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, DNSSEC, IEEE 802.11i, 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), digital signatures, 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.

Keywords: security protocols, compositional analysis, simulation-based security

1 Introduction

Security protocols employed in practice, such as SSL/TLS, SSH, IPsec, DNSSEC, IEEE 802.11i, 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 [8, 30]: 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. Hence, they can be used as low-level idealized components in even more complex systems.

* This work was partially supported by the DFG under Grant KU 1434/5-1 and KU 1434/6-1.

This appealing approach has so far, however, been only 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 [8, 30, 23], only recently a functionality for symmetric encryption, which we denote by $\mathcal{F}_{\text{enc}}$ here, was proposed [25]. 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, for a reasonable class of environments, $\mathcal{F}_{\text{crypto}}$ 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. To prove this result, we extend the hybrid argument for $\mathcal{F}_{\text{enc}}$ in [25]. 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 [25, 24]) 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., [25, 24]). 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 [20]. More precisely, we analyze the 4-Way Handshake protocol (4WHS) for key exchange and the CCM Protocol (CCMP) of the pre-shared key mode of WPA2 (WPA2-PSK) 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 [32, 29]. Since we use standard cryptographic assumptions and constructions, our modeling of 4WHS and CCMP, including the message formats, is very 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 all proofs are provided in our technical report [26].

2 Simulation-based Security

In this section, we briefly recall the IITM model for simulation-based security (see [22] 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 [10], the IITM model has several advantages over the UC model and avoids some technical problems, as demonstrated and discussed in [22, 23, 25, 19].

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 [22].

Definition 1 ([22]). *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 [22]. 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 ([22]). *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, $\mathcal{P}_1 \,|\, \mathcal{P}_2$ and $\mathcal{F}_1 \,|\, \mathcal{F}_2$ are well-formed, and $\mathcal{P}_i \leq \mathcal{F}_i$, for $i \in \{1, 2\}$. Then, $\mathcal{P}_1 \,|\, \mathcal{P}_2 \leq \mathcal{F}_1 \,|\, \mathcal{F}_2$.*

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 ([22]). *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 (see [26] for details).

As mentioned in the introduction, $\mathcal{F}_{\text{crypto}}$ extends $\mathcal{F}_{\text{enc}}$, proposed in [25], 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 generate public/private keys, iii) to derive symmetric keys from other symmetric keys, iv) to encrypt and decrypt bit strings (public-key encryption and both unauthenticated and authenticated symmetric encryption is supported), v) to compute and verify MACs and digital signatures, and vi) 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, 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., [9]).

3.1 The Ideal Crypto Functionality

The ideal crypto functionality $\mathcal{F}_{\text{crypto}}$ is parametrized by what we call a *leakage algorithm L*, a probabilistic polynomial time algorithm which takes as input a security parameter η and a message m, and returns the information that may be leaked about m. Typical examples are i) $L(1^\eta, m) = 0^{|m|}$ and ii) the algorithm that returns a random bit string of length $|m|$. Both leakage algorithms leak exactly the length of 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); $\mathcal{F}_{\text{crypto}}$ has one I/O input and output tape for each role.

In $\mathcal{F}_{\text{crypto}}$, symmetric keys are equipped with types. Keys that may be used for authenticated encryption have type authenc-key, those for unauthenticated encryption have type unauthenc-key. We have the types mac-key for MAC keys and pre-key for

keys from which new keys can be derived. All types are disjoint, i.e., a key can only have one type, reflecting common practice that a symmetric key only serves one purpose. For example, a MAC key is not used for encryption and keys from which other keys are derived are typically not used as encryption/MAC keys.

While users of $\mathcal{F}_{\text{crypto}}$, and its realization, are provided with the actual public keys generated within $\mathcal{F}_{\text{crypto}}$ (the corresponding private keys remain in $\mathcal{F}_{\text{crypto}}$), they do not get their hands on the actual symmetric keys stored in the functionality, but only on pointers to these keys, since otherwise no security guarantees could be provided. These pointers may be part of the messages given to $\mathcal{F}_{\text{crypto}}$ for encryption. Before a message is actually encrypted, the pointers are replaced by the keys they refer to. Upon decryption of a ciphertext, keys embedded in the plaintext are first turned into pointers before the plaintext is given to the user. In order to be able to identify pointers/keys, we assume pointers/keys in plaintexts to be tagged according to their types. We speak of *well-tagged* messages. For real-world protocols, including those mentioned in the introduction, it is typically possible to define tagging in such a way that the message formats used in these protocols is captured precisely on the bit level, as demonstrated by our case study in 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 symmetric keys (via pointers) and which keys are known to the environment/adversary, i.e., have been corrupted or have been encrypted under a known key, and as a result became known. For this purpose, among others, $\mathcal{F}_{\text{crypto}}$ maintains a set $\mathcal{K}$ of all symmetric keys stored within $\mathcal{F}_{\text{crypto}}$, a set $\mathcal{K}_{\text{known}} \subseteq \mathcal{K}$ of *known* keys, and a set $\mathcal{K}_{\text{unknown}} := \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ of unknown keys.

Before any cryptographic operation can be performed, $\mathcal{F}_{\text{crypto}}$ expects to receive (descriptions of) algorithms from the ideal adversary for symmetric and public-key encryption/decryption as well as the generation and verification of MACs and digital signatures. Also, $\mathcal{F}_{\text{crypto}}$ expects to receive public/private key pairs 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 and keys; all security guarantees that $\mathcal{F}_{\text{crypto}}$ provides are made explicit within $\mathcal{F}_{\text{crypto}}$ without relying on specific properties of these algorithms. As a result, when using $\mathcal{F}_{\text{crypto}}$ in the analysis of systems, one can abstract from these algorithms entirely. We now sketch the operations that $\mathcal{F}_{\text{crypto}}$ provides.

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

corrupted, before adding k to $\mathcal{K}_{\text{known}}$, $\mathcal{F}_{\text{crypto}}$ verifies that k does not belong to $\mathcal{K}_{\text{unknown}}$. If $\mathcal{F}_{\text{crypto}}$ accepts k, a new pointer ptr to k is created (by increasing a counter) and returned to $(p, lsid, r)$. 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 (it provides these values anyway). The point is that operations performed with unknown keys are ideal (see below). In the realization of $\mathcal{F}_{\text{crypto}}$, however, keys in $\mathcal{K}_{\text{unknown}}$ will of course not be known to the adversary.

Establishing pre-shared keys [$(\mathsf{GetPSK}, t, name)$]**.** This request is similar to (New, t). However, if $\mathcal{F}_{\text{crypto}}$ already recorded a key under $(t, name)$, a new pointer to this key is returned. In particular, if different users invoke this command with the same name and type, they are provided with pointers to the same key. This allows users to establish shared keys: For example, for WPA (see Section 5), requests of suppliers (e.g., laptops) and authenticators (e.g., access points) are of the form $(\mathsf{GetPSK}, t, kid)$, where kid is a key ID (instances of) suppliers and authenticators obtain from the environment (e.g., a system administrator) upon initialization.

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

Encryption [(Enc, ptr, x)] **and decryption** [(Dec, ptr, y)]**.** We concentrate on authenticated encryption and decryption (see [26] for unauthenticated and public-key encryption and decryption). A user $(p, lsid, r)$ can ask to encrypt a well-tagged message x using a pointer ptr that has to belong to the user and points to a key, say k, of type $\mathsf{authenc\text{-}key}$. We first consider the case that $k \in \mathcal{K}_{\text{unknown}}$. First, all pointers in x, which again have to belong to the user, are replaced by the actual keys, resulting in a message x'. Then, the *leakage* $\overline{x} = L(1^\eta, x')$ of x' is encrypted under k using the encryption algorithm previously provided by the adversary (see above). The resulting ciphertext y (if any) is returned to the user and (x', y) is stored by $\mathcal{F}_{\text{crypto}}$ for later decryption of y under k. Decryption of a ciphertext y, an arbitrary bit string, under a key k (as above), in fact only succeeds if for y exactly one pair of the form (x', y) is stored in $\mathcal{F}_{\text{crypto}}$. If $k \in \mathcal{K}_{\text{known}}$, the encryption and decryption algorithms provided by the adversary are applied to x' (rather than to $\overline{x} = L(1^\eta, x')$) and y, respectively.

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

For verification, $\mathcal{F}_{\text{crypto}}$ runs the MAC verification algorithm previously provided by the adversary on x, σ, and k. If $k \in \mathcal{K}_{\text{known}}$, $\mathcal{F}_{\text{crypto}}$ returns the result of the verification

to the user. If $k \in \mathcal{K}_{\text{unknown}}$, this is done too, but success is only returned if x previously has been recorded for k.

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

Further operations. For further operations, including computing and verifying digital signatures, requests to obtain public keys, storing and retrieving of symmetric keys, checking the corruption status of keys, and checking whether two pointers point to the same key, we refer the reader to [26].

As illustrated by our case study, $\mathcal{F}_{\text{crypto}}$ is a convenient and easy to use tool for analyzing (real-world) security protocols. We note that, 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 (see Section 5 for an example).

3.2 Realizing the Ideal Crypto Functionality

Let $\Sigma_{\text{unauthenc}}$, Σ_{authenc}, 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 $F = \{F_\eta\}_{\eta\in\mathbb{N}}$ be a family of pseudo-random functions with $F_\eta \colon \{0,1\}^\eta \times \{0,1\}^* \to \{0,1\}^\eta$ for all $\eta \in \mathbb{N}$. For simplicity of presentation, we assume keys to be chosen uniformly at random from $\{0,1\}^\eta$.

These schemes induce a realization $\mathcal{P}_{\text{crypto}}$ of $\mathcal{F}_{\text{crypto}}$ in the obvious way: The realization $\mathcal{P}_{\text{crypto}}$ maintains keys and pointers to keys in the same way as $\mathcal{F}_{\text{crypto}}$ does, but it does not maintain the sets $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$. However, it is recorded whether a key is corrupted. Uncorrupted keys are honestly generated within $\mathcal{P}_{\text{crypto}}$ whereas corrupted keys are provided by the adversary. All ideal operations are replaced by their real counterparts in the natural way. Key derivation for a key k and a seed s is realized by computing F_η on k and s.

One cannot prove that $\mathcal{P}_{\text{crypto}}$ realizes $\mathcal{F}_{\text{crypto}}$ for standard assumptions about the symmetric encryption schemes $\Sigma_{\text{unauthenc}}$ and Σ_{authenc}, namely IND-CCA security and authenticated encryption (IND-CPA and INT-CTXT security), respectively, because it is easy to see that such a theorem does not hold in the presence of environments that may produce so-called key cycles (see, e.g., [6, 2]) or cause the so-called commitment problem (see, e.g., [2]). Therefore, similar to [25] and [2], we restrict the class of environments that we consider basically to those environments that do not produce key cycles or cause the commitment problem. More precisely, to formulate such a class of environments that captures what is typically encountered in applications, we observe, as was first pointed in [2], 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*; for example, WPA can trivially be seen to be standard (see Section 5). This observation can be generalized to *used-order respecting environments*, which we formulate based on $\mathcal{F}_{\text{crypto}}$: 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 $\mathcal{E} \mid \mathcal{F}_{\text{crypto}}$, an unknown key k (i.e., k is marked unknown in $\mathcal{F}_{\text{crypto}}$) which has been used at some point (for encryption or key derivation, in case of keys of type unauthenc-key

also for decryption) 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.) 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.

We can now state the theorem, which shows that $\mathcal{F}_{\text{crypto}}$ *exactly* captures IND-CCA security, authenticated encryption, and UF-CMA security. In the theorem, instead of explicitly restricting the class of environments introduced above, we use 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.

Theorem 3. *Let $\Sigma_{\text{unauthenc}}, \Sigma_{\text{authenc}}, \Sigma_{\text{pub}}$ 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 digital 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, in the proof of the above theorem we need to carry out a single hybrid argument, intertwining both encryption and key derivation (see [26] for details).

The following corollary shows that if a protocol system $\mathcal{P}$ that uses $\mathcal{F}_{\text{crypto}}$ is *non-committing* and *used-order respecting*, i.e., $\mathcal{E}|\mathcal{P}$ is a *non-committing, used-order respecting* environment for all environment systems $\mathcal{E}$, then $\mathcal{F}^*$ can be omitted. As mentioned above, most protocols, including standard protocols, have this property and this can typically be easily checked by inspection of the protocol (see Section 5 for an example).

Corollary 1. *Let $\Sigma_{\text{unauthenc}}$, Σ_{authenc}, 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, using 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 [25] 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 multi-session case by joint state theorems [13, 23, 25]. 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 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., [10], 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 parametrized by a number n which specifies the number of roles, 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 $r \in \{1, \ldots, n\}$. 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)*, which is meant to realize $\mathcal{F}_{\text{ke}}$, is a protocol system $\mathcal{P}$ which connects to the I/O interface of $\mathcal{F}_{\text{crypto}}$ such that $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ has the same I/O interface as $\mathcal{F}_{\text{ke}}$. The system $\mathcal{P}$ is of the form $\mathcal{P} = !M_1 \mid \ldots \mid !M_n$ for some n and machines (IITMs) $M_1, \ldots, M_n$. For every user $(p, lsid, r)$, there is one instance of M_r; intuitively, such an instance is meant to realize a local session in $\mathcal{F}_{\text{ke}}$. Every instance of M_r may arbitrarily communicate with the adversary (the network) and may use $\mathcal{F}_{\text{crypto}}$ in the name of the corresponding user.[1] Analogously to $\mathcal{F}_{\text{ke}}$, a user $(p, lsid, r)$ initiates a key exchange by sending a *session-start* message to (its 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 its user 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 $\mathcal{P}$ in which, similar to $\mathcal{F}_{\text{ke}}$, the actual session key is given to the user). This instance then provides its user 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 then 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

[1] We note that an environment of $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ cannot directly access the I/O interface of $\mathcal{F}_{\text{crypto}}$, but only via the IITMs $M_1, \ldots, M_n$.

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 instances of M_r (this corresponds to the environment asking $\mathcal{F}_{\text{ke}}$ whether a local session is corrupted), ii) once an instance of M_r is corrupted, it stays corrupted, and iii) an instance of M_r 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 $\mathcal{F}_{\text{crypto}}$. We note that the protocol $\widehat{\mathcal{P}}$ is 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).

We note that in [26] we consider a more general form of $\mathcal{F}_{\text{crypto}}$-KE protocols: We allow $\mathcal{P}$ and $\widehat{\mathcal{P}}$ to use (arbitrary) ideal functionalities $\mathcal{F}_1 \mid \ldots \mid \mathcal{F}_l$ in addition to $\mathcal{F}_{\text{crypto}}$. These functionalities can provide additional cryptographic operations, such as public-key certification. As shown in [26], our criteria and all results obtained in this paper remain unchanged and carry over to these generalized $\mathcal{F}_{\text{crypto}}$-KE protocols.

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.[2]

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) *if there exists a valid partnering function τ for $\mathcal{P}$ such that for every environment $\mathcal{E}$ for $\mathcal{P} \mid \mathcal{F}_{\text{crypto}}$ the following holds with overwhelming probability, where*

[2] We note that partnering functions have been used in game-based security definitions (e.g., [4]). However, their use has been criticized in subsequent work (e.g., [3, 21]). We emphasize that here, partnering functions are only part of our criterion but not part of the security definition.

the probability is over runs of $\mathcal{E}|\mathcal{P}|\mathcal{F}_{\text{crypto}}$: 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).

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 $\widehat{\mathcal{P}}|\mathcal{P}_{\text{crypto}} \leq \mathcal{F}_{\text{ke}}$.*

4.2 Applications to Secure Channels

A secure channel, see, e.g., [12], 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 only briefly sketch our results; see [26] for details.

We 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 (using authenticated encryption) under the session key, the PIDs of the sender and receiver are added to the plaintexts as well as a counter.

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. While we could use "strongly $\mathcal{F}_{\text{crypto}}$-secure" as our criterion, a weaker criterion in fact 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), i.e., condition iii) in Definition 2 is dropped. But then, messages encrypted under these keys in the key exchange protocol should not interfere with messages sent over the secure channel. Instead of condition iii), we therefore consider a set α of messages and require that only messages in α are encrypted under the session key in the key exchange protocol. We note that strongly $\mathcal{F}_{\text{crypto}}$-secure protocols are $\emptyset$-$\mathcal{F}_{\text{crypto}}$-secure.

The following theorem states that α-$\mathcal{F}_{\text{crypto}}$-security is a sufficient condition for the generic secure channels protocols to realize the ideal secure channel functionalities, provided that plaintexts sent over the secure channel do not belong to α. Usually, the key exchange and the secure channel protocol use different message formats such that the messages cannot be confused, e.g., because of tagging with different protocol identifiers. In this case, an appropriate α can easily be defined.

Theorem 5. *Let $\mathcal{P}$ be an $\mathcal{F}_{\text{crypto}}$-KE protocol and α be a set of messages as above such that it does not contain any plaintext that is potentially encrypted by $\mathcal{P}_{\text{sc}}$ (or $\mathcal{P}_{\text{sc}}^{+}$). If $\mathcal{P}$ is α-$\mathcal{F}_{\text{crypto}}$-secure, then $\mathcal{P}_{\text{sc}} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$ and $\mathcal{P}_{\text{sc}}^{+} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}^{+}$.*

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) [20], namely the 4-Way Handshake (4WHS) protocol and the CCM Protocol (CCMP), with more details provided in [26]. We 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 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).

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

1. $A \to S$: p_A, n_A, c_1
2. $S \to A$: $p_S, n_S, c_2, \mathrm{MAC}_{\mathrm{KCK}}(n_S, c_2)$
3. $A \to S$: $p_A, n_A, c_3, \mathrm{MAC}_{\mathrm{KCK}}(n_A, c_3)$
4. $S \to A$: $p_S, c_4, \mathrm{MAC}_{\mathrm{KCK}}(c_4)$

Fig. 1. The 4-Way Handshake Protocol of IEEE 802.11i.

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 [17], and our security proofs would fail.

The adversary can (statically) corrupt an instance of A or S, i.e., a local session, by sending a special corrupt message to it. This has to be the first message this instance receives from the adversary. A corrupted instance grants the adversary full control over its interface, including the interface it has to $\mathcal{F}_{\mathrm{crypto}}$. If the instance is corrupted, all keys it has should be corrupted as well. We therefore require that the adversary corrupts all keys a corrupted instance creates using $\mathcal{F}_{\mathrm{crypto}}$. A corrupted instance always checks (by asking $\mathcal{F}_{\mathrm{crypto}}$) if its keys created in $\mathcal{F}_{\mathrm{crypto}}$ indeed have been corrupted by the adversary and terminates if they have not been corrupted. Note that since keys in $\mathcal{F}_{\mathrm{crypto}}$ of a corrupted instance are known, it is not a problem if the adversary generates key cycles or causes the commit problem with those keys. Conversely, uncorrupted instances always check that the key, PSK, and the nonce, n_A or n_S, they have created using $\mathcal{F}_{\mathrm{crypto}}$ are uncorrupted at the time of their creation.

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.

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

Theorem 6. *The protocol* 4WHS *is strongly* $\mathcal{F}_{\mathrm{crypto}}$*-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 $\mathcal{F}_{\mathrm{crypto}}$ 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.

It remains to show that 4WHS is strongly $\mathcal{F}_{\mathrm{crypto}}$-secure w.r.t. τ and every environment $\mathcal{E}$ of $4\mathrm{WHS} \mid \mathcal{F}_{\mathrm{crypto}}$: Let ρ be a run of $\mathcal{E} \mid 4\mathrm{WHS} \mid \mathcal{F}_{\mathrm{crypto}}$ 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 $\mathcal{F}_{\mathrm{crypto}}$, and which established the pre-shared key PSK and derived KCK and TK from it in $\mathcal{F}_{\mathrm{crypto}}$.

First, we observe that, by our corruption model, since $(p, lsid, r)$ is uncorrupted, PSK is uncorrupted (in $\mathcal{F}_{\mathrm{crypto}}$). 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 [17].) 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 $\mathcal{F}_{\text{crypto}}$. This key is of type authenc-key because, 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, $4\widehat{\text{WHS}}$ (recall that $4\widehat{\text{WHS}}$ outputs the session key instead of a pointer to it) is a standard protocol (as defined in Section 3), hence, it is used-order respecting and non-committing. Using Theorem 4 and 6, we immediately obtain that $4\widehat{\text{WHS}} \,|\, \mathcal{P}_{\text{crypto}}$ is a universally composable secure key exchange protocol.

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

5.2 The CCM Protocol

WPA2-PSK uses CCMP with the Temporal Key (TK), exchanged by running the 4WHS protocol, to establish a secure channel between the authenticator and the supplicant. CCMP can be modeled faithfully by $\mathcal{P}_{\text{sc}}$ (see Section 4.2). By Theorem 5 and 6 we obtain that CCMP using 4WHS and $\mathcal{F}_{\text{crypto}}$ is a universally composable secure channel protocol. Moreover, it is easy to see that CCMP | 4WHS is a standard protocol (as defined in Section 3), and hence, it is used-order respecting and non-committing. By Corollary 1, we then obtain:

Corollary 3. $\text{CCMP} \mid \text{4WHS} \mid \mathcal{F}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$ *and* $\text{CCMP} \mid \text{4WHS} \mid \mathcal{P}_{\text{crypto}} \leq \mathcal{F}_{\text{sc}}$.

6 Related Work

Backes et al. (see, e.g., [2]) 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 [27, 15, 1, 24]. 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 hands 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) [2], it is an open problem whether there is a reasonable realization; the original proof of the realization of the crypto library in [2] is flawed, as examples presented in [14] illustrate (see also [25]).

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. [31] (see also [25]). 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 [33]; 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. [18], 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: The Internet Key-Exchange (IKE) protocol (which is part of IPsec) was analyzed in [11]. (Fragments of) TLS were analyzed in [16, 28, 5], assuming session identifiers in ciphertexts [16] or the random oracle for key derivation [28, 5]. Cryptographic analysis of Kerberos was carried out for

example in [7], where 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, M. Dürmuth, and R. Küsters. On Simulatability Soundness and Mapping Soundness of Symbolic Cryptography. In *FSTTCS 2007*, volume 4855 of *LNCS*, pages 108–120. Springer, 2007.
2. M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *CSFW-17 2004*, pages 204–218. IEEE Computer Society, 2004.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.
4. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: The Three Party Case. In *STOC 1995*, pages 57–66. ACM, 1995.
5. K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Cryptographically Verified Implementations for TLS. In *CCS 2008*, pages 459–468. ACM, 2008.
6. J. Black, P. Rogaway, and T. Shrimpton. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In *SAC 2002*, volume 2595 of *LNCS*, pages 62–75. Springer, 2002.
7. B. Blanchet, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Computationally Sound Mechanized Proofs for Basic and Public-key Kerberos. In *ASIACCS 2008*, pages 87–99. ACM, 2008.
8. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.
9. R. Canetti. Universally Composable Signature, Certification, and Authentication. In *CSFW-17 2004*, pages 219–233. IEEE Computer Society, 2004.
10. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical Report 2000/067, Cryptology ePrint Archive, December 2005. `http://eprint.iacr.org/2000/067/`.
11. R. Canetti and H. Krawczyk. Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 143–161. Springer, 2002.
12. R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 337–351. Springer, 2002.
13. R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.
14. 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`.
15. V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *FSTTCS 2006*, volume 4337 of *LNCS*, pages 176–187. Springer, 2006.
16. S. Gajek, M. Manulis, O. Pereira, A. Sadeghi, and J. Schwenk. Universally Composable Security Analysis of TLS. In *ProvSec 2008*, volume 5324 of *LNCS*, pages 313–327. Springer, 2008.
17. C. He and J. C. Mitchell. Security Analysis and Improvements for IEEE 802.11i. In *NDSS 2005*. The Internet Society, 2005.
18. C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A Modular Correctness Proof of IEEE 802.11i and TLS. In *CCS 2005*, pages 2–15. ACM, 2005.

19. 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/.
20. 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.
21. K. Kobara, S. Shin, and M. Strefler. Partnership in key exchange protocols. In *ASIACCS 2009*, pages 161–170. ACM, 2009.
22. R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *CSFW-19 2006*, pages 309–320. IEEE Computer Society, 2006.
23. R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digitial Signature Functionalities with Local Computation. In *CSF 2008*, pages 270–284. IEEE Computer Society, 2008.
24. R. Küsters and M. Tuengerthal. Computational Soundness for Key Exchange Protocols with Symmetric Encryption. In *CCS 2009*, pages 91–100. ACM Press, 2009.
25. R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *CSF 2009*, pages 293–307. IEEE Computer Society, 2009.
26. R. Küsters and M. Tuengerthal. Ideal Key Derivation and Encryption in Simulation-based Security. Technical Report 2010/295, Cryptology ePrint Archive, 2010. http://eprint.iacr.org/2010/295/.
27. D. Micciancio and B. Warinschi. Soundness of Formal Encryption in the Presence of Active Adversaries. In *TCC 2004*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
28. P. Morrissey, N. P. Smart, and B. Warinschi. A Modular Security Analysis of the TLS Handshake Protocol. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 55–73. Springer, 2008.
29. T. Ohigashi and M. Morii. A Practical Message Falsification Attack on WPA. In *JWIS 2009*, 2009.
30. B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *S&P 2001*, pages 184–201. IEEE Computer Society, 2001.
31. A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Proofs of Computational Secrecy. In *ESORICS 2007*, volume 4734 of *LNCS*, pages 219–234. Springer, 2007.
32. E. Tews and M. Beck. Practical Attacks against WEP and WPA. In *WISEC 2009*, pages 79–86. ACM, 2009.
33. F. Zhang, J. Ma, and S. Moon. The Security Proof of a 4-Way Handshake Protocol in IEEE 802.11i. In *CIS 2005*, volume 3802 of *LNCS*, pages 488–493. Springer, 2005.