcal{Q}(1^\eta, a)$ for any system $\mathcal{Q}$, security parameter $\eta \in \mathbf{N}$ and initial input $a \in \{0,1\}^*$ is bound by $p_{\mathcal{E}}(\eta + |a|)$. Since $\mathcal{E}$ is an environmental system (all input tapes are consuming) such a polynomial always exists.

For all $r \in \mathbf{N}$, $b \in \{\mathsf{real}, \mathsf{ideal}\}$ we define the following *combined* systems:

$$\mathcal{C}^{(r)} = \mathcal{E} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}^{(r)}_{\text{crypto}}$$
$$\widehat{\mathcal{C}}^{(r)}_b = \mathcal{E} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}^* \,|\, \widehat{\mathcal{F}}^{(r)}_{\text{crypto}} \,|\, \text{Oracle}(b) \ .$$

Next, we define an error set (i.e., a negligible set of runs we do not want to consider) for collisions of honestly generated keys and nonces. Let $B^{(r)}_{\text{coll}}(1^\eta, a)$ be the set of runs of $\mathcal{C}^{(r)}(1^\eta, a)$ where the simulator Sim_{crypto} or $\mathcal{F}^{(r)}_{\text{crypto}}$ generates a new key of some type, say t, or a nonce, that collides with some key in $\mathcal{K}$ or a nonce in $\mathcal{N}$, respectively.

The following lemma is used in the proofs of Lemma 3 and 4. It is easy to prove because we assume that all keys and nonces are chosen uniformly at random. But even without this assumption the lemma is simple to prove because key collisions with non-negligible probability can be used to construct an adversary with non-negligible advantage.

Lemma 2. *There exists a negligible function f_{coll} such that for all $r \in \mathbf{N}$, $\eta \in \mathbf{N}$ and $a \in \{0,1\}^*$*

$$\Pr\left[B^{(r)}_{\text{coll}}(1^\eta, a)\right] \leq f_{\text{coll}}(1^\eta, a) \ . \tag{1}$$

In the following, we call two systems $\mathcal{P}$ and $\mathcal{Q}$ *f-indistinguishable* ($\mathcal{P} \equiv_f \mathcal{Q}$) iff the difference between the probability that $\mathcal{P}$ outputs 1 (on the decision tape) and the probability that $\mathcal{Q}$ outputs 1 (on the decision tape) is bounded from above by $f(1^\eta, a)$ for all security parameter $\eta \in \mathbf{N}$ and external input $a \in \{0,1\}^*$.

Lemma 3. *There exists a negligible function f_0 such that*

$$\mathcal{C}^{(0)} \equiv_{f_0} \mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}'_{\text{crypto}} \tag{2}$$
$$\mathcal{C}^{(p_{\mathcal{E}})} \equiv_{f_0} \mathcal{E} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}} \ . \tag{3}$$

Proof. Let $f_{\text{coll}}(1^\eta, a) = \Pr[B^{(0)}_{\text{coll}}(1^\eta, a)]$. By Lemma 2 we have that f_{coll} is negligible.

ad (2): Now, we show that $\mathcal{E}\,|\,\mathcal{F}^*\,|\,\mathcal{P}'_{\text{crypto}} \equiv_{f_{\text{coll}}} C^{(0)}$. Note that in every run of $C^{(0)}(1^\eta, a)$ it always holds that $nextused \geq 1$ and for any $(t,k) \in \mathcal{K}$ it holds that $used(t,k) = \bot$ or $used(t,k) > 0$. In particular this implies that $\text{Guess}^{(0)}(t,k) = \mathsf{false}$ for all $(t,k) \in \mathcal{K}$. One easily verifies that every run of $C^{(0)}(1^\eta, a)$ where $B^{(0)}_{\text{coll}}(1^\eta, a)$ does not occur corresponds, i.e., can be injectively mapped, to a run of $\mathcal{E}\,|\,\mathcal{F}^*\,|\,\mathcal{P}'_{\text{crypto}}(1^\eta, a)$ with the same overall output and probability. We conclude that

$$\left|\Pr[\mathcal{E}\,|\,\mathcal{F}^*\,|\,\mathcal{P}'_{\text{crypto}}(1^\eta, a) \rightsquigarrow 1] - \Pr[C^{(0)}(1^\eta, a) \rightsquigarrow 1]\right| \leq f_{\text{coll}}(1^\eta, a) \ .^3$$

ad (3): First, we define an intermediate system Q which is defined exactly like the ideal system $\mathcal{E}\,|\,Sim_{\text{crypto}}\,|\,\mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$ except that verification of a MAC is done as in the real system, i.e., the request $(\mathsf{MacVerify}, ptr, x, \sigma)$ is handled as in $\mathcal{P}'_{\text{crypto}}$. In every run of $C^{(p_\mathcal{E}(\eta+|a|))}(1^\eta, a)$ it always holds that $nextused \leq p_\mathcal{E}(\eta + |a|)$, hence, for all pairs (t,k) we have that $\text{Guess}^{(p_\mathcal{E}(\eta+|a|))}(t,k) = \mathsf{true}$ iff $(t,k) \in \mathcal{K}_{\text{unknown}}$. One easily verifies that every run of $C^{(p_\mathcal{E}(\eta+|a|))}(1^\eta, a)$ where $B^{(p_\mathcal{E}(\eta+|a|))}_{\text{coll}}(1^\eta, a)$ does not occur corresponds, i.e., can be injectively mapped, to a run of $Q(1^\eta, a)$ with the same overall output and probability. We conclude that

$$\left|\Pr[Q(1^\eta, a) \rightsquigarrow 1] - \Pr[C^{(p_\mathcal{E}(\eta+|a|))}(1^\eta, a) \rightsquigarrow 1]\right| \leq f_{\text{coll}}(1^\eta, a) \ .$$

Next, by a hybrid argument, we prove that $Q \equiv \mathcal{E}\,|\,Sim_{\text{crypto}}\,|\,\mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$ which completes the proof. Therefore, we define the hybrid systems $Q^{(r)}$ for $r \in \mathbf{N}$ which connect to Oracle. The system $Q^{(r)}$ is defined like Q except that we order the keys of type $\mathsf{mac\text{-}key}$ in the order they are created (i.e., generated upon request of the form (New, t) or $(\mathsf{Derive}, ptr, t', s)$). Note that this order is independent of the used-order defined above. Now, $Q^{(r)}$ handles every MAC key of order $< r$ as in Q, every key with order $> r$ as in $\mathcal{E}\,|\,Sim_{\text{crypto}}\,|\,\mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$, and the key of order r is externally handled in Oracle. Using Lemma 1, by a standard hybrid argument we obtain that $Q \equiv \mathcal{E}\,|\,Sim_{\text{crypto}}\,|\,\mathcal{F}^*\,|\,\mathcal{F}_{\text{crypto}}$. □

The next lemma is used to show that the r-th hybrid is indistinguishable from the $(r+1)$-th hybrid.

Lemma 4. *There exist negligible functions f_{real}, f_{ideal} such that*

$$C^{(r)} \equiv_{f_{real}} \widehat{C}^{(r)}_{real} \quad \textit{for all } r \in \mathbf{N} \textit{ and} \tag{4}$$

$$C^{(r+1)} \equiv_{f_{ideal}} \widehat{C}^{(r)}_{ideal} \quad \textit{for all } r \in \mathbf{N} \ . \tag{5}$$

Proof. We only show (4), the proof of (5) is similar. First, we note that the two systems $C^{(r)}$ and $\widehat{C}^{(r)}_{\text{real}}$ are already very close to each other because every key (including the r-th key) is treated in the same way (real or ideal) in the two systems. The only difference is that in $C^{(r)}$ the r-th key is handled inside $\mathcal{F}^{(r)}_{\text{crypto}}$ while in $\widehat{C}^{(r)}_{\text{real}}$ cryptographic operations with the r-th key are handled inside Oracle and $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ uses a different key than the key in Oracle if the r-th key is encrypted. Since the used-order is respected, the r-th key is always encrypted ideally, i.e., not the key itself is encrypted but its leakage. Hence, the actual value of the key when encrypted does not matter. Furthermore, even if the r-th key is a derived key, it has been derived ideally, i.e., chosen uniformly at random, and hence, it is distributed just like the key in Oracle. Thus, the only difference between $C^{(r)}$ and $\widehat{C}^{(r)}_{\text{real}}$ occurs upon key collisions (if a freshly generated key by Sim_{crypto} collides with some other key) or if the environment is able to guess the r-th key (see below). But because this keys was only encrypted ideally, we can show that this probability is negligible.

More formally, to show (4), we define a mapping from every run ρ of $C^{(r)}$, excluding a negligible set $B^{(r)}$ of error runs (see below), to a set S_ρ of runs of $\widehat{C}^{(r)}_{\text{real}}$ such that the probability of ρ is the same as the one for S_ρ and the overall output of ρ (on tape $\mathsf{decision}$) is the same as the overall output of every run in S_ρ. Such a mapping implies that $C^{(r)} \equiv \widehat{C}^{(r)}_{\text{real}}$.

We define the error set $B^{(r)}$ to be $B^{(r)}_{\text{coll}} \cup B^{(r)}_{\text{guess}}$. The set $B^{(r)}_{\text{coll}}$ was defined above and its probability was shown to be negligible in Lemma 2, with a bound that is independent of r. The set $B^{(r)}_{\text{guess}}$ is the set of runs of $C^{(r)}$ where $\mathcal{E}$ "guesses" the r-th key, i.e., where at some point during the run $\mathcal{E}$ wants to use the r-th key as a corrupted symmetric key, $\mathcal{E}$ wants to store the r-th key, or $\mathcal{E}$ decrypts a ciphertext with a known symmetric key or a corrupted public key and

[3] $\Pr[\mathcal{S}(1^\eta, a) \rightsquigarrow 1]$ denotes the probability that $\mathcal{S}$ outputs 1 (on the decision tape) upon external input a and security parameter η.

where the decryption contains the r-th key. If the probability for $B^{(r)}_{\text{guess}}$ were non-negligible, one could easily construct a successful adversary against the encryption and pseudo-random function games in question: The adversary would simulate the system $C^{(r)}$ using his oracles to perform operations with the r-th key and guesses the position (among polynomially many possible positions) where $\mathcal{E}$ guesses the r-th key. This simulation corresponds to the actual system $C^{(r)}$ because the r-th key is only encrypted ideally and if it is a derived key, it was derived ideally, and hence, its distribution corresponds to the one in the game. Thus, if the guesses of the adversary and the emulated $\mathcal{E}$ are correct, the adversary can easily win the game. It follows that the probability for $B^{(r)}_{\text{guess}}$ must be negligible. To obtain a bound which is independent of r, one can consider an adversary that first guesses r and then proceeds as above. Altogether, we obtain a negligible function, which is independent of r, and bounds the probability for $B^{(r)}$.

Now, the mapping is defined in the obvious way as follows: Let ρ be a run of $C^{(r)}$ which is not in $B^{(r)}$. We now define S_ρ to be the set of runs of $\widehat{C}^{(r)}_{\mathsf{real}}$ where the key in Oracle (in $\widehat{C}^{(r)}_{\mathsf{real}}$) is defined to be the r-th key in ρ and where we use a freshly generated key as the r-th key in $\widehat{\mathcal{F}}^{(r)}_{\text{crypto}}$ (in $\widehat{C}^{(r)}_{\mathsf{real}}$). Every such freshly generated key induces one run in S_ρ. By construction, the probabilities of ρ and S_ρ are equal. Now, with the observations made above (the r-th key is ideally encrypted and uniformly distributed) and the fact that collisions and guessing does not occur, one can easily show, by induction on the length of runs, that the view of $\mathcal{E}$ in ρ is the same as the view of $\mathcal{E}$ in every run ρ' in S_ρ, and hence, the overall output is the same.

Since the probability of $B^{(r)}$ is bounded by a negligible function independently of r, we obtain that there exists a negligible function f_{real} such that $C^{(r)} \equiv_{f_{\mathsf{real}}} \widehat{C}^{(r)}_{\mathsf{real}}$ for all $r \in \mathbf{N}$. □

Finally, using the lemmas above, we complete the proof of Theorem 3. Let $\mathcal{E}'$ be the system that upon initial input a first chooses $r \in \{0, \ldots, p_{\mathcal{E}}(\eta+|a|)-1\}$ uniformly at random and then behaves exactly like $\mathcal{E} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}^* \,|\, \widehat{\mathcal{F}}^{(r)}_{\text{crypto}}(1^\eta, a)$. Clearly, $\mathcal{E}'$ is an environment for Oracle and, hence, by Lemma 1 we find a negligible function f_{O} such that

$$\mathcal{E}' \mid \text{Oracle}(\mathsf{real}) \equiv_{f_{\text{O}}} \mathcal{E}' \mid \text{Oracle}(\mathsf{ideal}) \ . \tag{6}$$

By definition, for all $\eta \in \mathbf{N}$, $a \in \{0,1\}^*$, $r < p_{\mathcal{E}}(\eta + |a|)$, and $b \in \{\mathsf{real}, \mathsf{ideal}\}$ it holds that

$$\begin{aligned} &\Pr\left[\widehat{C}^{(r)}_b(1^\eta, a) \rightsquigarrow 1\right] \\ &= \Pr\left[(\mathcal{E}' \,|\, \text{Oracle}(b))(1^\eta, a) \rightsquigarrow 1 \,\middle|\, \mathcal{E}' \text{ chooses } r\right] \\ &= p_{\mathcal{E}}(\eta + |a|) \cdot \Pr\left[(\mathcal{E}' \,|\, \text{Oracle}(b))(1^\eta, a) \rightsquigarrow 1 \text{ and } \mathcal{E}' \text{ chooses } r\right] \ . \end{aligned} \tag{7}$$

In the following, we abbreviate $\Pr[Q] = \Pr[Q(1^\eta, a) \rightsquigarrow 1]$ for all systems Q, $p_{\mathcal{E}} = p_{\mathcal{E}}(\eta + |a|)$, and $f_x = f_x(1^\eta, a)$ for all x. Now, for all $\eta \in \mathbf{N}$ and $a \in \{0,1\}^*$ it holds that:

$$\begin{aligned} &\left|\Pr[\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}'_{\text{crypto}}] - \Pr[\mathcal{E} \,|\, Sim_{\text{crypto}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}]\right| \\ &\overset{(2),(3)}{\leq} \left|\Pr\left[C^{(0)}\right] - \Pr\left[C^{(p_{\mathcal{E}})}\right]\right| + 2f_0 = \left|\sum_{r<p_{\mathcal{E}}} \Pr\left[C^{(r)}\right] - \Pr\left[C^{(r+1)}\right]\right| + 2f_0 \\ &\overset{(4),(5)}{\leq} \left|\sum_{r<p_{\mathcal{E}}} \Pr\left[\widehat{C}^{(r)}_{\mathsf{real}}\right] - \Pr\left[\widehat{C}^{(r)}_{\mathsf{ideal}}\right]\right| + 2f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &\overset{(7)}{=} p_{\mathcal{E}} \cdot \Bigg|\sum_{r<p_{\mathcal{E}}} \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{real}) \text{ and } \mathcal{E}' \text{ chooses } r\right] \\ &\qquad\qquad - \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{ideal}) \text{ and } \mathcal{E}' \text{ chooses } r\right]\Bigg| + 2f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &= p_{\mathcal{E}} \cdot \left|\Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{real})\right] - \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{ideal})\right]\right| + 2f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &\overset{(6)}{\leq} 2f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}} + f_{\text{O}}) \ . \end{aligned}$$

Since $2f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}} + f_{\text{O}})$ is negligible, $\mathcal{F}^* \,|\, \mathcal{P}'_{\text{crypto}} \leq \mathcal{F}^* \,|\, \mathcal{F}_{\text{crypto}}$. This concludes the proof. □