Universally Composable Symmetric Encryption

Ralf Küsters and Max Tuengerthal
University of Trier, Germany
{*kuesters,tuengerthal*}*@uni-trier.de*

***Abstract*—For most basic cryptographic tasks, such as public-key encryption, digital signatures, authentication, key exchange, and many other more sophisticated tasks, ideal functionalities have been formulated in the simulation-based security approach, along with their realizations. Surprisingly, however, no such functionality exists for symmetric encryption, except for a more abstract Dolev-Yao style functionality. In this paper, we fill this gap. We propose two functionalities for symmetric encryption, an unauthenticated and an authenticated version, and show that they can be implemented based on standard cryptographic assumptions for symmetric encryption schemes, namely IND-CCA security and authenticated encryption, respectively, provided that the environment does not create key cycles or cause the so-called commitment problem. We also illustrate the usefulness of our functionalities in applications, both in simulation-based and game-based security settings.**

I. Introduction

For most basic cryptographic tasks, such as public-key encryption, digital signatures, mutual authentication, and key exchange, ideal functionalities have been proposed and realized in the simulation-based security approach (see, e.g., [11], [13], [12], [18], [2], [25], [29]). There are also many functionalities for more sophisticated cryptographic tasks (see, e.g., [14] for an overview). Surprisingly, however, a functionality for symmetric encryption, similar to the one for public-key encryption, as first proposed by Canetti [11], is still missing; there only exists an abstract Dolev-Yao style functionality (see Section VI). Our main goal in this paper is therefore to come up with ideal functionalities for symmetric encryption which capture standard cryptographic assumptions on symmetric encryption schemes. Such functionalities would be very useful for the modular design and analysis of systems that employ symmetric encryption.

Compared to a functionality for public-key encryption, one faces several challenges when devising a functionality for symmetric key encryption: In case of public-key encryption, it is reasonable to assume that the private key never leaves the functionality, making it relatively easy to formulate and provide certain security guarantees. However, symmetric keys, in particular session keys, typically have to travel between parties, as, e.g., in Kerberos. But, of course, a symmetric encryption functionality cannot just give out these keys, because no security guarantees could be provided. So, clearly a user must not get his/her hands on these keys directly, but should only be able to refer to these keys by pointers [3]. A user should, for instance, be able to instruct the functionality to encrypt message m with the key corresponding to pointer ptr, where m itself may contain pointers to keys; these keys can then travel (securely) encapsulated in the ciphertext to m. This implies that an ideal symmetric encryption functionality has to keep track of who knows which keys and which keys have been revealed, e.g., due to corruption or encryption with a previously revealed key. The functionality also has to provide mechanisms for bootstrapping symmetric encryption. For example, by such a mechanism it should be possible to distribute symmetric keys using encryption under long-term pre-shared keys or public-key encryption. Finally, one has to deal with two technical problems: key cycles [23], [1] and the commitment problem [3], [16], [21]. A key cycle occurs if an encryption under k_1 depends on a key k_2 and vice versa, e.g., k is encrypted under itself. In the context of symmetric encryption, the commitment problem occurs in the simulation-based approach if a key is revealed after it was used to encrypt a message.

Contribution of this Paper. In this paper, we propose two variants of an ideal functionality for symmetric encryption, $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$. We show that $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ can be realized by an encryption scheme iff the encryption scheme is IND-CCA secure. We also prove equivalence between $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$ and an encryption scheme for authenticated encryption, i.e., an IND-CPA and INT-CTXT secure encryption scheme [6]. In both cases, we have to assume that the environment does not use these functionalities in such a way that a key cycle is produced or the commitment problem occurs. So, we circumvent these two problems by assuming appropriate environments. This approach was also taken by Backes and Pfitzmann [3], who already pointed out that key cycles and, assuming static corruption, the commitment problem typically do not occur in applications. So, assuming such environments seems to be justified for most applications.

The functionalities $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$ currently contain two mechanisms for bootstrapping symmetric encryption: (authenticated and unauthenticated) encryption with long-term symmetric keys as well as public-key encryption. These bootstrapping mechanisms are added to our functionalities in a modular way, by factoring them out in separate ideal functionalities, $\mathcal{F}_{\mathrm{ltsenc}}$ and $\mathcal{F}_{\mathrm{pke}}$. In this way, these "bootstrapping functionalities" can be realized separately and can easily be extended and replaced. For example, we consider

both an authenticated and an unauthenticated version of $\mathcal{F}_{\text{ltsenc}}$. Also, new bootstrapping mechanisms can be added by adding new functionalities.

The functionality $\mathcal{F}_{\text{ltsenc}}$, we propose, allows two parties to encrypt and decrypt messages in an ideal way under a (long-term) shared key. The functionality $\mathcal{F}_{\text{pke}}$ is standard (see, e.g., [13], [29], [17], [25]). It may be used by many parties to (ideally) encrypt messages under a public-key and by one party to decrypt such messages. We provide realizations for $\mathcal{F}_{\text{ltsenc}}$, both for the authenticated and the unauthenticated case, and joint state theorems; for $\mathcal{F}_{\text{pke}}$ this was done in [25]. The joint state theorems guarantee that in different protocol sessions the same long-term and public/private keys may be used, provided that session identifiers are added to plaintexts before encryption. At the same time it suffices to analyze only a single protocol session in order to get security guarantees in a scenario with multiple, concurrent sessions. We believe that the functionality $\mathcal{F}_{\text{ltsenc}}$ that we propose and the results shown for this functionality are of independent interest.

Our functionalities $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ have applications both in simulation- and game-based settings. In case of static corruption, they tremendously simplify the analysis and (modular) design of systems, e.g., cryptographic protocols, that employ symmetric encryption: the often involved reasoning about IND-CCA games for symmetric encryption as well as IND-CPA and INT-CTXT games is made superfluous; this reasoning is done once and for all in the proofs of the realizations of these functionalities. Using $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ more abstract and simpler information theoretic arguments now suffice. To illustrate these points, we use $\mathcal{F}_{\text{senc}}^{\text{auth}}$ to show that a variant of the Amended Needham-Schroeder Symmetric Key Protocol [28] realizes a key exchange functionality. We also employ both $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ to reprove and in some respects generalize theorems on key indistinguishability and key usability properties of so-called secretive protocols, introduced by Roy et al. [30], [31], in the game-based approach. While the proofs of these theorems were quite technical and involved, these theorems are now immediate corollaries of our main theorems, namely the realizations of $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$. We believe that our functionalities are also useful for establishing new computational soundness results for symmetric encryption. But we leave this as future work.

Our results are based on the recently proposed so-called IITM model for simulation-based security [24]. While being in the spirit of Canetti's UC model [13], it has certain advantages over the UC model, as demonstrated and discussed in [24], [25]. Particularly relevant for this work is that while a joint state theorem for public-key encryption was established in the IITM model, there are several problems with the joint state theorems in the UC model [25]. Putting these problems aside, the results presented here would, however, also carry over to the UC model.

Structure of this Paper. In the next section, we recall the computational model for simulation-based security that we use. The functionalities for bootstrapping symmetric encryption are presented in Section III. The functionalities, $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$, for symmetric encryption are then introduced in Section IV, along with their implementation. Applications are presented in Section V. Related work is discussed in Section VI. Full definitions and proofs can be found in our technical report [26].

II. Simulation-based Security

In this section, we recall the IITM model for simulation-based security [24]. Based on a relatively simple, but expressive general computational model, in which so-called IITMs (inexhaustible interactive Turing machines) and systems of IITMs are defined, simulation-based security notions are formalized, and general composition theorems can be proven. We point the reader to [24] for details of the IITM model.

A. The General Computational Model

Our 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 input and output tapes. Each tape has a name associated with it. These names determine how IITMs are connected in a system of IITMs. If an IITM sends a message on an output tape named c, then only an IITM with an input tape named c can receive this message. An IITM is activated with one message on one of its input tapes and it writes at most one output message per activation on one of the output tapes. The runtime of the IITM *per activation* is polynomially bounded in the security parameter, the current input, and the size of the current configuration. This allows the IITM to "scan" the complete incoming message and its complete current configuration, and to react to all incoming messages, no matter how often the IITM is activated. In particular, an IITM cannot be exhausted, i.e., forced to stop (therefore the name *inexhaustible* interactive Turing machine).[1] An IITM runs in one of two modes, CheckAddress (deterministic computation) and Compute (probabilistic computation). The CheckAddress mode will be used to address different copies of IITMs in a system of IITMs (see below). This generic and flexible addressing mechanism avoids to fix up-front details of how an IITM is addressed. Such details are rather left to the specification (of the CheckAddress mode) of the IITM itself.

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

[1] This is not so in the UC model, which causes the joint state theorem to fail in that model, see [25].

say that the machines M'_j, $j \in \{1, \ldots, k'\}$, are in the scope of a bang. In a run of $\mathcal{S}$ there may be an unbounded number of copies of these machines, but only at most one copy of the machines M_i, $i \in \{1, \ldots, k\}$. We will consider so-called *well-formed systems*, which, due to a syntactical condition, are guaranteed to run in polynomial time [24].

In a run of $\mathcal{S}$ at every 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. The currently active machine may write at most one message, say m, on one of its output tapes, say c. This message is then delivered to an IITM with an input tape named c. By definition of a system, at most one of the machines $M_1, \ldots, M_k, M'_1, \ldots, M'_{k'}$ in the specification of $\mathcal{S}$ have an input tape named c, say M'_i has such a tape. Now, since M'_i is in the scope of a bang, in the run performed so far, there may be several copies of M'_i. In the order in which these copies were generated in the run, these copies are run in mode CheckAddress. The first of these copies to accept m will then get to process m (in mode Compute). If no such copy accepts m, a new copy of M'_i is generated (with fresh random coins) and it is checked whether it accepts m. If yes, this copy gets to process m (in mode Compute). Otherwise, the copy is deleted and the master IITM in $\mathcal{S}$ is activated. The master IITM is also activated if the currently active IITM does not produce output. A run stops if the master IITM after being activated 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.

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

Given an IITM M, we will often use its *identifier (ID) version* $\underline{M}$ to be able to address multiple copies of M. The identifier version $\underline{M}$ of M is an IITM which simulates M and acts as a "wrapper" around M. The wrapper requires that all messages received have to be prefixed by a particular ID, e.g., a session ID (SID) or party ID (PID); other messages will be rejected in the CheckAddress mode. Before giving a message to M, the wrapper strips off the ID. Messages sent out by M are prefixed with this ID by the wrapper. The ID that $\underline{M}$ will use is the one with which $\underline{M}$ was first activated. We often refer to $\underline{M}$ by *session version* or *party version* of M if the ID is meant to be a SID or PID, respectively. For example, if M specifies an ideal functionality, then $!\underline{M}$ denotes a system which can have an unbounded number of copies of $\underline{M}$, all with different SIDs. Of course an ID version $\underline{\underline{M}}$ of $\underline{M}$ can be considered, in which a copy of M is effectively addressed by a tuple of two IDs, e.g., an SID and a PID. Clearly, this can be iterated further. Given a system $\mathcal{S}$, its *identifier (ID) version* $\underline{\mathcal{S}}$ is obtained by replacing all IITMs in $\mathcal{S}$ by their ID version. For example, we obtain $\underline{\mathcal{S}} = \underline{M} \mid !\underline{M'}$ for $\mathcal{S} = M \mid !M'$.

B. 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*. We group these tapes into *I/O* and *network tapes*. We consider three different types of systems, modeling real/ideal protocols/functionalities, adversaries/simulators, and environments, respectively: *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 [24].

Definition 1 (Strong Simulatability)**.** Let $\mathcal{P}$ and $\mathcal{F}$ be well-formed 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), which may only connect to the network interface of $\mathcal{F}$, 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 $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$.

We emphasize that in this definition, no specific addressing or corruption mechanism is fixed. This can be defined in a rigorous, convenient, and flexible way as part of the specifications of $\mathcal{P}$ and $\mathcal{F}$.

C. Composition Theorems

We restate the composition theorems from [24]. The first composition theorem handles concurrent composition of a fixed number of protocol systems. The second one guarantees secure composition of an unbounded number of copies of a protocol system.

Theorem 1. *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, $\mathcal{P}_1 \mid \mathcal{P}_2 \leq \mathcal{F}_1 \mid \mathcal{F}_2$.*

Theorem 2. *Let $\mathcal{P}$ and $\mathcal{F}$ be well-formed protocol systems such that $\mathcal{P} \leq \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq !\underline{\mathcal{F}}$.*

Recall that in the above theorem $\underline{\mathcal{P}}$ and $\underline{\mathcal{F}}$ are the session versions of $\mathcal{P}$ and $\mathcal{F}$, respectively. The session versions allow to address the different copies of $\mathcal{F}$ and $\mathcal{P}$.

Theorems 1 and 2 can be applied iteratively, to get more and more complex systems. For example, using that $\leq$ is reflexive, we obtain the following corollary.

Corollary 1. *Let $\mathcal{Q}, \mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{Q} \mid !\underline{\mathcal{P}}$ and $\mathcal{Q} \mid !\underline{\mathcal{F}}$ are well-formed and $\mathcal{Q}$ only connects to the I/O interface of $!\underline{\mathcal{P}}$ and $!\underline{\mathcal{F}}$. Then, $\mathcal{P} \leq \mathcal{F}$ implies*

$\mathcal{Q} \mid !\underline{\mathcal{P}} \leq \mathcal{Q} \mid !\underline{\mathcal{F}}$, *i.e., $\mathcal{Q}$ using an unbounded number of copies of $\underline{\mathcal{P}}$ realizes $\mathcal{Q}$ using an unbounded number of copies of $\underline{\mathcal{F}}$.*

We say that a protocol $\mathcal{P}$ is a *sub-protocol* of $\mathcal{Q}$ if $\mathcal{Q}$ shields $\mathcal{P}$ from the environment, i.e., $\mathcal{Q}$ connects to the full I/O interface of $\mathcal{P}$, disallowing the environment or other protocols to connect to this I/O interface.

The above corollary is in the spirit of the UC composition theorem. Unlike that theorem, we, however, do not required that $\mathcal{P}$ is a sub-protocol of $\mathcal{Q}$. This may yield simpler and potentially more efficient implementations, e.g., in case of global setups, for which the UC model had to be extended [15].

III. Bootstrapping Symmetric Key Encryption

In this section, we briefly describe the two functionalities $\mathcal{F}_{\mathrm{pke}}$ and $\mathcal{F}_{\mathrm{ltsenc}}$ for bootstrapping symmetric key encryption, as mentioned in the introduction. Both functionalities can handle an unbounded number of encryption and decryption requests. Messages and ciphertexts are arbitrary bit strings of arbitrary length. It is up to the user how to interpret these bit strings, e.g., as payload data, party names, nonces, ciphertexts (including ciphertexts previously generated by the functionalities), digital signatures, hash values, non-interactive zero-knowledge proofs, etc.

The functionalities take as a parameter what we call a *leakage algorithm* L: a probabilistic polynomial time algorithm which takes a security parameter η and a message m as input 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. We call a leakage algorithm L *length preserving* if $\Pr[|L(1^\eta, m)| = |m|] = 1$ for all η and m.

A. The Public-Key Encryption Functionality

We use the functionality $\mathcal{F}_{\mathrm{pke}}$ proposed in [25] for public-key encryption. This functionality can be used by one decryptor and arbitrarily many encryptors. It provides ideal encryption and decryption w.r.t. one public/private key pair. The encryptors can invoke the functionality to ideally encrypt messages (arbitrary bit strings) under the public-key associated with the functionality. The decryptor can invoke the functionality to decrypt ciphertexts under the corresponding private key. A more detailed description of the functionality can be found in [25]. The flavor of this functionality is similar to the functionality $\mathcal{F}_{\mathrm{ltsenc}}$ described below.

As shown in [25], a public key encryption scheme realizes $\mathcal{F}_{\mathrm{pke}}$ iff it is IND-CCA secure. Moreover, it is shown that there is a joint state realization of $\mathcal{F}_{\mathrm{pke}}$, which distinguishes $\mathcal{F}_{\mathrm{pke}}$ from other functionalities for public-key encryption.

The functionalities $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$ for symmetric encryption under short-term keys will use the multi-party version $!\mathcal{F}_{\mathrm{pke}}$ of $\mathcal{F}_{\mathrm{pke}}$ as part of their bootstrapping mechanism. The multi-party version provides public-key encryption functionalities for an unbounded number of public/private key pairs.

B. The Symmetric Encryption Functionality with Long-Term Keys

The functionality $\mathcal{F}_{\mathrm{ltsenc}}$ allows two parties to establish a shared symmetric key and to encrypt and decrypt messages in an ideal way using this key. The key is meant to model a long-term shared key which is never given to the parties, but rather stays in the functionality. We consider an authenticated and an unauthenticated version of $\mathcal{F}_{\mathrm{ltsenc}}$, denoted $\mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{auth}}$ and $\mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{unauth}}$, respectively. In the following description of $\mathcal{F}_{\mathrm{ltsenc}}$, we point out the differences between the two versions of $\mathcal{F}_{\mathrm{ltsenc}}$ (see [26] for full details):

Initialization: Each party declares that it is willing to exchange a key with the other party. This information is forwarded to the (ideal) adversary who is required to provide encryption and decryption algorithms, enc and dec (which implicitly contain the long-term symmetric key). These algorithms are later applied to process encryption and decryption requests without further involvement of the adversary. Upon providing the algorithms, the adversary also decides whether or not it wants to corrupt the functionality (static corruption).

We note that no restrictions at all are put on the encryption and decryption algorithms provided by the adversary. All security guarantees that $\mathcal{F}_{\mathrm{ltsenc}}$ provides are made explicit in the description of $\mathcal{F}_{\mathrm{ltsenc}}$ in a rather syntactic way, without relying on (semantic) properties of these algorithms. As a result, when using $\mathcal{F}_{\mathrm{ltsenc}}$ in the analysis of more complex systems, one can completely abstract from these algorithms. The same is true for our formulations of $\mathcal{F}_{\mathrm{pke}}$ and $\mathcal{F}_{\mathrm{senc}}$.

Encryption request: If the functionality is requested by one of the two parties to encrypt a message m (which may be an arbitrary bit string), it will, in case the functionality is not corrupted, encrypt the leakage $L(1^\eta, m)$ of m instead of m itself, using enc. This results in some ciphertext c. The functionality then stores the pair (m, c) and returns c to the calling party. Note that, by construction, c leaks at most $L(1^\eta, m)$ (e.g., the length of m).

In case the functionality is corrupted, the message m (not its leakage) is encrypted using enc and the resulting ciphertext is returned to the calling party. There is no need to store the ciphertext. In other words, in the corrupted case, the functionality does not provide security guarantees.

Decryption request: Upon a decryption request by one of the two parties for a ciphertext c (which again may be an arbitrary bit string), an uncorrupted functionality performs the following actions: If the functionality has stored exactly one pair (m, c) for some plaintext m, this plaintext is returned. In case there is more than one such pair, an error is returned since no unique decryption is possible. If there

is none such pair, the following is done: In the authenticated variant of $\mathcal{F}_{\text{ltsenc}}$ ($\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$), which is supposed to model authenticated encryption, an error message is returned since it should not be possible to generate valid ciphertexts outside of $\mathcal{F}_{\text{ltsenc}}$. In the unauthenticated variant of $\mathcal{F}_{\text{ltsenc}}$ ($\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$), c is decrypted with the decryption algorithm dec and the result is returned to the calling party.

In case the functionality is corrupted, c is decrypted using dec and the result is returned to the calling party.

Corruption?: The environment can ask whether or not the functionality is corrupted. This capability is necessary for the functionality to make sense. Otherwise the functionality would be realizable with any (even insecure) encryption scheme as a simulator could simply provide the encryption and decryption algorithms of the encryption scheme and then corrupt the functionality. However, if the environment can ask for the corruption status, the simulator may only corrupt the functionality if its realization is corrupted as well.

We point out that corruption as defined above models that no ideal encryption/decryption is guaranteed, i.e., the functionality simply encrypts and decrypts messages/ciphertexts using the algorithms provided by the adversary. It does not model that the party itself is corrupt or dishonest. The place to capture this kind of corruption is one layer above, in the specification of the protocol that uses the functionality. However, one could, alternatively, specify a corruption behavior for $\mathcal{F}_{\text{ltsenc}}$ that models that the adversary takes over the functionality completely. We have, for example, chosen this alternative in the specification of our public-key encryption functionality [25]. Which corruption behavior to chose is a matter of convenience and taste.

We also note that, similarly to $\mathcal{F}_{\text{pke}}$, since in $\mathcal{F}_{\text{ltsenc}}$ the adversary provides the encryption and decryption algorithms (supposedly with a symmetric key embedded), $\mathcal{F}_{\text{ltsenc}}$ does not model that the key is kept secret. The point is that if $\mathcal{F}_{\text{ltsenc}}$ is uncorrupted, i.e., has not been corrupted by the adversary in the initialization step, then messages encrypted using $\mathcal{F}_{\text{ltsenc}}$ are encrypted ideally. By this, the confidentiality of plaintexts is guaranteed. Of course, in a realization of $\mathcal{F}_{\text{ltsenc}}$, the symmetric key will be kept secret; it will never leave the realization.

The functionalities $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ for symmetric encryption under short-term keys will use the multi-party version $!\mathcal{F}_{\text{ltsenc}}$ of $\mathcal{F}_{\text{ltsenc}}$ as part of their bootstrapping mechanism. This multi-party version provides functionalities for symmetric encryption with long-term keys for an unbounded number of pairs of parties, with one instance of $\mathcal{F}_{\text{ltsenc}}$ per pair.

Realizing $\mathcal{F}_{\text{ltsenc}}$. We show that the unauthenticated and authenticated versions, $\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$, of $\mathcal{F}_{\text{ltsenc}}$ exactly capture standard notions of security for symmetric encryption schemes.

A symmetric encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ induces a realization $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ of $\mathcal{F}_{\text{ltsenc}}$. This realization relies on the ideal functionality $\mathcal{F}_{\text{keysetup}}(\text{gen})$, which provides pairs of parties with the same symmetric key, generated according to gen. The functionality $\mathcal{F}_{\text{keysetup}}(\text{gen})$ is considered to be a sub-protocol of $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ and does not have a network interface, and hence, the environment cannot interact with it. This models that long-term symmetric keys are, for example, manually stored on the respective systems of the parties or provided by smart cards. Alternatively, one could use a key exchange functionality $\mathcal{F}_{\text{ke}}$ [5], [18].

More precisely, $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ is specified in the following obvious way: In the initialization phase, $\mathcal{F}_{\text{keysetup}}(\text{gen})$ is used to establish a shared symmetric key between the two parties. In case of corruption, the symmetric key provided by the adversary is used instead of the one delivered by $\mathcal{F}_{\text{keysetup}}(\text{gen})$. Encryption and decryption requests for a plaintext m or a ciphertext c are locally answered by simply running the encryption and decryption algorithms of Σ on m and c, respectively, in the obvious way. No extra randomness or tags need to be added to the plaintexts and ciphertexts.

We obtain the following theorem, which states i) equivalence between a symmetric encryption scheme being IND-CCA secure and the realization of the unauthenticated version $\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$ of $\mathcal{F}_{\text{ltsenc}}$ and ii) equivalence between a symmetric encryption scheme being IND-CPA and INT-CTXT secure and the realization of the authenticated version $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ of $\mathcal{F}_{\text{ltsenc}}$. (We refer the reader to [26] for the definitions of IND-CCA, IND-CPA, and INT-CTXT security.) The theorem directly captures that the multi-party version $!\mathcal{P}_{\text{ltsenc}}(\Sigma)$ of $\mathcal{P}_{\text{ltsenc}}(\Sigma)$ using $\mathcal{F}_{\text{keysetup}}(\text{gen})$ realizes the multi-party version $!\mathcal{F}_{\text{ltsenc}}$ of $\mathcal{F}_{\text{ltsenc}}$. The proof of the following theorem is presented in [26].

Theorem 3. *Let $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ be a symmetric encryption scheme and L be a leakage algorithm, which leaks exactly the length of a message. Then, we obtain the following equivalences, where the directions from left to right hold for any length preserving leakage algorithm L.*

1. *Σ is IND-CCA secure iff*

$$!\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \;\leq\; !\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}(L) \;.$$

2. *Σ is IND-CPA and INT-CTXT secure iff*

$$!\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \;\leq\; !\mathcal{F}_{\text{ltsenc}}^{\text{auth}}(L) \;.$$

We obtain a multi-*session* version of $!\mathcal{F}_{\text{ltsenc}}(L)$ by applying '$\dot{-}$' and '!' to $!\mathcal{F}_{\text{ltsenc}}(L)$, resulting in the system $!(!\mathcal{F}_{\text{ltsenc}}(L))$, which is the same as $!\mathcal{F}_{\text{ltsenc}}(L)$. In every run of this system there is (at most) one instance of $\mathcal{F}_{\text{ltsenc}}(L)$ per session and pairs of parties. Using Theorem 2, we obtain as a direct consequence of the above theorem that $!\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, !\mathcal{F}_{\text{keysetup}}(\text{gen}) \leq !\mathcal{F}_{\text{ltsenc}}(L)$, i.e., the multi-session version of $\mathcal{F}_{\text{ltsenc}}(L)$ is realized by the multi-session version of $!\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen})$. However, this realization is impractical: If two parties use the functionality

in different sessions, they have to use freshly generated long-term keys for *each* session, since each session uses a new instance of $\mathcal{F}_{\text{keysetup}}(\text{gen})$. We therefore also prove a joint state theorem for $\mathcal{F}_{\text{ltsenc}}$, along the lines of the joint state theorem for public-key encryption [25]. The purpose of $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ is explained following the theorem.

Theorem 4. *For every leakage algorithm L we have that*

$$!\mathcal{P}^{\text{JS}}_{\text{ltsenc}} \,|\, !\mathcal{F}_{\text{ltsenc}}(L') \;\leq\; !\underline{\mathcal{F}_{\text{ltsenc}}(L)} \;,$$

where $L'(1^\eta,(sid,m)) = (sid, L(1^\eta,m))$ for all SIDs sid and messages m, and $!\mathcal{F}_{\text{ltsenc}}(L')$ is a sub-protocol of $!\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$. This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}^{\text{auth}}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$.

The idea behind this theorem is as follows, where for simplicity we ignore the leakage algorithms for now. In $!\underline{\mathcal{F}_{\text{ltsenc}}}$ one instance of $\mathcal{F}_{\text{ltsenc}}$ per session and pair of parties can be generated. In particular, every pair of parties (p_1,p_2) uses a new instance of $\mathcal{F}_{\text{ltsenc}}$ for every session. In the realization $!\mathcal{P}^{\text{JS}}_{\text{ltsenc}} \,|\, !\mathcal{F}_{\text{ltsenc}}$ there is only one instance of $\mathcal{F}_{\text{ltsenc}}$ per pair of parties. This instance of $\mathcal{F}_{\text{ltsenc}}$ handles all sessions of this pair. The purpose of $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ is to act as a multiplexer between these sessions; there is only one instance of $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ per pair of parties. The multiplexer $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$, say for the pair of parties (p_1,p_2), works as follows: If the instance of $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ for (p_1,p_2) receives an encryption request for message m (from p_1 or p_2) in a session with SID sid, then $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ prefixes sid to m and forwards (sid,m) to the instance of $\mathcal{F}_{\text{ltsenc}}$ associated with (p_1,p_2). If $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ receives a decryption request in session sid, $\mathcal{P}^{\text{JS}}_{\text{ltsenc}}$ first forwards the ciphertext to the instance of $\mathcal{F}_{\text{ltsenc}}$ associated with (p_1,p_2) and then checks whether the resulting plaintext is of the form (sid,m), for some m. Intuitively, prefixing messages by SIDs in this way guarantees that different sessions do not interfere, even though they are handled by one instance of $\mathcal{F}_{\text{ltsenc}}$.

Note that in the above theorem, L' leaks the SID. This is motivated by the fact that the SID is not considered to be secret and an adversary might be able to tell in which session a ciphertext was generated.

Together with Theorem 1 and Theorem 3, we immediately obtain from Theorem 4:

$$!\mathcal{P}^{\text{JS}}_{\text{ltsenc}} \,|\, !\mathcal{P}_{\text{ltsenc}}(\Sigma) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \;\leq\; !\underline{\mathcal{F}_{\text{ltsenc}}(L)} \;.$$

This realization is practical in the sense that two parties use the same long-term symmetric key across all sessions, as all these sessions use the same instance of $\mathcal{F}_{\text{keysetup}}(\text{gen})$.

The proof of Theorem 4, which can be found in [26], is similar to the one for public-key encryption [25].

IV. The Symmetric Key Encryption Functionality

In this section, we describe the ideal functionalities $\mathcal{F}^{\text{unauth}}_{\text{senc}}$ and $\mathcal{F}^{\text{auth}}_{\text{senc}}$ for (authenticated) symmetric encryption, along with their realizations (see [26] for full details). In what follows, we write $\mathcal{F}_{\text{senc}}$ to refer to both $\mathcal{F}^{\text{unauth}}_{\text{senc}}$ and $\mathcal{F}^{\text{auth}}_{\text{senc}}$.

Parties can use $\mathcal{F}_{\text{senc}}$ to generate short-term keys and to encrypt and decrypt messages and ciphertexts, respectively, in an ideal way under these keys. Based on the bootstrapping components described in Section III, $\mathcal{F}_{\text{senc}}$ can also be used for public key encryption ($\mathcal{F}_{\text{pke}}$) and encryption under long-term symmetric keys ($\mathcal{F}_{\text{ltsenc}}$). Just as for the functionalities presented in Section III, $\mathcal{F}_{\text{senc}}$ can handle an unbounded number of encryption and decryption requests, with messages/ciphertexts being arbitrary bit strings of arbitrary length.

The distinguishing feature of $\mathcal{F}_{\text{senc}}$ compared to $\mathcal{F}_{\text{ltsenc}}$ is that short-term keys may be part of the messages to be encrypted (under other short-term keys, long-term keys or public keys). This allows short-term keys to travel (securely). However, as already mentioned in the introduction, the users of $\mathcal{F}_{\text{senc}}$ (or its realization) do not get their hands on the actual short-term keys, but only on pointers to keys stored in the functionality, since otherwise no security guarantees could be provided. These pointers may be part of the messages given to $\mathcal{F}_{\text{senc}}$ for encryption. They take the form (Key, ptr), where Key is a tag and ptr is the actual pointer to a key. The tag is used to identify the bit string ptr as a pointer. Before a message is actually encrypted, the pointers are replaced by the keys they point to. Keys are written in the form (Key, k), where k is the actual key. Again, the tag Key is used to identify the bit string k as a key. Upon decryption of a ciphertext, keys embedded in the plaintext are first turned into pointers before the plaintext is given to the user.

We note that instead of using tags for pointers and keys, we could parametrize the functionalities with encoding and decoding functions, for identifying pointers in a message and turning them into keys, and vice versa. The security guarantees that $\mathcal{F}_{\text{senc}}$ provides are not affected by the details of the encoding. For example, in applications in which the positions of pointers/keys in a message are known, pointers and keys could be extracted without relying on tags.

We next describe $\mathcal{F}_{\text{senc}}$ and its realization in more detail.

A. The Ideal Functionality

The ideal functionality $\mathcal{F}_{\text{senc}}$ handles the key generation, encryption, and decryption requests of an unbounded number of parties. It also provides an interface to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, for bootstrapping symmetric key encryption (see Section III). Just as these two functionalities, $\mathcal{F}_{\text{senc}}$ is parametrized by a leakage algorithm L.

The functionality $\mathcal{F}_{\text{senc}}$ has to keep track of which party 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, $\mathcal{F}_{\text{senc}}$ maintains a set $\mathcal{K}$ of all short-term keys stored within the

functionality, a set $\mathcal{K}_{\text{known}} \subseteq \mathcal{K}$ of known keys, a set $\mathcal{K}_{\text{unknown}} := \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ of unknown keys, and a set of corrupted keys $\mathcal{K}_{\text{corrupt}} \subseteq \mathcal{K}_{\text{known}}$. A partial function key yields the key $key(p, ptr) \in \mathcal{K}$ pointer ptr points to for party p. For ideal encryption and decryption, a table $decTable(k)$ is kept for every key $k \in \mathcal{K}_{\text{unknown}}$. It records pairs of the form (m, c), for a ciphertext c and its plaintext m. With these data structures, $\mathcal{F}_{\text{senc}}$ works as follows; differences between $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ are pointed out where necessary.

Obtain encryption and decryption algorithm: Before encryption and decryption can be performed, $\mathcal{F}_{\text{senc}}$ expects to receive an encryption and decryption algorithm from the (ideal) adversary, say enc and dec, respectively. As in the case of $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{ltsenc}}$, we do not put any restrictions on these algorithms; all security guarantees that $\mathcal{F}_{\text{senc}}$ provides are made explicit within $\mathcal{F}_{\text{senc}}$ in a rather syntactic way, without relying on specific properties of these algorithms. As a result, when using $\mathcal{F}_{\text{senc}}$ in the analysis of more complex systems, one can completely abstract from these algorithms.

(Short-term) key generation: A party p can ask $\mathcal{F}_{\text{senc}}$ to generate a key. This request is forwarded to the adversary, who is expected to provide such a key, say k. The adversary can decide to corrupt k right away (static corruption), in which case k is added to $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{corrupt}}$. In case the key is not marked corrupted, the functionality $\mathcal{F}_{\text{senc}}$ only accepts k if k does not belong to $\mathcal{K}$, modeling that k is fresh. In case k is corrupted, k still may not belong to $\mathcal{K}_{\text{unknown}}$ (no key guessing). We emphasize that the difference between $\mathcal{K}_{\text{known}}$ and $\mathcal{K}_{\text{unknown}}$ is not whether or not an adversary knows the value of a key; the adversary knows this value anyway, since he provides these values in the ideal world. The point is that if $k \in \mathcal{K}_{\text{unknown}}$, messages encrypted under k will be encrypted ideally, i.e., the leakage of these messages is encrypted instead of the messages itself. Conversely, if $k \in \mathcal{K}_{\text{known}}$, the actual messages are encrypted under k. So, no security guarantees are provided in this case. In the realization of $\mathcal{F}_{\text{senc}}$, however, keys corresponding to keys in $\mathcal{K}_{\text{unknown}}$ will of course not be known by the adversary.

After the key k has been provided by the adversary, a pointer to this key is created for party p, if there does not exist such a pointer already, and this pointer is given to p. (The value of the pointer does not need to be secret. In fact, new pointers are created by increasing a counter.)

Key generation requests for public/private keys and long-term symmetric keys are simply forwarded to (instances of) $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{ltsenc}}$, respectively.

Encryption request: We first consider encryption with short-term keys. Such a request is of the form $(p, \mathsf{Enc}, ptr, m)$, where m is the message to be encrypted, p is the name of the party who wants to encrypt m, and ptr is a pointer to the key under which p wants to encrypt m. Upon such a request, $\mathcal{F}_{\text{senc}}$ first checks whether ptr is associated with a key, i.e., whether $k = key(p, ptr)$ is defined. Also, this is checked for all pointers (Key, ptr') in m. If these checks are successful, these pointers are replaced by their corresponding keys (Key, k'), resulting in a message m'. Then, if $k \in \mathcal{K}_{\text{unknown}}$, the *leakage* $L(1^\eta, m')$ of m' is encrypted under k using enc (the encryption algorithm provided by the adversary). If c denotes the resulting ciphertext, the pair (m', c) is added to $decTable(k)$ and c is given to p. If $k \in \mathcal{K}_{\text{known}}$, m' itself is encrypted, resulting in some ciphertext c. All keys in m' are then added to $\mathcal{K}_{\text{known}}$, as they have been encrypted under a known key. The ciphertext c is given to p.

Encryption requests for public key encryption and long-term symmetric key encryption are handled similarly. The main difference is that the encryption of m' is handled by (an instance of) $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{ltsenc}}$, respectively. If such an instance is corrupted (this can be checked by simply asking about the corruption status), the keys stored in m' are marked as known in $\mathcal{F}_{\text{senc}}$.

Decryption requests: For brevity, we only describe decryption requests under short-term keys here; the cases of public/private and long-term symmetric keys are handled similarly using (instances of) $\mathcal{F}_{\text{pke}}$ and $\mathcal{F}_{\text{ltsenc}}$, respectively (see [26] for full details). A decryption request for a short-term key is of the form $(p, \mathsf{Dec}, ptr, c)$, where c is a ciphertext, p is the name of the party who wants to decrypt c, and ptr is a pointer to the key with which p wants to decrypt c. Similarly to the case of encryption, it is first checked whether $k = key(p, ptr)$ is defined. Then, two cases are distinguished:

i) If $k \in \mathcal{K}_{\text{unknown}}$, it is checked whether there exists exactly one m' such that $(m', c) \in decTable(k)$. If so, the keys (Key, k') in m' are turned into pointers (Key, ptr') for p; for new keys, new pointers are generated. The resulting message m is given to p. If there is more than one m' with $(m', c) \in decTable(k)$, an error is returned, since unique decryption is not possible. If there is no such m', the following is done: In $\mathcal{F}_{\text{senc}}^{\text{auth}}$, an error message is returned, since for authenticated encryption it should not be possible to generate valid ciphertexts outside of the functionality. In $\mathcal{F}_{\text{senc}}^{\text{unauth}}$, c is decrypted under k with dec (the decryption algorithm provided by the adversary) and then the following is done $(*)$: If the resulting plaintext m' contains a key (Key, k') with $k' \in \mathcal{K}_{\text{unknown}}$, an error message is given to p, modeling that this should not happen (no key guessing). Otherwise, the keys (Key, k') in m' are turned into pointers (Key, ptr') for p; for new keys, new pointers are generated and these keys are marked as known. The resulting message m is given to p.

ii) If $k \in \mathcal{K}_{\text{known}}$, c is decrypted under k with dec and then $\mathcal{F}_{\text{senc}}$ proceeds as in $(*)$ above.

Store and reveal requests: A party p can ask $\mathcal{F}_{\text{senc}}$ to *store* some bit string k as a key. If k belongs to $\mathcal{K}_{\text{unknown}}$, $\mathcal{F}_{\text{senc}}$ will return an error message (no key guessing). Otherwise, $\mathcal{F}_{\text{senc}}$ creates a pointer to k for party p, if there

does not exist such a pointer already, and this pointer is given to p. (Note that there might already exist a pointer to k in $\mathcal{F}_{\mathrm{senc}}$ if $k \in \mathcal{K}_{\mathrm{known}}$.) The key k is added to $\mathcal{K}_{\mathrm{known}}$.

A party p can ask $\mathcal{F}_{\mathrm{senc}}$ to *reveal* the bit string corresponding to some pointer in which case $\mathcal{F}_{\mathrm{senc}}$ will return this bit string to p and add it to $\mathcal{K}_{\mathrm{known}}$.

Corrupted keys?: The environment can ask, for a party p and a pointer ptr, whether the corresponding key, if any, is corrupted, i.e., belongs to $\mathcal{K}_{\mathrm{corrupt}}$. Similar questions for public/private keys and long-term symmetric keys are forwarded by $\mathcal{F}_{\mathrm{senc}}$ to (instances of) $\mathcal{F}_{\mathrm{pke}}$ and $\mathcal{F}_{\mathrm{ltsenc}}$.

This concludes the description of $\mathcal{F}_{\mathrm{senc}}$. As explained for encryption requests, if a message m is encrypted under a known key (or by a corrupted $\mathcal{F}_{\mathrm{pke}}$ or $\mathcal{F}_{\mathrm{ltsenc}}$), then all keys in m are marked known in $\mathcal{F}_{\mathrm{senc}}$. Yet, if in an application the ciphertext c for m is encrypted again under an unknown key and c is *always* kept encrypted under an unknown key, the keys in m might not be revealed from the point of view of the application. While in such a case, $\mathcal{F}_{\mathrm{senc}}$ 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, for example, not aware of any authentication or key exchange protocol where this is not the case (see, e.g., [10] for a collection of such protocols). Second, if in a session of a protocol symmetric keys known to the adversary are used, then typically no security guarantees are provided for that session anyway.

B. The Realization

A symmetric encryption scheme $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ induces a realization $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ of $\mathcal{F}_{\mathrm{senc}}$ in the obvious way: Upon key generation, the adversary is asked whether he wants to corrupt the key, in which case he provides the key. Otherwise, the key is generated honestly within $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ using $\mathrm{gen}(1^\eta)$. Upon a request for encryption under a short-term key of the form $(p, \mathsf{Enc}, ptr, m)$, it is first checked whether there is a key k corresponding to ptr, then pointers in m are replaced by keys (just as in $\mathcal{F}_{\mathrm{senc}}$). Unlike $\mathcal{F}_{\mathrm{senc}}$, the resulting message, say m', is then encrypted under k by running enc, i.e., $\mathrm{enc}(k, m')$ is returned to p as ciphertext. (Note that no extra randomness or tagging is added.) Requests for encryption under long-term symmetric keys or public keys are forwarded by $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ to (instances of) $\mathcal{F}_{\mathrm{ltsenc}}$ and $\mathcal{F}_{\mathrm{pke}}$. Note that while $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ still makes use of these ideal functionalities, using the composition theorem, these functionalities can be replaced by their realizations (see also Section IV-C). Requests for decryption of ciphertexts under short-term keys, which are of the form $(p, \mathsf{Dec}, ptr, c)$, are answered by decrypting c under the key k corresponding to ptr (if any), i.e., $\mathrm{dec}(k, c)$ is computed. If the decryption is successful and returns the message m, then keys in m are replaced by pointers (with possibly new pointers generated). The resulting message is returned to p. Requests for decryption under long-term symmetric keys or public keys are forwarded to (instances) of $\mathcal{F}_{\mathrm{ltsenc}}$ and $\mathcal{F}_{\mathrm{pke}}$, respectively.

C. Main Results

We would like to prove that $\mathcal{P}_{\mathrm{senc}}(\Sigma)$ realizes $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$ for standard assumptions about the symmetric encryption scheme Σ, namely IND-CCA security and authenticated encryption (IND-CPA and INT-CTXT security), respectively. However, 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, as mentioned in the introduction: It is well-known that standard assumptions about symmetric encryption schemes are too weak to deal with key cycles [8], [4]. Recall that 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 by the simulator, i.e., the leakage of the message is encrypted by the simulator. After the key has been revealed, the simulator 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., [3]). As already mentioned in the introduction, similarly to [3], we therefore restrict the class of environments that we consider basically to those environments that do not produce key cycles or cause the commitment problem.

To formulate such a class of environments that captures what is typically encountered in applications, we observe, as was first pointed in [3], 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 $\mathcal{F}_{\mathrm{senc}}$: In what follows, we say that an unknown key k, i.e., $k \in \mathcal{K}_{\mathrm{unknown}}$, has been *used (for encryption)*, if $\mathcal{F}_{\mathrm{senc}}$ has been instructed to encrypt a message under k. Now, an environment is *used order respecting* if runs of the following form occur only with negligible probability: An unknown key k used for the first time at some point is encrypted itself by an unknown key k' used for the first time later than k. It is clear from this definition that used order respecting environments produce key cycles (among unknown keys) only with negligible probability. (We do not need to prevent key cycles among known keys.)

We say that an *environment does not cause the commitment problem*, if runs of the following form occur only with negligible probability: After an unknown key k has been used to encrypt a message, k does not become known later on in the run, i.e., is not added to $\mathcal{K}_{\mathrm{known}}$. Assuming static corruption of keys, it is easy to see that for standard

protocols, as introduced above, the commitment problem does not occur.

We can now state the main theorem of this paper, which shows that realizing $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ ($\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$) by a symmetric encryption scheme is *equivalent* to IND-CCA security (IND-CPA and INT-CTXT security) of that scheme. In other words, $\mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}$ exactly capture IND-CCA secure and authenticated encryption, respectively. 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 $\mathcal{F}_{\mathrm{senc}}$ (and hence, $\mathcal{P}_{\mathrm{senc}}$), but before forwarding requests to $\mathcal{F}_{\mathrm{senc}}$ 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. We need an unauthenticated and an authenticated version of $\mathcal{F}^*$, called $\mathcal{F}^*_{\mathrm{unauth}}$ and $\mathcal{F}^*_{\mathrm{auth}}$. The authenticated version, works as just described. For the unauthenticated version, in the definition of used order respecting environments, we consider an unknown key to have been used also if it has successfully been used to *decrypt* a message. Typical environments are used order respecting also with respect to this new interpretation of "used". For example, protocols are typically standard (see above) also with this interpretation of "used". The proof of the following theorem can be found in [26].

Theorem 5. *Let Σ be a symmetric encryption scheme, L be a leakage algorithm which leaks exactly the length of a message, and $\mathcal{F}_{\mathrm{bs}} = !\mathcal{F}_{\mathrm{ltsenc}}(L) \,|\, !\mathcal{F}_{\mathrm{pke}}(L)$ be the bootstrapping component. Then, for both $\mathcal{F}_{\mathrm{ltsenc}} = \mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{auth}}$ and $\mathcal{F}_{\mathrm{ltsenc}} = \mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{unauth}}$, we have that*

1. *Σ is IND-CCA secure iff*

$$\mathcal{F}^*_{\mathrm{unauth}} \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, \mathcal{F}_{\mathrm{bs}} \;\leq\; \mathcal{F}^*_{\mathrm{unauth}} \,|\, \mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}(L) \,|\, \mathcal{F}_{\mathrm{bs}}$$

   and

2. *Σ is IND-CPA and INT-CTXT secure iff*

$$\mathcal{F}^*_{\mathrm{auth}} \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, \mathcal{F}_{\mathrm{bs}} \;\leq\; \mathcal{F}^*_{\mathrm{auth}} \,|\, \mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}(L) \,|\, \mathcal{F}_{\mathrm{bs}} \;.$$

We remark that in the above theorem, $\mathcal{F}_{\mathrm{bs}}$ is a sub-protocol of $\mathcal{F}_{\mathrm{senc}}$ and $\mathcal{F}_{\mathrm{senc}}$ is a sub-protocol of $\mathcal{F}^*$.

We note that, by Theorems 1 and 2, the bootstrapping component $\mathcal{F}_{\mathrm{bs}}$ on the left-hand side of $\leq$ can be replaced by its realization, where for $\mathcal{F}_{\mathrm{ltsenc}}$ we can use the realization given in Theorem 3 and for $\mathcal{F}_{\mathrm{pke}}$ the one proposed in [25].

Theorem 5 yields the following corollary, which gets rid of the functionality $\mathcal{F}^*$, assuming that $\mathcal{F}_{\mathrm{senc}}$ is used by what we call a non-committing, used order respecting protocol. A protocol system $\mathcal{P}$ that uses $\mathcal{F}_{\mathrm{senc}}$ is called *non-committing, used order respecting* if the probability that in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\mathrm{senc}}(L) \,|\, \mathcal{F}_{\mathrm{bs}}$ 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, in case of static corruption, standard protocols (see above) are non-committing and used order respecting; see also Section V-A for an example.

Corollary 2. *Let Σ, L, and $\mathcal{F}_{\mathrm{bs}}$ be given as in Theorem 5. Let $\mathcal{P}$ be a non-committing, used order respecting protocol system. Then, for both $\mathcal{F}_{\mathrm{ltsenc}} = \mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{auth}}$ and $\mathcal{F}_{\mathrm{ltsenc}} = \mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{unauth}}$, we have that*

1. *if Σ is IND-CCA secure, then*

$$\mathcal{P} \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, \mathcal{F}_{\mathrm{bs}} \;\leq\; \mathcal{P} \,|\, \mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}(L) \,|\, \mathcal{F}_{\mathrm{bs}}$$

   and

2. *if Σ is IND-CPA and INT-CTXT secure, then*

$$\mathcal{P} \,|\, \mathcal{P}_{\mathrm{senc}}(\Sigma) \,|\, \mathcal{F}_{\mathrm{bs}} \;\leq\; \mathcal{P} \,|\, \mathcal{F}_{\mathrm{senc}}^{\mathrm{auth}}(L) \,|\, \mathcal{F}_{\mathrm{bs}} \;.$$

V. Applications

As mentioned in the introduction, $\mathcal{F}_{\mathrm{senc}}$ has applications both in the simulation- and game-based setting. We now illustrate the usefulness of $\mathcal{F}_{\mathrm{senc}}$ in these settings by two examples. In what follows, let $\mathcal{F}_{\mathrm{enc}} = \mathcal{F}_{\mathrm{senc}} \,|\, \mathcal{F}_{\mathrm{bs}}$, where $\mathcal{F}_{\mathrm{bs}}$ is the bootstrapping component (see Theorem 5). We write $\mathcal{F}_{\mathrm{enc}}^{\mathrm{unauth}}$, if in $\mathcal{F}_{\mathrm{enc}}$ we set $\mathcal{F}_{\mathrm{senc}} = \mathcal{F}_{\mathrm{senc}}^{\mathrm{unauth}}$ and $\mathcal{F}_{\mathrm{ltsenc}} = \mathcal{F}_{\mathrm{ltsenc}}^{\mathrm{unauth}}$; analogously for $\mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$. We write $\mathcal{P}_{\mathrm{enc}}$ for a realization of $\mathcal{F}_{\mathrm{enc}}$, as obtained in the previous sections. We write $\mathcal{P}_{\mathrm{enc}}^{\mathrm{unauth}}$ if $\mathcal{P}_{\mathrm{enc}}$ is based on an IND-CCA secure symmetric encryption scheme and $\mathcal{P}_{\mathrm{enc}}^{\mathrm{auth}}$ in case of authenticated encryption.

A. Simulation-based Analysis of a Key Exchange Protocol

In this section, we analyze a variant of the Amended Needham Schroeder Symmetric Key (ANSSK) protocol [28]. Compared to ANSSK, our variant, which we call ANSSK', is augmented by a short-term key; the analysis of the original ANSSK protocol is even simpler. The ANSSK' protocol is depicted in Figure 1. We show that this protocol realizes a key exchange functionality $\mathcal{F}_{\mathrm{ke}} = \mathcal{F}_{\mathrm{ke}}(\mathrm{gen})$ in case authenticated encryption is used. (In [26], we show that this is not true if the encryption scheme is merely IND-CCA secure.) Due to our main result (Section IV-C), it suffices to show that ANSSK' realizes $\mathcal{F}_{\mathrm{ke}}$ when encryption and decryption is performed based on $\mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$. As we will see, the use of $\mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$ tremendously simplifies the analysis. Also, we only need to analyze one protocol session. The composition theorems and the joint state theorems for the bootstrapping component then yield a practical realization for multiple sessions of $\mathcal{F}_{\mathrm{ke}}$, where session identifiers are added to plaintexts before encryption.

In what follows, let $\mathcal{F}_{\mathrm{ke}}$ be a key exchange functionality, as, for example, specified in [17]. This functionality describes one session of an ideal key exchange between two parties. It waits for key exchange requests from the two parties and if the (ideal) adversary sends a complete message for one party, this party receives the key generated within

1. $B \to A$: $\{A, Kb\}_{Kbs}$
2. $A \to S$: $A, B, Na, \{A, Kb\}_{Kbs}$
3. $S \to A$: $\{Na, B, Kab, \{Kab, A\}_{Kb}\}_{Kas}$
4. $A \to B$: $\{Kab, A\}_{Kb}$

Kas:	long-term key between A and S
Kbs:	long-term key between B and S
Kb:	short-term key generated by B
Na:	nonce generated by A
Kab:	key generated by S, the session key to be established between A and B

Figure 1. The ANSSK' Protocol

$\mathcal{F}_{\text{ke}}$. Upon corruption of $\mathcal{F}_{\text{ke}}$, the adversary can determine which key to give to a party.

Let $\mathcal{P}$ be a protocol system that describes one session of ANSSK', allows for static corruption of parties, and relies on $\mathcal{F}_{\text{enc}}$ for encryption and decryption. It is straightforward to specify $\mathcal{P}$ precisely. Since Kas, Kbs, and Kb are used for encryption in the protocol, encryption and decryption under these keys will be handled by $\mathcal{F}_{\text{enc}}$. If a key is corrupted in $\mathcal{F}_{\text{enc}}$, $\mathcal{P}$ is specified in such a way that the corresponding party (or parties in case of long-term keys) is considered to be corrupted, i.e., the adversary takes full control over this party. (Note that $\mathcal{P}$ can ask whether a key in $\mathcal{F}_{\text{enc}}$ is corrupted.) For example, if Kb is corrupted, then so is B, and if Kas is corrupted, then so are A and S. We assume, similarly to formulations of key exchange protocols by Canetti et al. (see, e.g., [17]), that the entities in $\mathcal{P}$, i.e., initiator, responder, and server, are invoked with the same pair (A, B), which tells each entity the names of the parties which want to exchange a session key in that session. This information could be exchanged at the beginning of a protocol run as part of an ID for that session. It is easy to see that $\mathcal{P}$ is a standard protocol (see Section IV-C), in particular $\mathcal{P}$ is non-committing and used order respecting.

We obtain the following theorem, which says that the ANSSK' protocol when using ideal (authenticated) encryption, realizes the ideal key exchange functionality $\mathcal{F}_{\text{ke}}$.

Theorem 6. $\mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}} \leq \mathcal{F}_{\text{ke}}$.

Before proving this theorem, we note that by the results of Section IV-C, we immediately obtain the following corollary, in which $\mathcal{F}_{\text{enc}}^{\text{auth}}$ is replaced by its realization $\mathcal{P}_{\text{enc}}^{\text{auth}}$.

Corollary 3. $\mathcal{P} \,|\, \mathcal{P}_{\text{enc}}^{\text{auth}} \leq \mathcal{F}_{\text{ke}}$.

This corollary says that the ANSSK' protocol when implemented with authenticated encryption realizes $\mathcal{F}_{\text{ke}}$.

The above theorem and corollary are only concerned with a single protocol session. Security w.r.t. multiple sessions follows immediately by Theorem 2. Together with Theorem 6 we obtain $!\underline{\mathcal{P}} \,|\, !\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}} \leq !\underline{\mathcal{F}_{\text{ke}}}$. In the realization $!\underline{\mathcal{P}} \,|\, !\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}}$ of $!\underline{\mathcal{F}_{\text{ke}}}$, $\mathcal{P}$ uses fresh long-term keys in every session. However, the joint state theorem (Theorem 4) allows us to replace $!\underline{\mathcal{F}_{\text{ltsenc}}^{\text{auth}}}$ in $!\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}}$ by its joint state realization, in which a single long-term key between two parties is used across all sessions. Altogether, this realization of the ANSSK' protocol is secure (as a key exchange protocol) in every polynomially bounded environment and no matter how many copies of this protocol run concurrently, even if a pair of parties uses the same long-term symmetric key across all sessions.

Proof sketch of Theorem 6: We first need to define a simulator: The simulator $\mathcal{S}$ simulates $\mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$ and sends a completion request to $\mathcal{F}_{\text{ke}}$ if $\mathcal{P}$ outputs a key. Upon corruptions of any party, the simulator can corrupt $\mathcal{F}_{\text{ke}}$ and then is free to complete with exactly the same key as in the real world. Hence, in case of corruption nothing is to show.

For the uncorrupted case, first note that, by definition of $\mathcal{F}_{\text{enc}}^{\text{auth}}$, the only plaintexts returned by $\mathcal{F}_{\text{enc}}^{\text{auth}}$ upon decryption are the ones "inserted" upon encryption. (We can ignore public-key encryption as it is not used in the protocol.) Now, in every run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$ (without corruption), we have: i) Since $\mathcal{P}$ handles only a single protocol session, say between A, B, and S, the instance of $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ for $\{B, S\}$ contains at most the plaintext (A, Kb), the instance of $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ for $\{A, S\}$ contains at most the plaintext (Na', B, Kab', c), and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ contains at most the plaintext (Kab, A) for the key Kb. (Here we use that, in the session we consider, the instances of A, B, and S expect a key exchange between A and B.) The latter two plaintexts were inserted by S, hence, by definition of S, it follows that $Kab = Kab'$. ii) Since we are in the uncorrupted case, Kb is initially marked unknown in $\mathcal{F}_{\text{senc}}^{\text{auth}}$, when created for B. Also, the instance of $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ for $\{B, S\}$, which is used to encrypt Kb, is uncorrupted. It follows that Kb is always marked unknown. Given this and the fact that the instance $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ for $\{A, S\}$ is uncorrupted, the session key Kab is only encrypted ideally, i.e., instead of the messages containing Kab only the leakage of these messages are encrypted. Hence, $\mathcal{E}$'s view on a run is information theoretically independent of Kab. But then, $\mathcal{E}$ cannot distinguish between the session key Kab output by $\mathcal{P}$ in runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$ and the session key generated and output by $\mathcal{F}_{\text{ke}}$ in runs of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{ke}}$. Therefore, it is easy to establish a one-to-one correspondence between the runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$ and those of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{ke}}$. Thus, we obtain: $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{ke}}$. ■

We emphasize that due to the use of $\mathcal{F}_{\text{enc}}$, we did not need to reason about IND-CPA and INT-CTXT games. We also remark that, in the above proof, we did not use the nonce Na in our argumentation (see Appendix A for more on this point).

B. Theorems on Secretive Protocols in the Game-based Approach

We now use $\mathcal{F}_{\text{enc}}$ for proving general theorems about cryptographic protocols in a game-based setting. More

specifically, Roy et al. [31], [30] define what they call secretive protocols for key exchange protocols that rely on symmetric encryption and that can be corrupted statically. Intuitively, a protocol is secretive w.r.t. some key k, typically the session key to be exchanged, if the key is only sent "properly" encrypted. Roy et al. show that if in a secretive protocol the key k is never used, then this guarantees key indistinguishability for k, in the sense of Bellare et al. [7], [5]. In case the key is used within the protocol (e.g., in a key confirmation phase), IND-CCA key usability is still guaranteed. IND-CCA key usability [22] means that k can be used securely as a key in an IND-CCA game, i.e., an adversary first interacting with the protocol and then with the IND-CCA game, in which k is used as a key, has only a 1/2 plus negligible chance of winning the game. Roy et al. also prove that if in a protocol run of a secretive protocol an honest party successfully decrypts a ciphertext with k, then, with overwhelming probability, this ciphertext originates from an encryption by an honest party.

In this section, we define secretive protocols and formulate the mentioned theorems by Roy et al. in our setting. Using Corollary 2, the proofs of these theorems are now very simple; they do not require to reason about IND-CCA, IND-CPA, or INT-CTXT games. These theorems can be applied to many protocols, including the one discussed above, the original ANSSK protocol, and Kerberos. While Roy et al. do not consider public-key encryption, our theorems immediately extend to protocols that use public-key encryption, as public-key encryption is supported by $\mathcal{F}_{\mathrm{enc}}/\mathcal{P}_{\mathrm{enc}}$.

Let $\mathcal{P}$ be a protocol system that relies solely on public key and symmetric encryption, uses $\mathcal{P}_{\mathrm{enc}}$ for this purpose, and specifies an unbounded number of sessions of a key exchange protocol.

Definition 2. A protocol system $\mathcal{P}$ as above is called *secretive* if $\mathcal{P}$ is non-committing, used order respecting, and for every environmental system $\mathcal{E}$, which may connect to the I/O and network interfaces of $\mathcal{P} \,|\, \mathcal{F}_{\mathrm{enc}}$, it holds with overwhelming probability that the session key in some uncorrupted session picked by $\mathcal{E}$ is always marked unknown in $\mathcal{F}_{\mathrm{enc}}$ or, for key indistinguishability (where the session key is not a short-term key in $\mathcal{F}_{\mathrm{enc}}$), has never been encrypted by a corrupted instance of $\mathcal{F}_{\mathrm{ltsenc}}$ or $\mathcal{F}_{\mathrm{pke}}$, or keys marked known in $\mathcal{F}_{\mathrm{enc}}$.

We note that Roy et al. did not assume the protocol to be non-committing and used order respecting, but they also needed to prohibit key cycles.

Key indistinguishability. Following Roy et al., for key indistinguishability, we assume that session keys are never used as keys in the protocol itself. Therefore, in the specification of $\mathcal{P}$ in our setting, these keys do not have to be handled by $\mathcal{P}_{\mathrm{enc}}$, they can rather be modeled as bit strings outside of $\mathcal{P}_{\mathrm{enc}}$, generated within $\mathcal{P}$ itself. We also assume that $\mathcal{P}$ is such that an environment interacting with $\mathcal{P}$ may pick a party in a complete and uncorrupted session, and then obtains the corresponding session key output by that party.

Key indistinguishability for $\mathcal{P}$ can now be formulated as follows in our setting: Consider an environmental system $\mathcal{E}$ for $\mathcal{P} \,|\, \mathcal{P}_{\mathrm{enc}}$ consisting of two subsystems. One subsystem, call it $\mathcal{A}$, interacts with $\mathcal{P} \,|\, \mathcal{P}_{\mathrm{enc}}$ (both on the I/O and network interfaces) and at some point picks a party in a complete and uncorrupted session. The other subsystem, which is the same for all $\mathcal{A}$, call it $\mathcal{T}$, receives the session key output by that party and then gives this key or a randomly generated key to $\mathcal{A}$. The task of $\mathcal{A}$ is to decide which key it was given. Then, $\mathcal{T}$ outputs 1 if $\mathcal{A}$ guessed correctly, and 0 otherwise. *Key indistinguishability* for $\mathcal{P}$ means that the probability that $\mathcal{T}$ outputs 1 is bounded above $1/2$ by a negligible function in the security parameter for every $\mathcal{A}$.

We obtain the following analog of the theorem by Roy et al. on key indistinguishability. Due to the use of $\mathcal{F}_{\mathrm{enc}}$, the proof of this theorem is considerably simpler than the one by Roy et al. In particular, we do not need to reason about IND-CCA, IND-CPA, or INT-CTXT games.

Theorem 7. *Let $\mathcal{P}$ be a secretive protocol as described above. Then, $\mathcal{P}$ satisfies key indistinguishability. This holds both in case $\mathcal{P}$ relies on $\mathcal{P}_{\mathrm{enc}}^{\mathrm{unauth}}$ and on $\mathcal{P}_{\mathrm{enc}}^{\mathrm{auth}}$ for encryption and decryption, i.e., for both IND-CCA secure and authenticated encryption schemes.*

Proof: Let $\mathcal{E}$ be an environmental system as described for key indistinguishability. Since $\mathcal{P}_{\mathrm{enc}}$ is a realization of $\mathcal{F}_{\mathrm{enc}}$, by Theorem 1, Corollary 2, and the transitivity of $\leq$ we know that there exists a simulator $\mathcal{S}$ such that:

$$\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{P}_{\mathrm{enc}} \equiv \mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{enc}} \ . \tag{1}$$

Since $\mathcal{P}$ is secretive and $\mathcal{E} \,|\, \mathcal{S}$ can be considered to be an environment for $\mathcal{P} \,|\, \mathcal{F}_{\mathrm{enc}}$, it follows that in runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{enc}}$, the session key, say k, output by the party picked by $\mathcal{A}$ has never been encrypted by a corrupted instance of $\mathcal{F}_{\mathrm{pke}}$ or $\mathcal{F}_{\mathrm{ltsenc}}$, or keys marked known in $\mathcal{F}_{\mathrm{senc}}$. Moreover, by assumption, we know that k is not used to encrypt other messages. But then, from the definition of $\mathcal{F}_{\mathrm{enc}}$ it follows that k has always been encrypted ideally. Hence, the view of $\mathcal{A}$ in runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{enc}}$ is independent of the value of k. Consequently, the probability that $\mathcal{T}$ outputs 1 is exactly $1/2$. By (1), the probability that $\mathcal{T}$ outputs 1 in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{P}_{\mathrm{enc}}$ is bounded above $1/2$ by a negligible function, as desired. ■

Key usability. Let $\mathcal{P}$ be a protocol as above that uses $\mathcal{P}_{\mathrm{enc}}$ for encryption and decryption, i.e., $\mathcal{P}$ uses an IND-CCA secure or authenticated encryption scheme. For key usability, unlike key indistinguishability, session keys may be used as keys in the protocol. Therefore, in the specification of $\mathcal{P}$, session keys will be treated as short-term keys within $\mathcal{P}_{\mathrm{enc}}$.

As mentioned above, key usability for session keys established in uncorrupted sessions means that these keys can be used securely as keys in IND-CCA games, i.e., an adversary first interacting with the protocol and then with the IND-CCA game, in which a session key of an uncorrupted session picked by the adversary is used as a key, has only a 1/2 plus negligible chance of winning the game.

Following Roy et al., we want to show that secretive protocols $\mathcal{P}$ imply key usability. Using $\mathcal{F}_{\mathrm{enc}}$ this is very simple: We first consider the experiment for key usability described above in the case where $\mathcal{P}_{\mathrm{enc}}$ is replaced by $\mathcal{F}_{\mathrm{enc}}$, i.e., $\mathcal{F}_{\mathrm{enc}}$ is used both by $\mathcal{P}$ and the IND-CCA game. Now, since $\mathcal{P}$ is secretive, it follows by definition of secretive protocols that when running $\mathcal{P}$ with $\mathcal{F}_{\mathrm{enc}}$ session keys in uncorrupted sessions are marked unknown in $\mathcal{F}_{\mathrm{enc}}$. Consequently, the session key used in the IND-CCA game is marked unknown in $\mathcal{F}_{\mathrm{enc}}$. Hence, queries of the form (m_0, m_1) to the left-right-oracle of the IND-CCA game are answered by encrypting m_0 or m_1 ideally in $\mathcal{F}_{\mathrm{enc}}$, i.e., only the leakage of m_0/m_1 is encrypted. Since m_0 and m_1 have the same length, the leakage of m_0 and m_1 is the same. But then it follows immediately that the adversary can only win the IND-CCA game with probability $1/2$.

Now, since $\mathcal{P}_{\mathrm{enc}}$ realizes $\mathcal{F}_{\mathrm{enc}}$, we can replace $\mathcal{F}_{\mathrm{enc}}$ by $\mathcal{P}_{\mathrm{enc}}$, and it follows that the adversary can only win the IND-CCA game with probability $1/2$ plus some negligible function. Hence, we obtain the following analog of the theorem by Roy et al. on key usability (see Appendix B for more details).

Theorem 8. *Let $\mathcal{P}$ be a secretive protocol where the session keys are never used to encrypt other short-term keys. Then, $\mathcal{P}$ satisfies key usability. This holds both in case $\mathcal{P}$ relies on $\mathcal{P}_{\mathrm{enc}}^{\mathrm{unauth}}$ and in case $\mathcal{P}$ relies on $\mathcal{P}_{\mathrm{enc}}^{\mathrm{auth}}$ for encryption and decryption, i.e., for both IND-CCA secure and authenticated encryption schemes.*

In the above theorem, we assume that a protocol does not use a session key to encrypt other short-term keys. This assumption is quite natural (all protocols that we have encountered satisfy this property). However, the assumption did not seem to be necessary in the work by Roy et al. Our assumption guarantees that the environment remains non-committing after the session key has been "given" to the IND-CCA game. A more relaxed assumption would guarantee this as well. Nevertheless, for simplicity we state the theorem for this simpler assumption.

Ciphertext integrity. Similarly to the above theorems, we obtain the following theorem for ciphertext integrity.

We allow session keys to be used within the protocol. Therefore, in the specification of $\mathcal{P}$, session keys will be treated as short-term keys within $\mathcal{P}_{\mathrm{enc}}$.

Ciphertext integrity under session keys for $\mathcal{P}$ can now be formulated as follows in our setting: Consider an environmental system $\mathcal{E}$ running with $\mathcal{P} \mid \mathcal{P}_{\mathrm{enc}}$ which picks a party in a complete and uncorrupted session. If a ciphertext is successfully decrypted with the session key output by that party but the ciphertext did not originate from an encryption by an honest party, then $\mathcal{E}$ outputs 1. Note that this event cannot be observed by $\mathcal{E}$ alone. However, within $\mathcal{P}$ this event can be observed. It is easy to extend any $\mathcal{P}$ to say $\mathcal{P}_{\mathrm{int}}$ in which this event is observed and reported to $\mathcal{E}$. Now, $\mathcal{P}$ satisfies *ciphertext integrity under session keys* for every environmental system $\mathcal{E}$ as just described if the probability that in a run of $\mathcal{E} \mid \mathcal{P}_{\mathrm{int}} \mid \mathcal{P}_{\mathrm{enc}}$ the bit 1 is output is negligible in the security parameter.

Theorem 9. *Let $\mathcal{P}$ be a secretive protocol. Then, $\mathcal{P}$ satisfies ciphertext integrity under session keys. This holds in case $\mathcal{P}$ relies on $\mathcal{P}_{\mathrm{enc}}^{\mathrm{auth}}$ for encryption and decryption, i.e., authenticated encryption schemes.*

Proof: Let $\mathcal{E}$ be an environmental system as described for ciphertext integrity. First it is easy to see that if $\mathcal{P}$ is secretive, then so is $\mathcal{P}_{\mathrm{int}}$.

Now, since $\mathcal{P}_{\mathrm{enc}}$ is a realization of $\mathcal{F}_{\mathrm{enc}}$, by Theorem 1, Corollary 2, and the transitivity of $\leq$ we know that there exists a simulator $\mathcal{S}$ such that:

$$\mathcal{E} \mid \mathcal{P}_{\mathrm{int}} \mid \mathcal{P}_{\mathrm{enc}} \equiv \mathcal{E} \mid \mathcal{P}_{\mathrm{int}} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}} \quad . \tag{2}$$

Since $\mathcal{P}_{\mathrm{int}}$ is secretive and $\mathcal{E} \mid \mathcal{S}$ can be considered to be an environment for $\mathcal{P}_{\mathrm{int}} \mid \mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$, it follows that in runs of $\mathcal{E} \mid \mathcal{P}_{\mathrm{int}} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$, the session key picked by $\mathcal{E}$ is always marked unknown in $\mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$. But then, from the definition of $\mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$ it follows that if a ciphertext successfully decrypts, it must originate from an encryption of an honest party. Hence, $\mathcal{E}$ never outputs 1 in a run with $\mathcal{P}_{\mathrm{int}} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{enc}}^{\mathrm{auth}}$. By (2), the probability that $\mathcal{E}$ outputs 1 in a run with $\mathcal{P}_{\mathrm{int}} \mid \mathcal{P}_{\mathrm{enc}}^{\mathrm{auth}}$ is negligible, as desired. ■

VI. Related Work

Backes and Pfitzmann proposed a functionality for symmetric encryption within their cryptographic library [3]. This library differs from our functionality in several aspects. The main motivation for this library was to relate Dolev-Yao style protocol analysis with cryptographic protocol analysis. For this purpose, the user is provided with an abstract Dolev-Yao style interface, which, except for payload data, never provides the user with real bit strings, but only with pointers to Dolev-Yao terms in the library. This allows for quite abstract, Dolev-Yao style reasoning; protocol analysis has to be carried out w.r.t. multiple sessions, though, as Backes and Pfitzmann do not have a joint state theorem. The Dolev-Yao style abstraction, however, comes with a price: *All* operations on messages, not only encryption and decryption, but also concatenation of messages, nonce generation, sending and receiving of messages, etc. have to be performed via the

library. Also, the realization of the library requires non-standard assumptions about the encryption scheme; authenticated encryption schemes do not suffice, extra randomness as well as identifiers for symmetric keys have to be added. Moreover, the realization of the library assumes specific message formats: message tags are used for all types of messages (nonces, ciphertexts, payloads, pairs of messages). Altogether, based on the library only those protocols can be analyzed which merely use operations provided by the library and these protocols can only be shown to be secure w.r.t. the non-standard encryption schemes and assuming the specific message formats. Conversely, our symmetric key functionalities provide less abstraction than the library by Backes and Pfitzmann: Arbitrary messages (bit strings) can be encrypted, where only pointers to keys are interpreted, and real ciphertexts are returned to the user. But not only are the interfaces of our functionalities less abstract, also the treatment of plaintexts and ciphertexts *inside* the functionalities is: Rather than performing abstract manipulations on Dolev-Yao terms, actual encryption and decryption algorithms are applied to bit strings. By being lower-level functionalities the range of applications of our functionalities is broader and the analysis performed based on these functionalities makes only standard cryptographic assumptions, IND-CCA security and authenticated encryption, with almost no restriction on message formats or cryptographic primitives used alongside symmetric encryption.

Other works concerned with abstractions of symmetric encryption include [1], [27], [19]. However, these works do not consider simulation-based security and, just as the work by Backes and Pfitzmann, aim at computational soundness of Dolev-Yao style reasoning. In the full version [20] of the work by Comon-Lundh and Cortier [19], several examples are presented pointing out a problem that forced the authors to make the rather unrealistic assumption that the adversary cannot fabricate keys, except for honestly running the key generation algorithm. In other words, dishonestly generated keys are disallowed. The authors pointed out that they do not know how the problem that they encountered is solved in the cryptographic library by Backes and Pfitzmann. Indeed the examples by Comon-Lundh and Cortier suggest that dishonestly generated keys also have to be forbidden for the cryptographic library, in case symmetric encryption is considered. We note that, in our setting, dishonestly generated keys do not cause problems, since, as mentioned above, our functionalities provide a lower level of abstraction of symmetric encryption.

In [22], a formal logic that enjoys a computational, game-based semantics is used to reason about protocols that use symmetric encryption. In [21], Datta et al. prove that certain variants of symmetric encryption cannot have realizable ideal functionalities.

We finally mention the tool CryptoVerif [9] by Blanchet for analyzing protocols that employ symmetric encryption in a game-based cryptographic setting.

ACKNOWLEDGMENT

We would like to thank Ran Canetti for helpful discussions on early versions of our functionalities. We also thank the anonymous reviewers for their remarks, which helped to improve the presentation of this work.

This work was partially supported by the *Deutsche Forschungsgemeinschaft* (DFG) under Grant KU 1434/5-1 and 1434/4-2, and the *Schweizerischer Nationalfonds* (SNF) under Grant 200021-116596.

REFERENCES

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, International Conference (IFIPTCS 2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2000.

[2] M. Backes and D. Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. In Kan Zhang and Yuliang Zheng, editors, *Information Security, 7th International Conference, ISC 2004, Proceedings*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004.

[3] 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.

[4] 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.

[5] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols (Extended Abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing (STOC 1998)*, pages 419–428. ACM Press, 1998.

[6] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In Tatsuaki 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.

[7] 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 1995)*, pages 57–66. ACM, 1995.

[8] J. Black, Ph. Rogaway, and Th. Shrimpton. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In Kaisa Nyberg and Howard 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.

[9] B. Blanchet. A Computationally Sound Mechanized Prover for Security Protocols. In *IEEE Symposium on Security and Privacy (S&P 2006)*, pages 140–154. IEEE Computer Society, 2006.

[10] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[11] 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.

[12] 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.

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

[14] R. Canetti. Security and Composition of Cryptographic Protocols: A Tutorial. Cryptology ePrint Archive, Report 2006/465, 2006. Online available at http://eprint.iacr.org/.

[15] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally Composable Security with Global Setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[16] R. Canetti and M. Fischlin. Universally Composable Commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.

[17] R. Canetti and J. Herzog. Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[18] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In Lars R. Knudsen, editor, *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] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, pages 109–118. ACM Press, 2008.

[20] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. Technical Report INRIA Research Report RR-6508, INRIA, 2008. Online available at http://www.loria.fr/~cortier/Papiers/CCS08-report.pdf.

[21] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan, and A. Scedrov. Games and the Impossibility of Realizable Ideal Functionality. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2006.

[22] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi. Computationally Sound Compositional Logic for Key Exchange Protocols. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 321–334. IEEE Computer Society, 2006.

[23] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[24] 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.

[25] 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.

[26] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. Technical Report 2009/055, Cryptology ePrint Archive, 2009. Online available at http://eprint.iacr.org/.

[27] P. Laud. Symmetric Encryption in Automatic Analyses for Confidentiality against Active Adversaries. In *IEEE Symposium on Security and Privacy 2004 (S&P 2004)*, pages 71–85. IEEE Computer Society, 2004.

[28] R. M. Needham and M. D. Schroeder. Authentication revisited. *SIGOPS Operating Systems Review*, 21(1):7–7, 1987.

[29] 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 2001 (S&P 2001)*, pages 184–201. IEEE Computer Society Press, 2001.

[30] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Proofs of Computational Secrecy. In Joachim Biskup and Javier 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.

[31] 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 2007)*, 2007.

APPENDIX A.
REMARKS ON THE ANALYSIS OF THE ANSSK' PROTOCOL

In our argumentation in the proof of Theorem 6 we did not use Na. In fact, this nonce is not needed. The reason is that the analysis only involved a single session of the protocol (see below). Nevertheless, by applying Theorem 2 we obtain from Theorem 6 that $!\underline{\mathcal{P}} \mid !\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}} \leq !\underline{\mathcal{F}_{\text{ke}}}$, i.e., the multi-session version of $\mathcal{P}$ realizes ideal key exchanges in multiple sessions. In this realization, $\mathcal{P}$ uses fresh long-term keys in every session. However, we can apply the joint state theorem (Theorem 4) for $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ to replace $!\underline{\mathcal{F}_{\text{ltsenc}}^{\text{auth}}}$ in $!\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}}$ by a joint state realization. We obtain that

$$!\underline{\mathcal{P}} \mid !\underline{\mathcal{F}_{\text{senc}}^{\text{auth}}} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{JS}} \mid !\mathcal{F}_{\text{ltsenc}}^{\text{auth}} \leq !\underline{\mathcal{F}_{\text{ke}}} \ .$$

By the composition theorems (Theorem 1 and 2), Theorem 3, Corollary 2, and transitivity of $\leq$ we finally obtain

$$!\underline{\mathcal{P}} \mid !\underline{\mathcal{P}_{\text{senc}}^{\text{auth}}} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{JS}} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{auth}} \leq !\underline{\mathcal{F}_{\text{ke}}} \ .$$

As explained in Section III, in this realization SIDs are embedded into plaintexts before encryption with long-term keys. This is why $\mathcal{P}$ realizes $\mathcal{F}_{\text{ke}}$ also for multiple concurrent sessions, even if the nonce Na is dropped.

We note that one could show Theorem 6 directly for the case that $\mathcal{P}$ and $\mathcal{F}_{\text{ke}}$ handle multiple sessions. In this case, one does not need to resort to the joint state realization. In the analysis one would need to make use of Na. Altogether the analysis would become more involved. However, using $\mathcal{F}_{\text{enc}}$ it would still be fairly simple, in particular since the arguments would be rather straightforward information theoretically, without the need for considering IND-CCA and INT-CTXT games, as this has been done once and for all in proofs of the realizations of $\mathcal{F}_{\text{senc}}$ and $\mathcal{F}_{\text{ltsenc}}$ (Theorems 3 and 5).

APPENDIX B.
KEY USABILITY FOR SECRETIVE PROTOCOLS

Let $\mathcal{P}$ be a protocol as in Section V-B that uses $\mathcal{P}_{\text{enc}}$ for encryption and decryption. To define key usability more formally, we consider an extension $\mathcal{P}'$ of $\mathcal{P}$. In $\mathcal{P}'$, an environment interacting with $\mathcal{P}$ may pick a party, say p, in a complete and uncorrupted session. As a result, $\mathcal{P}'$ will provide the environment with the pointer ptr to the session key of party p in that session. Moreover, $\mathcal{P}'$ allows the environment to encrypt and decrypt messages under the key corresponding to the pointer ptr. However, the protocol $\mathcal{P}$ stops. Hence, from now on, the environment can only encrypt and decrypt messages under ptr using $\mathcal{P}_{\text{enc}}$.

Key usability for $\mathcal{P}$ can now be formulated as follows in our setting: Consider an environmental system $\mathcal{E}$ for $\mathcal{P}' \mid \mathcal{P}_{\text{enc}}$ consisting of two subsystems. One subsystem, call it $\mathcal{A}$, interacts with $\mathcal{P}' \mid \mathcal{P}_{\text{enc}}$ (both on the I/O and network interfaces) and at some point picks a party p in a complete and uncorrupted session. The pointer ptr to the corresponding session key is given to the second subsystem of $\mathcal{E}$, call it $\mathcal{T}$. From now on, $\mathcal{A}$ cannot interact with the protocol anymore but only with $\mathcal{T}$. Moreover, $\mathcal{T}$ can only use $\mathcal{P}'$ as an interface to encrypt and decrypt messages under the key corresponding to ptr in $\mathcal{P}_{\text{enc}}$. The subsystem $\mathcal{T}$, which is the same for all $\mathcal{A}$, behaves like a left-or-right oracle for encryption and decryption under the key corresponding to ptr: $\mathcal{T}$ first randomly chooses a bit b. Upon an encryption request from $\mathcal{A}$ of the form (m_0, m_1), where m_0, m_1 are arbitrary bit strings of the same length, $\mathcal{T}$ uses $\mathcal{P}_{\text{enc}}$ to encrypt m_b with the key corresponding to ptr, where the encryption of m_0 and m_1 is done in such a way that the bit strings m_0 and m_1 are encrypted exactly as they are, without interpreted substrings of the form (Key, x), if any. (This is possible using the *store* and *reveal* commands of $\mathcal{F}_{\text{senc}}$, see [26] for details). Upon a decryption request from $\mathcal{A}$ of the form c, where c is a bit string which has not been returned by $\mathcal{T}$ before, $\mathcal{T}$ uses $\mathcal{P}_{\text{enc}}$ to decrypt c with the key corresponding to ptr, the resulting plaintext (if any) is returned to $\mathcal{A}$ (again uninterpreted). The task of $\mathcal{A}$ is to guess b. When $\mathcal{A}$ sends its guess b' to $\mathcal{T}$, $\mathcal{T}$ outputs 1 if $b' = b$, and 0 otherwise.

Now, *key usability* for $\mathcal{P}$ means that the probability that $\mathcal{T}$ outputs 1 is bounded above $1/2$ by a negligible function in the security parameter for every $\mathcal{A}$. This exactly captures the notion of key usability in [22].

Proof Sketch of Theorem 8: Let $\mathcal{E}$ be an environmental system as described for key usability. First, one can show that $\mathcal{E} \mid \mathcal{P}'$ is non-committing and used order respecting [26], using that $\mathcal{P}$ has these properties.

Since $\mathcal{P}_{\text{enc}}$ is a realization of $\mathcal{F}_{\text{enc}}$, by Theorem 1, Corollary 2, and the transitivity of $\leq$ we know that there exists a simulator $\mathcal{S}$ such that:

$$\mathcal{E} \mid \mathcal{P}' \mid \mathcal{P}_{\text{enc}} \equiv \mathcal{E} \mid \mathcal{P}' \mid \mathcal{S} \mid \mathcal{F}_{\text{enc}} \ . \tag{3}$$

Since $\mathcal{P}$ is secretive and $\mathcal{E} \mid \mathcal{S}$ can be considered to be an environment for $\mathcal{P}' \mid \mathcal{F}_{\text{enc}}$, it follows by the definition of $\mathcal{P}'$ that in the first phase in runs of $\mathcal{E} \mid \mathcal{P}' \mid \mathcal{S} \mid \mathcal{F}_{\text{enc}}$, where the subsystem $\mathcal{A}$ of $\mathcal{E}$ interact with $\mathcal{P}$, the session key picked by $\mathcal{A}$ is always marked unknown in $\mathcal{F}_{\text{enc}}$. This does not change in the second phase, where $\mathcal{A}$ interacts with the subsystem $\mathcal{T}$ of $\mathcal{E}$. Hence, from the definition of $\mathcal{F}_{\text{enc}}$ it follows that encrypt requests of the form (m_0, m_1) from $\mathcal{A}$ are answered by $\mathcal{T}$ using $\mathcal{F}_{\text{enc}}$ as encryption of the leakage of m_b. Since the leakages of m_0 and m_1 have the same distribution, as m_0 and m_1 have the same length, no information about bit b is revealed to $\mathcal{A}$. Consequently, the probability that $\mathcal{T}$ outputs 1 is exactly $1/2$. By (3), the probability that $\mathcal{T}$ outputs 1 in a run of $\mathcal{E} \mid \mathcal{P}' \mid \mathcal{P}_{\text{enc}}$ is bounded above $1/2$ by a negligible function, as desired. ■