Universally Composable Symmetric Encryption

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

February, 2009

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. We also illustrate the usefulness of our functionalities in applications, both in simulation-based and game-based security settings.

Contents

1 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., [14, 16, 15, 21, 3, 32, 35]). There are also many functionalities for more sophisticated cryptographic tasks (see, e.g., [17] for an overview). Surprisingly, however, a functionality for symmetric encryption, similar to the one for public key encryption, as first proposed by Canetti [14], is still missing; there only exists an abstract Dolev-Yao style functionality (see below). 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. This makes 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, a user must not get his/her hands on these keys directly, but should only be able to refer to these keys by pointers. 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 in order to be able to encrypt other keys, which, encapsulated in the ciphertext, could then travel (securely). 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 [28, 1] and the commitment problem [4, 19, 27]. 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, $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$. We show that $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ can be realized by an encryption scheme iff the encryption scheme is IND-CCA secure. Similarly, we show equivalence between $\mathcal{F}_{\text{senc}}^{\text{auth}}$ and an encryption scheme for authenticated encryption, i.e., an IND-CPA and INT-CTXT secure encryption scheme [9]. 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 [4], 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 $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ currently contain two mechanisms for bootstrapping symmetric encryption: (authenticated or 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, $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$. 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., [16, 35, 20, 32]). 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 [32]. These joint state theorems guarantee that in different protocol sessions the same long-term and public/private keys may be used, while 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 [34] 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 by Roy et al. [36, 37] 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 [29]. While being in the spirit of Canetti's UC model [16], it has certain advantages over the UC model, as demonstrated and discussed in [29, 32]. 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 [32]. Putting these problems aside, the results presented here would, however, also carry over to the UC model.

Related Work. Backes and Pfitzmann proposed a functionality for symmetric encryption within their cryptographic library [4]. 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. By being lower-level functionalities the range of applications 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, 33, 23]. 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 [24] of the work by Comon-Lundh and Cortier [23], 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.

In [26], a formal logic that enjoys a computational, game-based semantics is used to reason about protocols that use symmetric encryption. In [27], Datta et al. prove that certain variants of symmetric

encryption cannot have realizable ideal functionalities.

We finally mention the tool CryptoVerif [12] by Blanchet for analyzing protocols that employ symmetric encryption in a game-based cryptographic setting.

Structure of this paper. In the next section, we recall the computational model for simulation-based security that we use. Notation on defining functionalities and other preliminaries are given in Section 3. The functionalities for bootstrapping symmetric encryption are presented in Section 4. The functionality, $\mathcal{F}_{\text{senc}}^{\text{auth}}$, for symmetric encryption is introduced in Section 5, along with their implementation. In Section 6 the functionality $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and their implementation is described. Applications are presented in Section 7.

2 The IITM Model

In this section, we briefly recall the IITM model for simulation-based security (see [29] for details). Based on a relatively simple, but very expressive general computational model in which IITMs and systems of IITMs are defined, simulation-based security notions are formalized, and a general composition theorem can be proved in the IITM model.

2.1 The General Computational Model

We first define IITMs and then systems of IITMs. We note that this general computation model is also useful in contexts other than simulation-based security [25].

Syntax of IITMs. An *(inexhaustible) interactive Turing machine (IITM, for short, or simply ITM) M* is a probabilistic Turing machine with input and output tapes. These tapes have names and, in addition, input tapes have an attribute with values consuming or enriching (see below for an explanation). We require that different tapes of M have different names. The names of input and output tapes 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 with a (enriching) input tape named start, is called a *master IITM*. It will be triggered if no other IITM was triggered. An IITM is triggered by another IITM if the latter sends a message to the former. Each IITM comes with an associated polynomial q which is used to bound the computation time per activation and the length of the overall output produced by M.

Computation of IITMs. 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 can not be exhausted (therefore the name *inexhaustible* interactive Turing machine). The length of the configuration and the length of the overall output an IITM can produce is polynomially bounded in the security parameter and the length of the overall input received on *enriching* input tapes so far, i.e., writing messages on these tapes increases the resources (runtime, potential size of the configuration, and potential length of the output) of the IITM. 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 is a very generic addressing mechanism: Details of how an IITM is addressed are not fixed up-front, but left to the specification of the IITM itself.

Systems of IITMs. A *system* $\mathcal{S}$ of IITMs is defined according to the following grammar:

$$\mathcal{S} ::= M \mid (\mathcal{S} \parallel \mathcal{S}) \mid !\mathcal{S}.$$

where M ranges of the set of IITMs. No two input tapes occurring in IITMs in $\mathcal{S}$ are allowed to have the same names, i.e. every input tape name belongs to exactly one IITM in the system. The system $\mathcal{S}_1 \parallel \mathcal{S}_2$ is the *concurrent composition* of the two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ and $!\mathcal{S}$ is the *concurrent*

composition of an unbounded number of copies of (machines in) the system $\mathcal{S}$. Each machine M that occurs in a subexpression $!\mathcal{S}'$ of $\mathcal{S}$ is said to be in the *scope of a bang*. Below, we define the way a system runs. From that it will be clear that every system $\mathcal{S}$ is equivalent to a system of the shape $M_1 \parallel \cdots \parallel M_k \parallel !M'_1 \parallel \cdots \parallel !M'_{k'}$ where M_i for all $i \in \{1, \ldots, k\}$ and M'_j for all $j \in \{1, \ldots, k'\}$ are IITMs.

In a run of $\mathcal{S}$ at every time only one IITM, say a copy of some M in $\mathcal{S}$, is active and all other IITMs wait for new input; the first IITM to be activated in a run of $\mathcal{S}$ will be the master IITM, which may get some auxiliary input written on tape start. The 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. There may be several copies of an IITM M' in $\mathcal{S}$ with input tape named c. In the order in which these copies were generated, these copies are run in mode $\mathsf{CheckAddress}$. The first of these copies to accept m will process m in mode $\mathsf{Compute}$. If no copy accepts m, it is checked whether a newly generated copy of M' (if M' is in the scope of a bang) with fresh random coins, would accept m. If yes, this copy gets to process m. Otherwise, the master IITM in $\mathcal{S}$ is activated (by writing ε on tape start). The master IITM is also activated if the currently active IITM did not produce output. A run stops if the master IITM does not produce output (and hence, does not trigger another machine) or an IITM outputs a message on a tape named $\mathsf{decision}$. Such a message is considered to be the overall output of the system.

Formally, as defined in [29], a *configuration* C of $\mathcal{S}$ given the security parameter η and external input a (a *configuration* of $\mathcal{S}(1^\eta, a)$, for short) is a tuple of the form (A, P) where A is the sequence of configurations of IITMs (the sequence of previously activated machines) and P is a system (the passive machines). A *(complete) run* ρ of $\mathcal{S}(1^\eta, a)$ is a sequence of configurations $C_0, \ldots, C_k$ of $\mathcal{S}(1^\eta, a)$ such that C_0 is an initial configuration, C_{i+1} is a successor configuration of C_i, for all $i < k$, and C_k does not have a successor configuration. By $\Pr[\rho]$ we denote the probability of the run ρ, see [30] for details. By $\mathrm{runs}^a_\eta(\mathcal{S})$ we denote the set of all runs of $\mathcal{S}(1^\eta, a)$.

We will only consider so-called well-formed systems [29], which satisfy a simple syntactic condition that guarantees polynomial runtime of systems and suffices for applications since it allows to always provide sufficient resources to IITMs via enriching tapes.

A system is *well-formed* if the master IITM (if there is any) does not occur in the scope of a bang and there are no cycles in the connection of the IITMs via their enriching tapes. For example, the system $\mathcal{S} = M_1 \parallel M_2$ is not well-formed if M_1 is a master IITM with an output tape c_2 which is an enriching input tape of M_2 and M_2 has an output tape c_1 which is an enriching input tape of M_1. In fact, M_1 and M_2 could sent messages back and forth between each other forever as they are connected via enriching tapes.

Theorem 1 ([29])**.** *(informal) Well-formed systems run in polynomial time.*

We say that a run is *accepting* if the overall output (the message written on tape $\mathsf{decision}$, if any) is 1. We write $\Pr[\mathcal{S}(1^\eta, a) \rightsquigarrow 1]$ to denote the probability that a run of a (well-formed) system $\mathcal{S}$ with security parameter η and auxiliary input a for the master IITM is accepting. The set of accepting runs is denoted by $\mathrm{aruns}^a_\eta(\mathcal{S})$. By $\Pr[\rho]$ we denote the probability of a run ρ. For sets of runs $\mathcal{C} \subseteq \mathrm{runs}^a_\eta(\mathcal{S})$ we have that $\Pr[\mathcal{C}] = \sum_{\rho \in \mathcal{C}} \Pr[\rho]$ and we define the *complement* $\overline{\mathcal{C}} = \mathrm{runs}^a_\eta(\mathcal{S}) \setminus \mathcal{C}$. For example, $\Pr[\overline{\mathcal{C}}] = 1 - \Pr[\mathcal{C}]$ and $\Pr[\mathrm{aruns}^a_\eta(\mathcal{S})] = \Pr[\mathcal{S}(1^\eta, a) \rightsquigarrow 1]$.

Let $f \colon \{1\}^* \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$. Two well-formed systems $\mathcal{P}$ and $\mathcal{Q}$ are called *f-equivalent* or *f-indistinguishable* ($\mathcal{P} \equiv_f \mathcal{Q}$) iff $|\Pr[\mathcal{P}(1^\eta, a) \rightsquigarrow 1] - \Pr[\mathcal{Q}(1^\eta, a) \rightsquigarrow 1]| \leq f(1^\eta, a)$ for all $\eta \in \mathbb{N}$, $a \in \{0,1\}^*$. Furthermore, $\mathcal{P}$ and $\mathcal{Q}$ are called *equivalent* or *indistinguishable* ($\mathcal{P} \equiv \mathcal{Q}$) iff there exists a negligible function f (i.e., for all polynomials p and q there exists $\eta_0 \in \mathbb{N}$ such that for all $\eta > \eta_0$ and all bit strings $a \in \{0,1\}^*$ with length $|a| \leq q(\eta)$ we have that $f(1^\eta, a) \leq \frac{1}{p(\eta)}$) such that $\mathcal{P} \equiv_f \mathcal{Q}$. Note that $\equiv_0$ denotes perfect indistinguishability.

Given an IITM M, we will often use its *identifier (ID) version* $\underline{M}$ to be able to address multiple copies of M (see [29, 30] for a detailed definition). The identifier version $\underline{M}$ of M is an IITM which simulates M within a "wrapper". 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 $\mathsf{CheckAddress}$ 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. If M specifies the actions

performed by a party in a multi-party protocol, then $!\underline{M}$ specifies the multi-party protocol where every copy of $\underline{M}$ has a different PID. Note that one can consider an ID version $\underline{\underline{M}}$ of $\underline{M}$, which effectively means that the ID is a tuple of two IDs. Of course, this can be iterated further. Given a system $\mathcal{S}$, its *identifier (ID) version* $\underline{\mathcal{S}}$ is obtained by replacing all IITMs by their ID version. For example, with $\mathcal{S} = M_1 \parallel \cdots \parallel M_k \parallel !M'_1 \parallel \cdots \parallel !M'_{k'}$ as above, we obtain $\underline{\mathcal{S}} = \underline{M_1} \parallel \ldots \parallel \underline{M_k} \parallel !\underline{M'_1} \parallel \ldots \parallel !\underline{M'_{k'}}$. Note that for all i, all copies of $\underline{M'_i}$ in a run of $\underline{\mathcal{S}}$ will have different IDs.

2.2 Notions of Simulation-Based Security

In order to define security notions for simulation-based security, we need further notation.

Let us consider a system $\mathcal{S}$ and an IITM M. By $\mathcal{T}(M)$ ($\mathcal{T}(\mathcal{S})$) we denote the set of (names of) tapes of the machine M (of the machines in $\mathcal{S}$). The set $\mathcal{T}(M)$ is partitioned into the set of input and output tapes $\mathcal{T}_{in}(M)$ and $\mathcal{T}_{out}(M)$, respectively. A tape c in $\mathcal{T}(\mathcal{S})$ is called *internal* if there exist machines M and M' in $\mathcal{S}$ such that c is an input tape of M and an output tape of M', i.e. $c \in \mathcal{T}_{in}(M) \cap \mathcal{T}_{out}(M')$. Otherwise, c is called *external*. The set of external tapes of $\mathcal{S}$ is denoted by $\mathcal{T}_{ext}(\mathcal{S})$ and is partitioned into the set of *(external) input* and *(external) output tapes* of $\mathcal{S}$, $\mathcal{T}_{in}(\mathcal{S})$ and $\mathcal{T}_{out}(\mathcal{S})$, respectively. An external tape c is an *input tape* of $\mathcal{S}$, if there exists an IITM M in $\mathcal{S}$ with an input tape c. On the other hand, an external tape c is an *output tape* of $\mathcal{S}$ if there exists an IITM M in $\mathcal{S}$ with an output tape c. The set of external tapes is further partitioned into the set of *network* and *I/O tapes*. This partitions each of the sets $\mathcal{T}_{ext}(\mathcal{S})$, $\mathcal{T}_{in}(\mathcal{S})$ and $\mathcal{T}_{out}(\mathcal{S})$ into $\mathcal{T}^{net}_{ext}(\mathcal{S})$ and $\mathcal{T}^{io}_{ext}(\mathcal{S})$, $\mathcal{T}^{net}_{in}(\mathcal{S})$ and $\mathcal{T}^{io}_{in}(\mathcal{S})$ and $\mathcal{T}^{net}_{out}(\mathcal{S})$ and $\mathcal{T}^{io}_{out}(\mathcal{S})$, respectively.

With the composition $\mathcal{P} \mid \mathcal{Q}$ of two systems $\mathcal{P}$ and $\mathcal{Q}$, we describe the concurrent composition $\mathcal{P}' \parallel \mathcal{Q}'$ where $\mathcal{P}'$ and $\mathcal{Q}'$ are obtained from $\mathcal{P}$ and $\mathcal{Q}$ by renaming all internal tapes such that the internal tapes of $\mathcal{P}'$ are disjoint from the tapes of $\mathcal{Q}'$ and vice versa. Informally speaking, $\mathcal{P}$ and $\mathcal{Q}$ communicate only via their external tapes.

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are *compatible*, if they have the same external tapes with the same attributes, i.e. $\mathcal{T}^{net}_{in}(\mathcal{P}) = \mathcal{T}^{net}_{in}(\mathcal{Q})$, $\mathcal{T}^{net}_{out}(\mathcal{P}) = \mathcal{T}^{net}_{out}(\mathcal{Q})$, $\mathcal{T}^{io}_{in}(\mathcal{P}) = \mathcal{T}^{io}_{in}(\mathcal{Q})$, $\mathcal{T}^{io}_{out}(\mathcal{P}) = \mathcal{T}^{io}_{out}(\mathcal{Q})$, and each external tape c is enriching in $\mathcal{P}$ iff it is enriching in $\mathcal{Q}$.

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are *I/O compatible* if they do not interfere on network tapes, i.e. $\mathcal{T}^{net}_{ext}(\mathcal{P}) \cap \mathcal{T}^{net}_{ext}(\mathcal{Q}) = \emptyset$, and have the same set of I/O tapes, i.e. $\mathcal{T}^{io}_{in}(\mathcal{P}) = \mathcal{T}^{io}_{in}(\mathcal{Q})$, $\mathcal{T}^{io}_{out}(\mathcal{P}) = \mathcal{T}^{io}_{out}(\mathcal{Q})$ and the attributes are the same.

A system $\mathcal{P}$ is *connectible* for a system $\mathcal{Q}$ if each common external tape has the same type in both systems (network or I/O) and complementary directions (input or output), i.e. for each common external tape $c \in \mathcal{T}_{ext}(\mathcal{P}) \cap \mathcal{T}_{ext}(\mathcal{Q})$, it holds that c is a network tape in $\mathcal{P}$ iff it is one in $\mathcal{Q}$ and c is an input tape in $\mathcal{P}$ iff it is an output tape in $\mathcal{Q}$. For a set $\mathbf{B}$ of systems, $\mathrm{Con}_{\mathbf{B}}(\mathcal{Q})$ denotes the set of systems in $\mathbf{B}$ which are connectible for $\mathcal{Q}$.

A system $\mathcal{A}$ is *adversarially connectible* for a system $\mathcal{P}$ if it is connectible for $\mathcal{P}$ and $\mathcal{A}$ does not communicate with $\mathcal{P}$ via I/O tapes, i.e. $\mathcal{T}_{ext}(\mathcal{A}) \cap \mathcal{T}^{io}_{ext}(\mathcal{P}) = \emptyset$. For a set $\mathbf{B}$ of systems, $\mathrm{Sim}^{\mathcal{P}}_{\mathbf{B}}(\mathcal{F})$ denotes the set of systems $\mathcal{S}$ in $\mathbf{B}$ which are adversarially connectible for the system $\mathcal{F}$ and $\mathcal{S} \mid \mathcal{F}$ is compatible with $\mathcal{P}$.

A system $\mathcal{E}$ is *environmentally connectible* for a system $\mathcal{P}$ if it is connectible for $\mathcal{P}$ and does not communicate with $\mathcal{P}$ via network tapes, i.e. $\mathcal{T}_{ext}(\mathcal{E}) \cap \mathcal{T}^{net}_{ext}(\mathcal{P}) = \emptyset$. For a set $\mathbf{B}$ of systems, $\mathrm{Env}_{\mathbf{B}}(\mathcal{P})$ denotes the set of systems in $\mathbf{B}$ which are environmentally connectible for $\mathcal{P}$.

We define three different types of well-formed systems (whose composition will again be well-formed): A system $\mathcal{P}$ is called a *protocol system* if it is well-formed, $\mathcal{P}$ has no tape named start or decision, all network tapes are consuming (I/O tapes may be enriching) and if an IITM M of $\mathcal{P}$ occurs not in the scope of a bang, then M accepts every message in mode CheckAddress. The set of all protocol systems is denoted by $\mathbf{P}$. Requiring network tapes to be consuming is not a real restriction in applications since sufficient resources can always be provided by an environment via the I/O tapes, e.g., to forward messages between the network and I/O interface. A system $\mathcal{A}$ is called an *adversarial system* if it is well-formed and $\mathcal{A}$ has no tape named start or decision. (All external tapes of $\mathcal{A}$ may be enriching.) The set of all adversarial systems is denoted by $\mathbf{A}$ or $\mathbf{S}$. A system $\mathcal{E}$ is called an *environmental system* if it is well-formed, tape start may be enriching and all other external tapes are consuming. The set of all environmental systems is denoted by $\mathbf{E}$.

We are now ready to define the security notion that we will use.

Definition 1 (Strong Simulatability (SS); [29])**.** Let $\mathcal{P}$ and $\mathcal{F}$ be I/O compatible protocol systems, the real and the ideal protocol, respectively. Then, $\mathcal{P}$ *SS-realizes* $\mathcal{F}$ ($\mathcal{P} \leq^{SS} \mathcal{F}$) iff there exists an adversarial system $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}}_{\mathbf{S}}(\mathcal{F})$ such that for all environmental systems $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P})$ it holds that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$.

In a similar way, other equivalent security notions such as black-box simulatability and (dummy) UC can be defined [29]. We emphasize that in these and the above definitions, no specific addressing or corruption mechanism is fixed. This can be defined in a rigorous, convenient, and flexible way as part of the real/ideal protocol specifications.

We note that the strong simulatability relation is transitive, i.e. if $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$ are pairwise I/O compatible protocol systems and $\mathcal{Q}_1 \leq^{SS} \mathcal{Q}_2$ and $\mathcal{Q}_2 \leq^{SS} \mathcal{Q}_3$, then $\mathcal{Q}_1 \leq^{SS} \mathcal{Q}_3$. The strong simulatability relation is also reflexive, i.e., for all protocol systems $\mathcal{P}$, we have that $\mathcal{P} \leq^{SS} \mathcal{P}$ [1].

2.3 Composition Theorems

We restate the composition theorems from [29]. The first composition theorem describes concurrent composition of a fixed number of protocol systems while the second one the composition of an unbounded number of copies of a protocol system.

Theorem 2 ([29])**.** *Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be protocol systems such that the systems $\mathcal{P}_1 \,|\, \ldots \,|\, \mathcal{P}_k$ and $\mathcal{F}_1 \,|\, \ldots \,|\, \mathcal{F}_k$ are well-formed and for every $j \in \{1, \ldots, k\}$ the following conditions are satisfied:*

1. *$\mathcal{P}_j$ is environmentally connectible for $\mathcal{P}_{j+1} \,|\, \ldots \,|\, \mathcal{P}_k$,*
2. *$\mathcal{F}_j$ is environmentally connectible for $\mathcal{F}_{j+1} \,|\, \ldots \,|\, \mathcal{F}_k$,*
3. *$\mathcal{P}_j$ and $\mathcal{F}_j$ are I/O compatible and*
4. *$\mathcal{P}_j \leq^{SS} \mathcal{F}_j$.*

Then,

$$\mathcal{P}_1 \,|\, \ldots \,|\, \mathcal{P}_k \leq^{SS} \mathcal{F}_1 \,|\, \ldots \,|\, \mathcal{F}_k \ .$$

Theorem 3 ([29])**.** *Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems such that $\mathcal{P}$ and $\mathcal{F}$ are I/O compatible and $\mathcal{P} \leq^{SS} \mathcal{F}$. Then,*

$$!\underline{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}} \ .$$

As an immediate consequence of the above theorems, we obtain:

Corollary 1. *If $\mathcal{P}_1, \mathcal{P}_2, \mathcal{F}_1$ and $\mathcal{F}_2$ are protocol systems such that $\mathcal{P}_1 \,|\, !\underline{\mathcal{P}_2}$ and $\mathcal{F}_1 \,|\, !\underline{\mathcal{F}_2}$ are well-formed, $\mathcal{P}_1$ and $\mathcal{F}_1$ are environmentally connectible for $\mathcal{P}_2$ and $\mathcal{F}_2$ (resp.), $\mathcal{P}_1$ and $\mathcal{F}_1$ are I/O compatible, $\mathcal{P}_2$ and $\mathcal{F}_2$ are I/O compatible, $\mathcal{P}_1 \leq^{SS} \mathcal{F}_1$ and $\mathcal{P}_1 \leq^{SS} \mathcal{F}_1$, then*

$$\mathcal{P}_1 \,|\, !\underline{\mathcal{P}_2} \leq^{SS} \mathcal{F}_1 \,|\, !\underline{\mathcal{F}_2} \ .$$

Iterated application of Theorem 2 and 3 allows to construct very complex systems, e.g., protocols using several levels of an unbounded number of copies of sub-protocols. Unlike the UC model, super-protocols can directly access sub-protocols across levels, yielding simpler and possibly more efficient implementations. In the UC model, a protocol has to completely shield its sub-protocol from the environment, and hence, from super-protocols on higher levels. In [18], the composition operator therefore had to be extended to allow access to a globally available functionality. No such extension would have been necessary in the IITM model to obtain the results proved in this work. We also note that Theorem 3 cannot only be interpreted as yielding multi session realizations from single session realizations, but also providing multi party realizations from single party realizations (when $\underline{\mathcal{P}}$ and $\underline{\mathcal{F}}$ are considered as multi party versions).

[1] Technically, we have that $\mathcal{P} \leq^{SS} \mathcal{P}'$ where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by renaming the network tapes. This is required because by I/O compatibility the network tapes have to be disjoint.

2.4 Proof Method for Equivalence of Systems

In this section we establish a method for proving equivalence of systems.

Let $\mathcal{S}_0$ and $\mathcal{S}_1$ be well-formed systems, $\eta \in \mathbb{N}$, and $a \in \{0,1\}^*$. Let $R \subseteq \text{runs}^a_\eta(\mathcal{S}_0) \times \text{runs}^a_\eta(\mathcal{S}_1)$. By $\text{dom}(R) = \{\rho_0 \mid \exists \rho_1 \colon (\rho_0, \rho_1) \in R\}$ we denote the *domain* and by $\text{rng}(R) = \{\rho_1 \mid \exists \rho_0 \colon (\rho_0, \rho_1) \in R\}$ we denote the *range* of R. By $\sim_R$ we denote the transitive-reflexive closure of $R \cup R^{-1}$, i.e. an equivalence relation on $\text{dom}(R) \cup \text{rng}(R)$,[2] and by $[\sim_R]$ the equivalence classes of $\sim_R$. The relation R is called a (c_0, c_1, c_2)-*probabilistic trace (or run) relation* between $\text{runs}^a_\eta(\mathcal{S}_0)$ and $\text{runs}^a_\eta(\mathcal{S}_1)$ (where $c_0, c_1, c_2 \in \mathbb{R}_{\geq 0}$) if for all $C \in [\sim_R]$

1. all runs in C are either all accepting or all not accepting,
2. $\Pr[\overline{\text{dom}(R)}] \leq c_2$, and
3. $|\Pr[C_0] - \Pr[C_1]| \leq c_0 \cdot \Pr[C_0] + c_1 \cdot \Pr[C_1]$ where $C_0 = C \cap \text{runs}^a_\eta(\mathcal{S}_0)$ and $C_1 = C \cap \text{runs}^a_\eta(\mathcal{S}_1)$.

Theorem 4. *Let $\mathcal{S}_0$ and $\mathcal{S}_1$ be well-formed systems, $\eta \in \mathbb{N}$, $a \in \{0,1\}^*$, and R a (c_0, c_1, c_2)-probabilistic trace relation between $\text{runs}^a_\eta(\mathcal{S}_0)$ and $\text{runs}^a_\eta(\mathcal{S}_1)$. Then,*

$$\left|\Pr[\mathcal{S}_0(1^\eta, a) \rightsquigarrow 1] - \Pr[\mathcal{S}_1(1^\eta, a) \rightsquigarrow 1]\right| \leq c_0 + c_1 + c_2 \ .$$

Proof. We have that

$$\begin{aligned}
\Pr[\text{aruns}^a_\eta(\mathcal{S}_0)] &\leq \Pr[\text{aruns}^a_\eta(\mathcal{S}_0) \cap \text{dom}(R)] + \Pr[\overline{\text{dom}(R)}] \\
&= \sum_{C \in [\sim_R]} \Pr[\text{aruns}^a_\eta(\mathcal{S}_0) \cap C] + \Pr[\overline{\text{dom}(R)}] \\
&\leq \sum_{C \in [\sim_R]} \Pr[\text{aruns}^a_\eta(\mathcal{S}_1) \cap C] + c_0 + c_1 + \Pr[\overline{\text{dom}(R)}] \\
&\leq \Pr[\text{aruns}^a_\eta(\mathcal{S}_1)] + c_0 + c_1 + c_2 \ .
\end{aligned}$$

Furthermore, we have that

$$\begin{aligned}
\Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_1)}] &\geq \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_1)} \cap \text{rng}(R)] = \sum_{C \in [\sim_R]} \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_1)} \cap C] \\
&\geq \sum_{C \in [\sim_R]} \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_0)} \cap C] - c_0 - c_1 \geq \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_0)}] - \Pr[\overline{\text{dom}(R)}] - c_0 - c_1 \\
&\geq \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_0)}] - c_0 - c_1 - c_2
\end{aligned}$$

from which we conclude that

$$\begin{aligned}
\Pr[\text{aruns}^a_\eta(\mathcal{S}_1)] &= 1 - \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_1)}] \\
&\leq 1 - \Pr[\overline{\text{aruns}^a_\eta(\mathcal{S}_0)}] + c_0 + c_1 + c_2 = \Pr[\text{aruns}^a_\eta(\mathcal{S}_0)] + c_0 + c_1 + c_2
\end{aligned}$$

which completes the proof. □

By the above theorem, to prove that $\mathcal{S}_0$ and $\mathcal{S}_1$ are equivalent ($\mathcal{S}_0 \equiv \mathcal{S}_1$) it suffices to show that there exists a negligible function f and a family of relations $(R_{\eta,a})_{\eta \in \mathbb{N}, a \in \{0,1\}^*}$ such that $R_{\eta,a}$ is an $(f(1^\eta, a), f(1^\eta, a), f(1^\eta, a))$-probabilistic trace relation for all for all $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$.

A useful special case of the above theorem is the following: Let $B \subseteq \text{runs}^a_\eta(\mathcal{S}_0)$ be an *error set* of runs of $\mathcal{S}_0$ (i.e., runs we do not want to consider) and R an injective function from $\text{runs}^a_\eta(\mathcal{S}_0) \setminus B$ to $\text{runs}^a_\eta(\mathcal{S}_1)$ where ρ_0 is accepting iff $R(\rho_0)$ is accepting and $\Pr[\rho_0] = \Pr[R(\rho_0)]$ for all $\rho_0 \in \text{runs}^a_\eta(\mathcal{S}_0) \setminus B$. Then $\Pr[B] = \Pr[\overline{\text{dom}(R)}]$ and thus R is a $(0, 0, \Pr[B])$-probabilistic trace relation. Hence, Theorem 4 implies

$$|\Pr[\mathcal{S}_0(1^\eta, a) \rightsquigarrow 1] - \Pr[\mathcal{S}_1(1^\eta, a) \rightsquigarrow 1]| \leq \Pr[B] \ .$$

[2] By R^{-1} we denote the inverse relation $R^{-1} = \{(\rho_1, \rho_0) \mid (\rho_0, \rho_1) \in R\}$.

3 Preliminaries

3.1 Notation

For describing IITMs and algorithms we use standard pseudo code with an obvious semantics. By $x := y$ we denote deterministic assignment of the variable or constant y to variable x. By $x \leftarrow A$ we denote probabilistic assignment to the variable x according to the distribution of algorithm A. By $x \leftarrow^{\mathrm{R}} S$ we denote that x is chosen uniformly at random from the set S. By x++ we abbreviate $x := x + 1$ and by $x := y$++ we abbreviate the sequence $x := y,\ y := y + 1$.

The description of an IITM will be divided into three parts: *State*, *CheckAddress* and *Compute*. The first part is used to describe the variables that describe the state of the IITM and also the initial state while the others describe the behavior of the IITM in mode CheckAddress and Compute, respectively. The description in mode Compute, consists of a sequence of blocks where every block is of the form $\langle condition \rangle : \langle actions \rangle$. Upon activation, the conditions of the blocks are checked one after the other. If a condition is satisfied the corresponding actions are carried out.

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

In the description of actions we often write "send m on t". This means that the IITM outputs message m on tape t and stops for this activation. In the next activation the IITM will not proceed at the point where it stopped, but again go through the list of conditions, starting with the first one, as explained above. However, if we write "send m on t and wait for receiving m' on t'" (or simply "send m on t and receive m' on t'"), then the IITM does the following: It outputs m on tape t and stops for this activation. In the next activation, it will check whether it received a message on input tape t' and check whether this message matches with m'. If it does, the computation continues. Otherwise, the IITM stops for this activation without producing output. In the next activation, it will again check whether it received a message on input tape t' and whether this message matches with m' and behaves as before, and so on, until it receives the expected message on t'.

Typically, an IITM M is parameterized by a set of tape names $\mathcal{T}_{\text{users}}$ and a tape name T_{adv} which induces the I/O and network interface as follows: M has the I/O input tape T^{in} and the I/O output tape T^{out} for each $T \in \mathcal{T}_{\text{users}}$. Furthermore, M has the network input tape $T^{\text{in}}_{\text{adv}}$ and the network output tape $T^{\text{out}}_{\text{adv}}$. Of course M might have additional tapes. In the description of M, we abbreviate "send m on T^{out} (resp., $T^{\text{out}}_{\text{adv}}$)" by "send m to T (resp., T_{adv})" and "receive m on T^{in} (resp., $T^{\text{in}}_{\text{adv}}$)" by "receive m from T (resp., T_{adv})".

Running External Code. Sometimes, an IITM M obtains the description of an algorithm A as input on some tape and has to execute it. We write $y \leftarrow \text{sim}_n(A, x)$ to say that the IITM simulates algorithm A on input x for n steps. The random coins that might be used by A are chosen by M. The variable y is set to the output of A if A terminates after at most n steps. Otherwise, y is set to the error symbol $\perp$. If we want to enforce M to simulate A in a deterministic way we write $y := \text{sim-det}_n(A, x)$. If A uses random coins, M can simply use the zero bit string.

The executing IITM is only allowed to perform a polynomial number of steps for executing the algorithm A, i.e., n has to be bounded polynomially in the security parameter plus the length of the input. Note that at least the degree of the polynomial that bounds n has to be fixed in advance because it must not depend on the security parameter. This holds true for any definition of polynomial time and is not a limitation of the definition of polynomial time in the IITM model.

3.2 Leakage Algorithms

The functionalities for encryption that we consider in this paper will all produce ciphertexts that do not depend on the actual plaintext that was encrypted but only leak some information about the plaintext, e.g., the length. A probabilistic leakage algorithm, as defined in this section, specifies the amount of information that is leaked. For ideal encryption, instead of directly encrypting a plaintext m the leakage algorithm is applied to m to obtain a plaintext $\overline{m}$, the leakage, which is then encrypted.

Definition 2. A *domain* D over $\{0,1\}^*$ is a family $(D_\eta)_{\eta \in \mathbb{N}}$ where $D_\eta \subseteq \{0,1\}^*$. We require that there exists a polynomial time algorithm T such that $T(1^\eta, x) = 1$ if and only if $x \in D_\eta$.

A *(probabilistic) leakage algorithm* L with domain D is a probabilistic polynomial time (PPT) algorithms L which takes as input 1^η (where η is the security parameter) and a bit string $x \in D_\eta$ and returns a bit string. By $\mathrm{dom}(L)$ we denote the domain D of L.

In this paper, we only consider leakage algorithms L where the domain satisfies the following property: If $x \in \mathrm{dom}(L)$ then $x' \in \mathrm{dom}(L)$ for all $x' \in \{0,1\}^{|x|}$.

Often, it is required or one wants to guarantee that encryption does not leak more than the length of the plaintext. This can be modeled by an appropriate leakage algorithm which leaks the length of a plaintext, e.g. one of the following leakage algorithms.

Example 1. The leakage algorithms L^0 and $L^{|\cdot|}$ both leak the length of a message.

1. $L^0(1^\eta, x)$ returns $0^{|x|}$.
2. $L^{|\cdot|}(1^\eta, x)$ chooses $r \leftarrow^{\mathrm{R}} \{0,1\}^{|x|}$ uniformly at random and returns r.

Most often we consider length preserving leakage algorithms or even leakage algorithms that leak exactly the length of a message.

Definition 3. A leakage algorithm L is *length preserving* if $\Pr[|L(1^\eta, x)| = |x|] = 1$ for all $\eta \in \mathbb{N}$ and $x \in \mathrm{dom}(L)_\eta$.

A leakage algorithm L *leaks exactly the length of a message* if it is length preserving and the probability distributions of $L(1^\eta, x)$ and $L(1^\eta, y)$ are equal for all $\eta \in \mathbb{N}$ and $x, y \in \mathrm{dom}(L)_\eta$ with $|x| = |y|$.

For example, the leakage algorithms in Example 1 both leak exactly the length of a message. Hence, the amount of information leaked is the same, namely the length of the message, but the second has another property which we call *high entropy*.

Definition 4. A leakage algorithm L has *high entropy* if for all $x, y \in \mathrm{dom}(L)$ the probability of collisions $\Pr[L(1^\eta, x) = L(1^\eta, y)]$ is negligible (as a function in η).

Leakage algorithms with high entropy have advantages when the ideal functionality is used for reasoning about security properties of protocols because collisions occur only with negligible probability. Leakage algorithms with high entropy have been studied in [32] for public key encryption functionalities and we can also use it for the long-term and short-term symmetric key encryption functionalities presented in this paper, see Section 4.1 and 5.1.

For example, L^0 does not have high entropy. If the domain of $L^{|\cdot|}$ contains only *long* messages, e.g. if $|x| \geq \eta$ for all $\eta \in \mathbb{N}$ and $x \in \mathrm{dom}(L)_\eta$ then $L^{|\cdot|}$ has high entropy. On the other hand, $L^{|\cdot|}$ does not have high entropy for domains with short messages because there are collisions with non-negligible probability.

3.3 IND-CCA, IND-CPA and INT-CTXT Security

Traditionally, security notions for symmetric encryption schemes, see, e.g., [8, 9] and references therein, are defined with respect to uniform adversaries, i.e., adversaries that do not obtain auxiliary input except for the security parameter. Simulation-based frameworks like [15, 29] deal with non-uniform environments and hence non-uniform adversaries. In order to reduce security to the underlying primitives the notions have to be compatible. We have chosen to define IND-CCA, IND-CPA and INT-CTXT security with respect to non-uniform adversaries. However, all our results carry over to the uniform setting, i.e., where all environments are uniform and we use the standard (i.e. uniform) definitions of IND-CCA, IND-CPA and INT-CTXT security.

For completeness, we recall the notions of [8, 9] and adapt them to non-uniform adversaries.

Definition 5. A *symmetric encryption scheme* $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ with domain $\mathrm{dom}(\Sigma)$ (see above) consists of a probabilistic polynomial-time (PPT) *key generation algorithm* $\mathrm{gen}(x\colon \{1\}^*)\colon \{0,1\}^*$, a PPT *encryption algorithm* $\mathrm{enc}(k\colon \{0,1\}^*, m\colon \{0,1\}^*)\colon \{0,1\}^*$, a deterministic polynomial-time *decryption algorithm* $\mathrm{dec}(k\colon \{0,1\}^*, m\colon \{0,1\}^*)\colon \{0,1\}^* \cup \{\perp\}$, and a *domain of plaintexts* $\mathrm{dom}(\Sigma)$. Furthermore, it is required that $\mathrm{dec}(k, \mathrm{enc}(k, m)) = m$ for all $\eta \in \mathbb{N}$, keys k generated by $\mathrm{gen}(1^\eta)$ and $m \in \mathrm{dom}(\Sigma)_\eta$.

We assume that every encryption scheme is associated with a polynomial q that bounds the runtime of the algorithms and the length of there description in some standard encoding. We say that Σ is *q-bounded*. For all encryption schemes Σ considered in this paper, we assume that (for a fixed security parameter) the generated keys have constant length, i.e., for all $\eta \in \mathbb{N}$ exists $\text{len-key}_\eta \in \mathbb{N}$ such that all keys generated by $\text{gen}(1^\eta)$ have length len-key_η. This is without loss of generality because one could add appropriate padding.

Note that dec outputs $\perp$ if the decryption fails, i.e., the presented ciphertext is not a valid ciphertext with respect to the used key.

We define $\text{LR}(m_0, m_1, b) = m_b$ for $b \in \{0,1\}$.

An *IND-CPA adversary* $A_{\text{cpa}}^{O(\cdot,\cdot)}(x : \{1\}^*, a : \{0,1\}^*) : \{0,1\}$ is a probabilistic algorithm that has access to the oracle O such that all the two messages queried of O always have equal length.

Definition 6 (IND-CPA secure)**.** Let Σ be a symmetric encryption scheme, $b \in \{0,1\}$, $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$. Let A_{cpa} be an IND-CPA adversary. The corresponding experiment is defined in Algorithm 1. The *advantage* of the adversary is defined as follows:

$$\text{Adv}^{\text{ind-cpa}}_{\Sigma,A_{\text{cpa}}}(1^\eta, a) = \left| \Pr\left[\text{Exp}^{\text{ind-cpa-1}}_{\Sigma,A_{\text{cpa}}}(1^\eta, a) = 1\right] - \Pr\left[\text{Exp}^{\text{ind-cpa-0}}_{\Sigma,A_{\text{cpa}}}(1^\eta, a) = 1\right] \right| .$$

A symmetric encryption scheme Σ is called *IND-CPA secure* if for all polynomial-time IND-CPA adversaries A_{cpa} the advantage $\text{Adv}^{\text{ind-cpa}}_{\Sigma,A_{\text{cpa}}}(1^\eta, a)$ is negligible as a function in η and a.

An *IND-CCA adversary* $A_{\text{cca}}^{O_1(\cdot,\cdot),O_2(\cdot)}(x : \{1\}^*, a : \{0,1\}^*) : \{0,1\}$ is a probabilistic algorithm that has access to the oracles O_1 and O_2 such that all the two messages queried of O_1 always have equal length and O_2 is never queried with a message returned by O_1.

Definition 7 (IND-CCA secure)**.** Let Σ be a symmetric encryption scheme, $b \in \{0,1\}$, $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$. Let A_{cca} be an IND-CCA adversary. The corresponding experiment is defined in Algorithm 2. The *advantage* of the adversary is defined as follows:

$$\text{Adv}^{\text{ind-cca}}_{\Sigma,A_{\text{cca}}}(1^\eta, a) = \left|\Pr\left[\text{Exp}^{\text{ind-cca-1}}_{\Sigma,A_{\text{cca}}}(1^\eta, a) = 1\right] - \Pr\left[\text{Exp}^{\text{ind-cca-0}}_{\Sigma,A_{\text{cca}}}(1^\eta, a) = 1\right]\right| .$$

A symmetric encryption scheme Σ is called *IND-CCA secure* if for all polynomial-time IND-CCA adversaries A_{cca} the advantage $\text{Adv}^{\text{ind-cca}}_{\Sigma,A_{\text{cca}}}(1^\eta, a)$ is negligible as a function in η and a.

An *INT-CTXT adversary* $A_{\text{int-ctxt}}^{O_1(\cdot),O_2(\cdot)}(x : \{1\}^*, a : \{0,1\}^*) : \{0,1\}$ is a probabilistic algorithm that has access to the oracles O_1 and O_2.

Definition 8 (INT-CTXT secure)**.** Let Σ be a symmetric encryption scheme, $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$. Let $A_{\text{int-ctxt}}$ be an INT-CTXT adversary. The corresponding experiment is defined in Algorithm 3. The *advantage* of an adversary is defined as follows:

$$\text{Adv}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta, a) = \Pr\left[\text{Exp}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta, a) = 1\right] .$$

A symmetric encryption scheme Σ is called *INT-CTXT secure* if for all polynomial-time INT-CTXT adversaries $A_{\text{int-ctxt}}$ the advantage $\text{Adv}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta, a)$ is negligible as a function in η, a.

To link the game based definition and ideal encryption in the simulation based setting more easy, we define the IITM Oracle which is parameterized by a symmetric encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$, a variable *mode* $\in \{\mathsf{unauth}, \mathsf{auth}, \mathsf{real}\}$ which specifies the behavior (unauthenticated ideal, authenticated ideal, or real behavior), and a leakage algorithm L. The IITM has an enriching I/O input tape $T^{\text{in}}_{\text{oracle}}$ and an I/O output tape $T^{\text{out}}_{\text{oracle}}$. First, Oracle expects a key generation message upon which it generates a key $k \leftarrow \text{gen}(1^\eta)$. After the key generation, the environment can use Oracle for encryptions and decryptions. In mode real these requests are answered by the actual encryption/decryption results. But in mode unauth and auth not the plaintext m is encrypted but instead the leakage $L(1^\eta, m)$. Also, for later decryption, m is recorded to be the decryption of the (ideal) ciphertext. Upon decryption of a ciphertext c Oracle returns i) $\perp$ if the decryption is ambiguous (i.e., if there are different recorded plaintext for c), ii) m if there is exactly one recorded plaintext m for c, and iii) $\text{dec}(k, c)$ in mode unauth and $\perp$ in mode auth if there is no recorded plaintext for c. See Figure 1 for a detailed definition.

Algorithm 1 $\mathrm{Exp}^{\text{ind-cpa-}b}_{\Sigma,A_{\text{cpa}}}(1^\eta, a)$

$k \leftarrow \mathrm{gen}(1^\eta)$
$b' \leftarrow A_{\text{cpa}}^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,b))}(1^\eta, a)$
return b'

Algorithm 2 $\mathrm{Exp}^{\text{ind-cca-}b}_{\Sigma,A_{\text{cca}}}(1^\eta, a)$

$k \leftarrow \mathrm{gen}(1^\eta)$
$b' \leftarrow A_{\text{cpa}}^{\mathrm{enc}(k,\mathrm{LR}(\cdot,\cdot,b)),\mathrm{dec}(k,\cdot)}(1^\eta, a)$
return b'

Algorithm 3 $\mathrm{Exp}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta, a)$

$k \leftarrow \mathrm{gen}(1^\eta)$
if $A_{\text{cpa}}^{\mathrm{enc}(k,\cdot),\mathrm{dec}(k,\cdot)}(1^\eta, a)$ makes a query c to $\mathrm{dec}(k,\cdot)$ such that $\mathrm{dec}(k,c) \neq \perp$ and c was never returned by the encryption oracle $\mathrm{enc}(k,\cdot)$ **then**
    return 1
else
    return 0

$\mathrm{Oracle}(\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec}), \mathit{mode} \in \{\mathsf{unauth}, \mathsf{auth}, \mathsf{real}\}, L)$

Tapes: enriching I/O input tape $T^{\text{in}}_{\text{oracle}}$, I/O output tape $T^{\text{out}}_{\text{oracle}}$
State: $\mathit{state} \in \{\mathsf{init}, \mathsf{ok}\}$ (initially init), $k \in \{0,1\}^* \cup \{\perp\}$ (initially $\perp$)
$\mathit{decTable} \subseteq \{0,1\}^* \times \{0,1\}^*$ (initially $\emptyset$)

1. Upon receiving (KeyGen) from T_{oracle} where $\mathit{state} = \mathsf{init}$:
   $k \leftarrow \mathrm{gen}(1^\eta)$, $\mathit{state} := \mathsf{ok}$, **send** (Ack) **to** T_{oracle}
2. Upon receiving (Enc, m) from T_{oracle} where $\mathit{state} = \mathsf{ok}$ and $m \in \mathrm{dom}(L)_\eta$:
   if $\mathit{mode} = \mathsf{real}$ **then**
      $c \leftarrow \mathrm{enc}(k, m)$
   else $\{\mathit{mode} \in \{\mathsf{unauth}, \mathsf{auth}\}\}$
      $\overline{m} \leftarrow L(1^\eta, m)$, $c \leftarrow \mathrm{enc}(k, \overline{m})$, $\mathit{decTable} := \mathit{decTable} \cup \{(m, c)\}$
   send $(\mathsf{Ciphertext}, c)$ **to** T_{oracle}
3. Upon receiving (Dec, c) from T_{oracle} where $\mathit{state} = \mathsf{ok}$:
   if $\mathit{mode} = \mathsf{real}$ **then**
      $m := \mathrm{dec}(k, c)$
   else $\{\mathit{mode} \in \{\mathsf{unauth}, \mathsf{auth}\}\}$

$$m := \begin{cases} m' & \text{if } \exists! m' \colon (m', c) \in \mathit{decTable} \quad (\text{exists unique } m') \\ \mathrm{dec}(k, c) & \text{if } \mathit{mode} = \mathsf{unauth} \text{ and } \forall m' \colon (m', c) \notin \mathit{decTable} \\ \perp & \text{otherwise} \end{cases}$$

   send $(\mathsf{Plaintext}, m)$ **to** T_{oracle}

Figure 1: The IITM Oracle is parameterized by a symmetric encryption scheme Σ, $\mathit{mode} \in \{\mathsf{unauth}, \mathsf{auth}, \mathsf{real}\}$ and a leakage L. Parameter *mode* specifies whether the behavior is unauthenticated ideal, authenticated ideal, or real.

Lemma 1. *Let L be a length preserving leakage algorithm and Σ a symmetric encryption scheme with domain* $\mathrm{dom}(L)$*. Then,*

1. $\mathrm{Oracle}(\Sigma, \mathsf{real}, L) \leq^{SS} \mathrm{Oracle}(\Sigma, \mathsf{unauth}, L)$ *if Σ is IND-CCA secure, and*
2. $\mathrm{Oracle}(\Sigma, \mathsf{real}, L) \leq^{SS} \mathrm{Oracle}(\Sigma, \mathsf{auth}, L)$ *if Σ is IND-CPA and INT-CTXT secure.*

Note: For systems $\mathcal{P}, \mathcal{Q}$ without network tapes, like Oracle, *$\leq^{SS}$ is symmetric, i.e., $\mathcal{P} \leq^{SS} \mathcal{Q}$ if and only if $\mathcal{Q} \leq^{SS} \mathcal{P}$.*

Proof. Throughout the proof, we abbreviate $\mathrm{Oracle}(\Sigma, mode, L)$ by $\mathrm{O}(mode)$. Since Oracle has no network tapes, we need no simulator. Let $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathrm{Oracle})$.

ad 1. We define an IND-CCA adversary A_{cca} and show that its advantage equals the advantage of $\mathcal{E}$ in distinguishing between $\mathrm{O}(\mathsf{real})$ and $\mathrm{O}(\mathsf{unauth})$.

The adversary $A_{\mathrm{cca}}^{O_1(\cdot,\cdot),O_2(\cdot)}(1^\eta, a)$ sets the variable $state := \bot$ and then simulates a run of $\mathcal{E}(1^\eta, a)$ as follows:

1. If $\mathcal{E}$ outputs (KeyGen) on $T^{\mathrm{in}}_{\mathrm{oracle}}$ then A_{cca} sets $state := \mathsf{ok}$ and sends (Ack) to $T^{\mathrm{out}}_{\mathrm{oracle}}$.
2. If $\mathcal{E}$ outputs (Enc, m) on $T^{\mathrm{in}}_{\mathrm{oracle}}$ then A_{cca} sends ε to start if $state \neq \mathsf{ok}$ or $m \notin \mathrm{dom}(L)_\eta$, and, otherwise, computes $c \leftarrow O_1(m, L(1^\eta, m))$, stores (m, c) and sends $(\mathsf{Ciphertext}, c)$ to $T^{\mathrm{out}}_{\mathrm{oracle}}$.
3. If $\mathcal{E}$ outputs (Dec, c) on $T^{\mathrm{in}}_{\mathrm{oracle}}$ then A_{cca} sends ε to start if $state \neq \mathsf{ok}$, and, otherwise, A_{cca} sends $(\mathsf{Plaintext}, m)$ to $T^{\mathrm{out}}_{\mathrm{oracle}}$ where
$$m := \begin{cases} m' & \text{if } \exists! m' \colon (m', c) \text{ stored} \\ O_2(c) & \text{if } \forall m' \colon (m', c) \text{ not stored} \\ \bot & \text{otherwise.} \end{cases}$$
4. If $\mathcal{E}$ terminates with overall output 1 (i.e., output 1 on tape $\mathsf{decision}$) then A_{cca} returns 1. Otherwise, if $\mathcal{E}$ terminates then A_{cca} returns 0.

First note that A_{cca} is a polynomial-time IND-CCA adversary as it never requests O_2 with ciphertexts returned by O_1 and the two messages given to O_1 are always of the same length because L is length preserving. As $\mathcal{E}$ is polynomial-time, so is A_{cca}.

One easily verifies that
$$\Pr\left[\mathrm{Exp}^{\text{ind-cca-1}}_{\Sigma, A_{\mathrm{cca}}}(1^\eta, a) = 1\right] = \Pr[(\mathcal{E} \mid \mathrm{O}(\mathsf{unauth}))(1^\eta, a) \rightsquigarrow 1] \ .$$
Furthermore, one obtains
$$\Pr\left[\mathrm{Exp}^{\text{ind-cca-0}}_{\Sigma, A_{\mathrm{cca}}}(1^\eta, a) = 1\right] = \Pr[(\mathcal{E} \mid \mathrm{O}(\mathsf{real}))(1^\eta, a) \rightsquigarrow 1]$$
because i) $\mathrm{dec}(k, \mathrm{enc}(k, m)) = m$ which implies that if (m, c) is stored then $O_2(m) = c$, and ii) for all stored pairs $(m, c), (m', c')$ we have $m = m'$ if $c = c'$ (this follows from i)). By i) and ii) we conclude that decryption is simulated perfectly.

Since Σ is IND-CCA secure, A_{cca} has only negligible advantage which yields that
$$\left|\Pr[(\mathcal{E} \mid \mathrm{O}(\mathsf{unauth}))(1^\eta, a) \rightsquigarrow 1] - \Pr[(\mathcal{E} \mid \mathrm{O}(\mathsf{real}))(1^\eta, a) \rightsquigarrow 1]\right|$$
is negligible as a function in η and a.

ad 2. We define an IND-CPA adversary A_{cpa} and an INT-CTXT adversary $A_{\text{int-ctxt}}$ and show that the probability of $\mathcal{E}$ distinguishing between $\mathrm{O}(\mathsf{real})$ and $\mathrm{O}(\mathsf{auth})$ is bounded above by the sum of the advantages of the two adversaries.

The adversary $A_{\mathrm{cpa}}^{O(\cdot,\cdot)}(1^\eta, a)$ is defined exactly like A_{cca} except that upon decryption it computes the plaintext m that is returned to $\mathcal{E}$ by
$$m := \begin{cases} m' & \text{if } \exists! m' \colon (m', c) \text{ stored} \\ \bot & \text{otherwise.} \end{cases}$$

Note that this is the only case where A_{cca} possibly uses its decryption oracle O_2. Hence, A_{cpa} is a polynomial-time IND-CPA adversary.

It is easy to see that

$$\Pr\left[\text{Exp}^{\text{ind-cpa-1}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)=1\right]=\Pr[(\mathcal{E}\,|\,\text{O}(\mathsf{auth}))(1^\eta,a)\rightsquigarrow 1]\ . \tag{1}$$

Let $B(1^\eta,a)$ be the event that in a run of $(\mathcal{E}\,|\,\text{O}(\mathsf{real}))(1^\eta,a)$ $\mathcal{E}$ sends (Dec,c) to Oracle such that $\text{dec}(k,c)\neq\perp$ and c has never been output by Oracle before. It is easy to find an injective mapping of runs of $(\mathcal{E}\,|\,\text{O}(\mathsf{real}))(1^\eta,a)$ where $B(1^\eta,a)$ does not occur to runs of $\text{Exp}^{\text{ind-cpa-0}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)$ with the same output and probability. By Theorem 4 we obtain that

$$\left|\Pr\left[\text{Exp}^{\text{ind-cpa-0}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)=1\right]-\Pr[(\mathcal{E}\,|\,\text{O}(\mathsf{real}))(1^\eta,a)\rightsquigarrow 1]\right|\leq\Pr[B(1^\eta,a)]\ . \tag{2}$$

We conclude

$$\begin{aligned}\text{Adv}^{\text{ind-cpa}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)&=\left|\Pr\left[\text{Exp}^{\text{ind-cpa-1}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)=1\right]-\Pr\left[\text{Exp}^{\text{ind-cpa-0}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)=1\right]\right|\\&\overset{(1),(2)}{\geq}\left|\Pr[(\mathcal{E}\,|\,\text{O}(\mathsf{auth}))(1^\eta,a)\rightsquigarrow 1]-\Pr[(\mathcal{E}\,|\,\text{O}(\mathsf{real}))(1^\eta,a)\rightsquigarrow 1]\right|-\Pr[B(1^\eta,a)]\ .\end{aligned}$$

Since Σ is IND-CPA secure $\text{Adv}^{\text{ind-cpa}}_{\Sigma,A_{\text{cpa}}}(1^\eta,a)$ is negligible and it suffices to show that $\Pr[B(1^\eta,a)]$ is negligible too.

We define the adversary $A^{O_1(\cdot),O_2(\cdot)}_{\text{int-ctxt}}(1^\eta,a)$ which first sets the variable $state:=\perp$ and then simulates a run of $\mathcal{E}(1^\eta,a)$ as follows:

1. If $\mathcal{E}$ outputs (KeyGen) on $T^{\text{in}}_{\text{oracle}}$ then $A_{\text{int-ctxt}}$ sets $state:=\mathsf{ok}$ and sends (Ack) to $T^{\text{out}}_{\text{oracle}}$.
2. If $\mathcal{E}$ outputs (Enc,m) on $T^{\text{in}}_{\text{oracle}}$ then $A_{\text{int-ctxt}}$ sends ε to start if $state\neq\mathsf{ok}$ or $m\notin\text{dom}(L)_\eta$, and, otherwise, computes $c\leftarrow O_1(m)$ and sends $(\mathsf{Ciphertext},c)$ to $T^{\text{out}}_{\text{oracle}}$.
3. If $\mathcal{E}$ outputs (Dec,c) on $T^{\text{in}}_{\text{oracle}}$ then $A_{\text{int-ctxt}}$ sends ε to start if $state\neq\mathsf{ok}$, and, otherwise, computes $m:=O_2(c)$ and sends $(\mathsf{Plaintext},m)$ to $T^{\text{out}}_{\text{oracle}}$.
4. If $\mathcal{E}$ terminates then $A_{\text{int-ctxt}}$ terminates.

Note that $A_{\text{int-ctxt}}$ is a polynomial-time INT-CTXT adversary because $\mathcal{E}$ is polynomial-time.

It is easy to see that

$$\text{Adv}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta,a)=\Pr[\text{Exp}^{\text{int-ctxt}}_{\Sigma,A_{\text{int-ctxt}}}(1^\eta,a)=1]=\Pr[B(1^\eta,a)]\ .$$

Hence, $\Pr[B(1^\eta,a)]$ is negligible because Σ is INT-CTXT secure, which concludes the proof. □

4 Bootstrapping Symmetric Key Encryption

In this section, we describe the two functionalities $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ for bootstrapping symmetric key encryption, as mentioned in the introduction.

4.1 Symmetric Key Encryption with Long-Term Keys

The functionality $!\mathcal{F}_{\text{ltsenc}}$ is a single session but multi-party functionality for pre-shared (or exchanged) long-term keys between two or more parties. We distinguish between an authenticated and an unauthenticated version of $\mathcal{F}_{\text{ltsenc}}$, denoted by $\mathcal{F}^{\text{auth}}_{\text{ltsenc}}$ and $\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$, respectively. We show that realizing $!\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$ and $!\mathcal{F}^{\text{auth}}_{\text{ltsenc}}$ by a symmetric encryption scheme is equivalent to the scheme being IND-CCA or IND-CPA and INT-CTXT secure, respectively. Furthermore, we give a joint state realization for both variants of $!\mathcal{F}_{\text{ltsenc}}$.

4.1.1 The Functionality

One instance of the functionality $\mathcal{F}_{\text{ltsenc}}$ provides two (or more) parties with a way to ideally encrypt and decrypt messages/ciphertexts under the same shared symmetric key. For this purpose, the parties first send a key exchange command to $\mathcal{F}_{\text{ltsenc}}$ and the adversary then provides encryption and decryption algorithms (with a built-in symmetric key), as described above. The key is meant to model a long-term shared key between the parties, and never leaves the functionality. We distinguish between an authenticated and an unauthenticated version of $\mathcal{F}_{\text{ltsenc}}$, denoted by $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$, respectively.

Both variants take as a parameter a leakage algorithm L and a polynomial q. Furthermore, the names of input and output tapes are parameterized by (sets of) tape names $\mathcal{T}_{\text{users}}$ and T_{adv}. The polynomial q is used to bound the length and runtime of the encryption and decryption algorithms received from the adversary, see Section 3.1 for remarks on running external code.

In mode CheckAddress, $\mathcal{F}_{\text{ltsenc}}$ only accepts messages prefixed by the set of parties *pids* that share this key. This set is defined by the first activation of $\mathcal{F}_{\text{ltsenc}}$. Only these parties are able to use the functionality. When a party requests $\mathcal{F}_{\text{ltsenc}}$, e.g., for encryption, then it provides its party ID p and $\mathcal{F}_{\text{ltsenc}}$ checks if $p \in pids$. The multi-party version $!\mathcal{F}_{\text{ltsenc}}$ of $\mathcal{F}_{\text{ltsenc}}$ provides symmetric encryption functionalities with long-term keys for an unbounded number of sets of parties. Each instance of $\mathcal{F}_{\text{ltsenc}}$ is used by one set of parties. The functionalities $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ will use $!\mathcal{F}_{\text{ltsenc}}$.

Next, we describe mode Compute of $\mathcal{F}_{\text{ltsenc}}$ informally. See Figure 2 for a precise definition.

In a key exchange phase the parties declare that they are willing to exchange a key with the other parties. These request are forwarded to the adversary who is required to provide encryption and decryption algorithms. In this phase the adversary also decides whether or not she wants to corrupt the functionality (static corruption). Encryption and decryption requests are then processed locally, i.e., without further involvement of the adversary. Our functionality can handle an unbounded number of encryption and decryption requests, with no bounds on the length of messages and ciphertexts.

If the functionality is requested 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, using the encryption algorithm and provided by the adversary, where the encryption algorithm is simulated for a polynomial number of steps, see above. This results in some ciphertext c. The functionality then stores the pair (m, c). Hence, even though the adversary knows the keys, only $L(1^\eta, m)$ (e.g., the length of m) will be leaked by the ciphertext. This is independent of the algorithms and keys provided by the adversary. Therefore when using $\mathcal{F}_{\text{ltsenc}}$ in the analysis of systems, one can abstract from these algorithms completely.

Upon a decryption request for a ciphertext c (which 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. If there is none such pair, the following is done: In the authenticated variant of $\mathcal{F}_{\text{ltsenc}}$, which is supposed to model authenticated encryption, an error message is returned. In the unauthenticated variant of $\mathcal{F}_{\text{ltsenc}}$, c is decrypted with the decryption algorithm provided by the adversary and the result is returned.

In case the functionality is corrupted, encryption and decryption are not handled ideally but as in the real system, i.e., the ciphertext/plaintext is computed by encrypting/decrypting m/c with the encryption/decryption algorithm provided by the adversary. Note that the adversary does not obtain full control over the party's behavior because corruption here models that the key is known (or exposed) to the adversary. It does not model that the party itself is corrupt or dishonest. This however is not restricting because corrupt or dishonest behavior of the party can be captured one layer above in the definition of the protocol that uses the functionality. For example, see Section 7.1 where we analyze a key exchange protocol using $\mathcal{F}_{\text{senc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}}^{\text{auth}}$. Alternatively, one could define $\mathcal{F}_{\text{ltsenc}}$ with a more involved corruption behavior as done for public key encryption in [32].

We note that if the functionality is used with a leakage that has high entropy, then it guarantees that unknown ciphertext cannot be guessed. Let us explain: Assume that, e.g., due to nested encryption, a ciphertext c was generated by $\mathcal{F}_{\text{ltsenc}}$ and that c is not known to the adversary because it was never output to the adversary. If the leakage has high entropy, the following is easy to see: The adversary has only negligible guessing probability for all ciphertexts that are stored in *decTable* in $\mathcal{F}_{\text{senc}}$ and which are formally unknown to the adversary. The proof idea is to exploit that the ciphertext has to contain as much information as $L(1^\eta, m)$, because of the decryption test during encryption. Since the leakage has

$\mathcal{F}_{\text{ltsenc}}(q, L, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$

Tapes: in: T^{in} for all $T \in \mathcal{T}_{\text{users}}$ (enriching I/O tapes), $T^{\text{in}}_{\text{adv}}$ (network tape)
out: T^{out} for all $T \in \mathcal{T}_{\text{users}}$ (I/O tapes), $T^{\text{out}}_{\text{adv}}$ (network tape)

State: $pids \subseteq \text{PIDS}$ (initially $\emptyset$)
$state\colon \text{PIDS} \dashrightarrow \{\mathsf{wait}\} \times \mathcal{T}_{\text{users}} \cup \{\mathsf{ok}\}$ (partially defined, initially domain $= \emptyset$)
$enc, dec\colon \text{PIDS} \dashrightarrow \{0,1\}^*$ (partially defined, initially domain $= \emptyset$)
$decTable \subseteq \{0,1\}^* \times \{0,1\}^*$ (initially $\emptyset$)
$corrupted, incorruptible \in \{\mathsf{false}, \mathsf{true}\}$ (initially false)

ChkAdr: Upon first activation accept only messages of shape $(pids', m)$ on all I/O tapes where $pids' \subseteq \text{PIDS}$. In mode $\mathsf{Compute}$, at first set $pids := pids'$.
Later accept only messages of shape $(pids', m)$ on all tapes where $pids = pids'$.

1. KEY EXCHANGE: Upon receiving $(pids, p, \mathsf{KeyExchange})$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$ do: Send $(pids, p, \mathsf{Completed})$ to T if $state(p) = \mathsf{ok}$, otherwise, set $state(p) := (\mathsf{wait}, T)$ and send $(pids, p, \mathsf{KeyExchange})$ to T_{adv}.
2. COMPLETE: Upon receiving $(pids, p, \mathsf{Complete}, enc, dec)$ from T_{adv} where $p \in pids$, $state(p) = (\mathsf{wait}, T)$ for some T, and $|enc|, |dec| \leq q(\eta)$ (Note: peers do not have to have sent $\mathsf{KeyExchange}$) do: $state(p) := \mathsf{ok}$, $enc(p) := enc$, $dec(p) := dec$, and $incorruptible := \mathsf{true}$ and send $(pids, p, \mathsf{Completed})$ to T.
3. CORRUPTION: Upon receiving $(pids, \mathsf{Corrupt})$ from T_{adv} where $|\operatorname{dom}(state)| > 0$, not *incorruptible* and not *corrupted* do: $corrupted := \mathsf{true}$ and send $(pids, \mathsf{Corrupted})$ to T_{adv}.
4. CORRUPTION REQUEST: Upon receiving $(pids, \mathsf{Corrupted?})$ from $T \in \mathcal{T}_{\text{users}}$ where exists $p \in pids$ s.t. $state(p) = \mathsf{ok}$ do: Send $(pids, \mathsf{CorruptionState}, corrupted)$ to T.
5. ENCRYPTION: Upon receiving $(pids, p, \mathsf{Enc}, m)$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$: If $state(p) \neq \mathsf{ok}$ or $m \notin \operatorname{dom}(L)_\eta$ then send $(pids, p, \mathsf{Ciphertext}, \bot)$ to T, otherwise:
   if not *corrupted* **then**
   $\overline{m} \leftarrow L(1^\eta, m)$, $c \leftarrow \text{sim}_{q(\eta+|\overline{m}|)}(enc(p), \overline{m})$
   if $\text{sim-det}_{q(\eta+|c|)}(dec(p), c) = \overline{m}$ **then**
   $decTable := decTable \cup \{(m, c)\}$, **send** $(pids, p, \mathsf{Ciphertext}, c)$ **to** T
   else
   send $(pids, p, \mathsf{Ciphertext}, \bot)$ **to** T
   else {*corrupted*}
   $c \leftarrow \text{sim}_{q(\eta+|m|)}(enc(p), m)$, **send** $(pids, p, \mathsf{Ciphertext}, c)$ **to** T
6. DECRYPTION: Upon receiving $(pids, p, \mathsf{Dec}, c)$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$: If $state(p) \neq \mathsf{ok}$ then send $(pids, p, \mathsf{Plaintext}, \bot)$ to T, otherwise:
   if not *corrupted* **then**
   $$m := \begin{cases} m' & \text{if } \exists! m'\colon (m', c) \in decTable \quad (\text{exists unique } m') \\ \text{sim-det}_{q(\eta+|c|)}(dec(p), c) & \text{if variant } \mathcal{F}^{\text{unauth}}_{\text{ltsenc}} \text{ and } \forall m'\colon (m', c) \notin decTable \\ \bot & \text{otherwise} \end{cases}$$
   send $(pids, p, \mathsf{Plaintext}, m)$ to T
   else {*corrupted*}
   $m := \text{sim-det}_{q(\eta+|c|)}(dec(p), c)$, **send** $(pids, p, \mathsf{Plaintext}, m)$ **to** T

Figure 2: Functionality $!\mathcal{F}_{\text{ltsenc}}$ for symmetric encryption with long-term keys. Both variants $\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$ and $\mathcal{F}^{\text{auth}}_{\text{ltsenc}}$.

$\mathcal{P}_{\text{ltsenc}}(\Sigma = (\text{gen}, \text{enc}, \text{dec}), (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$

Tapes: in: T^{in} (enriching) for all $T \in \mathcal{T}_{\text{users}}$, $T^{\text{out}}_{\text{keysetup}}$ (I/O tapes), $T^{\text{in}}_{\text{adv}}$ (network tape)
out: T^{out} for all $T \in \mathcal{T}_{\text{users}}$, $T^{\text{in}}_{\text{keysetup}}$ (I/O tapes), $T^{\text{out}}_{\text{adv}}$ (network tape)

State: $pids \subseteq \text{PIDS}$ (initially $\emptyset$)
$state\colon \text{PIDS} \dashrightarrow \{\mathsf{wait}\} \times \mathcal{T}_{\text{users}} \cup \{\mathsf{ok}\}$ (partially defined, initially domain $= \emptyset$)
$k\colon \text{PIDS} \dashrightarrow \{0,1\}^*$ (partially defined, initially domain $= \emptyset$)
$corrupted, incorruptible \in \{\mathsf{false}, \mathsf{true}\}$ (initially false)

ChkAdr: Like $\mathcal{F}_{\text{ltsenc}}$ (see Figure 2).

1. KEY EXCHANGE: Upon receiving $(pids, p, \mathsf{KeyExchange})$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$: Send $(pids, p, \mathsf{Completed})$ to T if $state(p) = \mathsf{ok}$, otherwise, set $state(p) := (\mathsf{wait}, T)$ and send $(pids, p, \mathsf{KeyExchange})$ to T_{adv}.
2. COMPLETE: Upon receiving $(pids, p, \mathsf{Complete}, k')$ from T_{adv} where $state(p) = (\mathsf{wait}, T)$ for some T and $|k'| \leq q(\eta)$ (where q is the polynomial associated with Σ that bounds the runtime of the algorithms):
   $state(p) := \mathsf{ok}$, $incorruptible := \mathsf{true}$
   if *corrupted* **then**
   $k(p) := k'$
   else {**not** *corrupted*}
   send $(pids, \mathsf{GetKey})$ **to** T_{keysetup} , **recv** $(pids, \mathsf{Key}, k'')$ **from** T_{keysetup} , $k(p) := k''$
   send $(pids, p, \mathsf{Completed})$ **to** T
3. CORRUPTION: Upon receiving $(pids, \mathsf{Corrupt})$ from T_{adv} where $|\operatorname{dom}(state)| > 0$, not *incorruptible*, and not *corrupted*: $corrupted := \mathsf{true}$ and send $(pids, \mathsf{Corrupted})$ to T_{adv}.
4. CORRUPTION REQUEST: Upon receiving $(pids, \mathsf{Corrupted?})$ from $T \in \mathcal{T}_{\text{users}}$ where exists $p \in pids$ s.t. $state(p) = \mathsf{ok}$: Send $(pids, \mathsf{CorruptionState}, corrupted)$ to T.
5. ENCRYPTION: Upon receiving $(pids, p, \mathsf{Enc}, m)$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$:
   Send $(pids, p, \mathsf{Ciphertext}, c)$ to T where $c \leftarrow \text{enc}(k(p), m)$ if $state(p) = \mathsf{ok}$ and $m \in \operatorname{dom}(\Sigma)_\eta$, otherwise, $c := \bot$.
6. DECRYPTION: Upon receiving $(pids, p, \mathsf{Dec}, c)$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$: Send $(pids, p, \mathsf{Plaintext}, m)$ to T where $m := \text{dec}(k(p), c)$ if $state(p) = \mathsf{ok}$, otherwise, $c := \bot$.

Figure 3: Protocol $!\mathcal{P}_{\text{ltsenc}}$ for symmetric encryption with long-term keys.

high entropy, $L(1^\eta, m)$ is sufficiently random and can be guessed only with negligible probability.

4.1.2 Realizing the Functionality

We show that realizing $!\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$ and $!\mathcal{F}^{\text{auth}}_{\text{ltsenc}}$ by a symmetric encryption scheme together with a key setup functionality which provides the parties with pre-shared keys is equivalent to the scheme being IND-CCA or IND-CPA and INT-CTXT secure, respectively.

The protocol $\mathcal{P}_{\text{ltsenc}}$ is parameterized by a symmetric encryption scheme Σ and (sets of) tape names. It provides the same interfaces as $\mathcal{F}_{\text{ltsenc}}$. See Figure 3 for a precise formulation. In the key exchange phase, the adversary has the ability to statically corrupt the key in which case she is requested to provide the corrupted key which is then used by the parties. In the uncorrupted case, $\mathcal{P}_{\text{ltsenc}}$ obtains the secret key from an ideal key setup functionality $\mathcal{F}_{\text{keysetup}}$ which is given in Figure 4. See Figure 5 for on overview of the connections between the machines.

Theorem 5. *Let L be a leakage algorithm, which leaks exactly the length of a message, and $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ be a symmetric encryption scheme with domain $\operatorname{dom}(L)$. Then, for all sufficiently large polynomials q (i.e., where all algorithms in Σ are bounded by q) it holds*

1. *Σ is IND-CCA secure iff $!\mathcal{P}_{\text{ltsenc}}(\Sigma, \hat{T}) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \leq^{SS} !\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}(q, L, \mathcal{T})$, and*
2. *Σ is IND-CPA and INT-CTXT secure iff $!\mathcal{P}_{\text{ltsenc}}(\Sigma, \hat{T}) \,|\, \mathcal{F}_{\text{keysetup}}(\text{gen}) \leq^{SS} !\mathcal{F}^{\text{auth}}_{\text{ltsenc}}(q, L, \mathcal{T})$,*

$\mathcal{F}_{\text{keysetup}}(\text{gen})$

Tapes: enriching I/O input tape $T^{\text{in}}_{\text{keysetup}}$, I/O output tape $T^{\text{out}}_{\text{keysetup}}$

State: $k\colon 2^{\text{PIDS}} \dashrightarrow \{0,1\}^*$ (partially defined, initially domain $= \emptyset$)

1. KEY REQUEST: Upon receiving $(pids, \mathsf{GetKey})$ from T_{keysetup} where $pids \subseteq \text{PIDS}$:
   If $pids \notin \text{dom}(k)$ then $k(pids) \leftarrow \text{gen}(1^\eta)$. Send $(pids, \mathsf{Key}, k(pids))$ to T_{keysetup}.

Figure 4: Key setup functionality $\mathcal{F}_{\text{keysetup}}$.

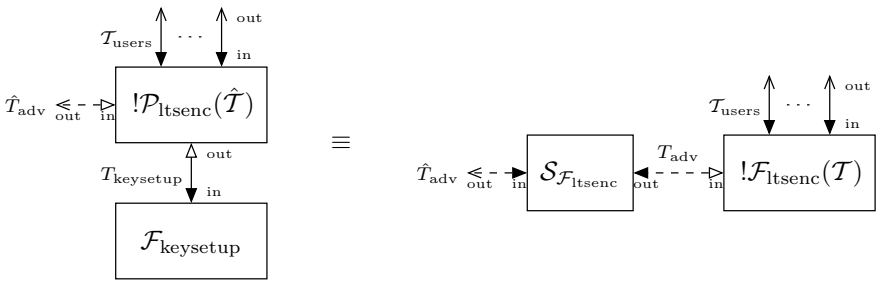

Figure 5: Theorem 5, $!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}$ and $\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}} \mid !\mathcal{F}_{\text{ltsenc}}$ are indistinguishable. Solid lines represent I/O tapes and dashed lines network tapes. Filled arrow heads represent enriching and unfilled arrow heads consuming input tapes.

for all disjoint (sets of) tape names $\mathcal{T}_{\text{users}}, T_{\text{adv}}, \hat{T}_{\text{adv}}$ *where* $\mathcal{T} = (\mathcal{T}_{\text{users}}, T_{\text{adv}})$ *and* $\hat{\mathcal{T}} = (\mathcal{T}_{\text{users}}, \hat{T}_{\text{adv}})$.
The directions from left to right hold for any length preserving leakage algorithm L.

Proof. The direction from right to left is easy to prove. Given any IND-CCA, IND-CPA or INT-CTXT adversary A we can easily construct an environment $\mathcal{E}$ of $!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}$ such that the advantage of A is bound by the advantage of $\mathcal{E}$ distinguishing between $!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}$ and $\mathcal{S} \mid !\mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$ or $\mathcal{S} \mid !\mathcal{F}^{\text{auth}}_{\text{ltsenc}}$, respectively, for any simulator $\mathcal{S}$. The environment $\mathcal{E}$ simply exchanges an uncorrupted key for two parties and simulates A by using this key for encryption and decryption to simulate the oracles of A.

Next, we show the direction from left to right. The definition of the simulator $\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}}$ is straight forward. It has two enriching network input tapes $T^{\text{out}}_{\text{adv}}$ and $\hat{T}^{\text{in}}_{\text{adv}}$ and two output tapes $T^{\text{in}}_{\text{adv}}$ and $\hat{T}^{\text{out}}_{\text{adv}}$ to connect to $!\mathcal{F}_{\text{ltsenc}}$ and the environment, respectively, see Figure 5 for on overview of the connections between the IITMs. It treats every instance of $\mathcal{F}_{\text{ltsenc}}$ separately. By $\mathcal{F}_{\text{ltsenc}}[pids]$ we denote the instance of $\mathcal{F}_{\text{ltsenc}}$ for the parties $pids$. In the uncorrupted case, upon completion it simply generates a key $k \leftarrow \text{gen}(1^\eta)$ (only one for all parties in $pids$) and provides the (description of the) algorithms $enc(\cdot) = \text{enc}(k, \cdot)$ and $dec(\cdot) = \text{dec}(k, \cdot)$ to $\mathcal{F}_{\text{ltsenc}}[pids]$. In the corrupted case it uses the key k' contained in the completion message and sends the algorithms $e = \text{enc}(k', \cdot)$ and $d = \text{dec}(k', \cdot)$ to $\mathcal{F}_{\text{ltsenc}}[pids]$. The messages $(pids, p, \mathsf{KeyExchange})$ are forwarded from $\mathcal{F}_{\text{ltsenc}}[pids]$ to the environment. One easily verifies that $\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}}$ is a simulator for $!\mathcal{F}_{\text{ltsenc}}$ w.r.t. $!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}$, i.e., $\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}} \in \text{Sim}^{!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}}_{\mathbf{S}}(!\mathcal{F}_{\text{ltsenc}})$.

Next, we show that $\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}}$ is a successful simulator. We use that $\text{Oracle}(\Sigma, \mathsf{real}, L)$ is indistinguishable from $\text{Oracle}(\Sigma, \mathsf{unauth}, L)$ (resp., $\text{Oracle}(\Sigma, \mathsf{auth}, L)$).

We define an intermediate system $\mathcal{Q}$ which behaves like $!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}}$ except that it connects to the identifier version $!\underline{\text{Oracle}}$ (as defined in Section 3.3) and if $\mathcal{P}_{\text{ltsenc}}[pids]$ is not corrupted then upon encryption and decryption it uses $\underline{\text{Oracle}}[pids]$ (i.e., $pids$ is used as the identifier and the prefix of all messages sent to $\underline{\text{Oracle}}[pids]$) to compute the ciphertext and plaintext, respectively. If $\mathcal{P}_{\text{ltsenc}}[pids]$ is corrupted then $\underline{\text{Oracle}}$ is not used but $\mathcal{Q}$ behaves like $\mathcal{P}_{\text{ltsenc}}[pids]$ upon encryption and decryption, respectively, i.e. it uses the corrupted key.

For any $\mathcal{E} \in \text{Con}_{\mathcal{E}}(!\mathcal{P}_{\text{ltsenc}} \mid \mathcal{F}_{\text{keysetup}})$, by definition of $\mathcal{Q}$, we have

$$\mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{real}, L)} \equiv_0 \mathcal{E} \mid !\mathcal{P}_{\text{ltsenc}}(\Sigma) \mid \mathcal{F}_{\text{keysetup}}(\text{gen}) \ .$$

Because q is chosen large enough such that $\mathcal{F}_{\text{ltsenc}}$ can always compute enc and dec, we can prove that

$$\mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{unauth}, L)} \equiv_0 \mathcal{E} \mid \mathcal{S}_{\mathcal{F}_{\text{ltsenc}}} \mid !\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}(q, L) \ , \text{ and}$$
$$\mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{auth}, L)} \equiv_0 \mathcal{E} \mid \mathcal{S}_{\mathcal{F}_{\text{ltsenc}}} \mid !\mathcal{F}_{\text{ltsenc}}^{\text{auth}}(q, L) \ .$$

By Lemma 1 and the composition theorem, we obtain

$$\mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{real}, L)} \equiv \mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{unauth}, L)} \ , \text{ and}$$
$$\mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{real}, L)} \equiv \mathcal{E} \mid \mathcal{Q} \mid !\underline{\text{Oracle}(\Sigma, \mathsf{auth}, L)} \ .$$

By transitivity of $\equiv$, we are done. □

Instead of using $\mathcal{F}_{\text{keysetup}}$ as a setup assumption, one could realize $\mathcal{F}_{\text{ltsenc}}$ by a key exchange protocol. The proof will be almost identical to the proof above if one uses an ideal key exchange functionality (e.g. [20]) which replaces $\mathcal{F}_{\text{keysetup}}$. Then, by the composition theorem one obtains a realization of $\mathcal{F}_{\text{ltsenc}}$ for any key exchange protocol which realizes the key exchange functionality.

4.1.3 Joint State Realization

As first explicitly studied by [22], see also [32], the composition theorem itself does not immediately yield practical realizations for multi-session, multi-party protocols. This is also the case for the multi-session version $!\underline{\mathcal{F}_{\text{ltsenc}}}$ of $!\mathcal{F}_{\text{ltsenc}}$. Here, we would have a different instance of $\underline{\mathcal{F}_{\text{ltsenc}}}$, i.e., a different fresh key, for every set of parties *pids* for every session. A practical realization can be obtained by a composition theorem that allows for joint state. The idea is to prove a theorem like $\mathcal{P}_{\text{ltsenc}}^{\text{js}} \mid !\mathcal{F}_{\text{ltsenc}} \leq^{SS} !\underline{\mathcal{F}_{\text{ltsenc}}}$ where $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$, the joint state realization, uses for every set of parties *pids* only a singe instance of $\mathcal{F}_{\text{ltsenc}}$ to realize the instances of $\underline{\mathcal{F}_{\text{ltsenc}}}$ for these parties in all sessions.

We give a joint state realization $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ for both variants of $\mathcal{F}_{\text{ltsenc}}$. The basic idea is that before encryption the plaintexts are prefixed by the session identifier. The same method has been used in [22] for digital signatures and in [32] for public key encryption.

The protocol $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ is parameterized by a polynomial q and a set of tape names. The polynomial q is used to bound the length of the session identifier. This is necessary as pointed out in [31]. $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ has enriching I/O tapes as $\mathcal{F}_{\text{ltsenc}}$ for parties and the environment to connect to. Furthermore, it has consuming I/O tapes to connect to $!\mathcal{F}_{\text{ltsenc}}$. Note that $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ will connect to all I/O tapes of $\mathcal{F}_{\text{ltsenc}}$, in particular to all enriching tapes. Thus, $\mathcal{F}_{\text{ltsenc}}$ only receives enriching input from $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ and $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ will always be able to forward messages from $\mathcal{F}_{\text{ltsenc}}$. The protocol $\mathcal{P}_{\text{ltsenc}}^{\text{js}}$ accepts message of the form $(sid, pids, m)$ from the environment and of the form $(pids, m)$ form $\mathcal{F}_{\text{ltsenc}}$. See Figure 6 for a precise definition and Figure 7 for an overview of the connection to $\mathcal{F}_{\text{ltsenc}}$.

Theorem 6. *For all polynomials q, q' and disjoint (sets of) tape names $\mathcal{T}_{\text{users}}, T_{\text{adv}}$ there exists a polynomial q'' such that for every leakage algorithm L we have*

$$!\mathcal{P}_{\text{ltsenc}}^{\text{js}}(q, \mathcal{T}_{\text{users}}) \mid !\mathcal{F}_{\text{ltsenc}}(q', L', \hat{\mathcal{T}}) \leq^{SS} !\underline{\mathcal{F}_{\text{ltsenc}}(q'', L, \mathcal{T})}$$

where $L'(1^\eta, (sid, m)) = (sid, L(1^\eta, m))$ for all $sid \in$ SIDS and $m \in \text{dom}(L)_\eta$, i.e., $\text{dom}(L') = (\{(sid, m) \mid sid \in \text{SIDS}, m \in \text{dom}(L)_\eta\})_{\eta \in \mathbb{N}}$, $\mathcal{T} = (\mathcal{T}_{\text{users}}, T_{\text{adv}})$, and $\hat{\mathcal{T}} = (\hat{\mathcal{T}}_{\text{users}}, \hat{T}_{\text{adv}})$ ($\hat{\mathcal{T}}_{\text{users}} = \{\hat{T} \mid T \in \mathcal{T}_{\text{users}}\}$).

This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$.

Proof. The proof is very similar to the proof of the joint state theorem for public key encryption [31].

First, we define a simulator $!\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}}^{\text{js}}$ and then sketch the proof that there is a $(0, 0, 0)$-probabilistic trace relation between runs of the joint state (JS) world $(\text{runs}_\eta^a(\mathcal{E} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{js}} \mid !\mathcal{F}_{\text{ltsenc}}))$ and runs of the ideal world $(\text{runs}_\eta^a(\mathcal{E} \mid !\mathcal{S}_{\mathcal{F}_{\text{ltsenc}}}^{\text{js}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}}))$, recall the definitions from Section 2.4.

The simulator is given in Figure 8. When the simulator receives the algorithms *enc* and *dec* it forwards the algorithms enc_{sid} and dec_{sid} to the instance of $\underline{\mathcal{F}_{\text{ltsenc}}}$ with session ID *sid*. The idea is that $enc_{sid}(m) = enc((sid, m))$ and that $dec_{sid}(c) = m$ if $dec(c) = (sid, m)$ for some m, and $dec_{sid}(c) = \bot$ otherwise. The difficulty is that we simulate the algorithms only a polynomial number of steps, e.g. in the JS world *enc* is simulated $q'(\eta + |(sid, m)|)$ steps while in the ideal world enc_{sid} is simulated $q''(\eta + |m|)$

$\mathcal{P}^{\text{js}}_{\text{ltsenc}}(q, \mathcal{T}_{\text{users}})$

Tapes: I/O in: T^{in} (enriching), $\hat{T}^{\text{out}}$ for all $T \in \mathcal{T}_{\text{users}}$
I/O out: T^{out}, $\hat{T}^{\text{in}}$ for all $T \in \mathcal{T}_{\text{users}}$

State: $pids \subseteq \textsc{Pids}$ (initially $\emptyset$)
$sids\colon \textsc{Pids} \dashrightarrow \textsc{Sids}$ (initially domain $= \emptyset$)
$state\colon \textsc{Pids} \dashrightarrow \{\mathsf{wait}\} \times \textsc{Sids} \cup \{\mathsf{ok}\}$ (initially domain $= \emptyset$)

ChkAdr: Upon first activation accept only messages of shape $(sid, pids', m)$ on all enriching I/O tapes where $pids' \subseteq \textsc{Pids}$, $sid \in \textsc{Sids}$ and $|sid| \leq q(\eta)$. In mode $\mathsf{Compute}$, at first do $pids := pids'$.
Later accept only messages of shape $(sid, pids', m)$ from T or $(pids', m)$ from $\hat{T}$ where $T \in \mathcal{T}_{\text{users}}$, $pids = pids'$, and (if applicable) $sid \in \textsc{Sids}$ and $|sid| \leq q(\eta)$.

1. Upon receiving $(sid, pids, p, \mathsf{KeyExchange})$ from $T \in \mathcal{T}_{\text{users}}$ where $p \in pids$:
   if $state(p) = \mathsf{ok}$ **then**
   $sids(p) := sids(p) \cup \{sid\}$, **send** $(sid, pids, p, \mathsf{Completed})$ **to** T
   else
   $state(p) := (\mathsf{wait}, sid)$, **send** $(pids, p, \mathsf{KeyExchange})$ **to** $\hat{T}$
2. Upon receiving $(pids, p, \mathsf{Completed})$ from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$, $state(p) = (\mathsf{wait}, sid)$ for some sid: $state(p) := \mathsf{ok}$, $sids(p) := \{sid\}$, and send $(sid, pids, p, \mathsf{Completed})$ to T.
3. Upon receiving $(sid, pids, p, \mathsf{Enc}, m)$ from $T \in \mathcal{T}_{\text{users}}$:
   if $sid \notin sids(p)$ **then**
   send $(sid, pids, p, \mathsf{Ciphertext}, \bot)$ **to** T
   else
   send $(pids, p, \mathsf{Enc}, (sid, m))$ **to** $\hat{T}$, **recv** $(pids, p, \mathsf{Ciphertext}, c)$ **from** $\hat{T}$
   send $(sid, pids, p, \mathsf{Ciphertext}, c)$ **to** T
4. Upon receiving $(sid, pids, p, \mathsf{Dec}, c)$ from $T \in \mathcal{T}_{\text{users}}$:
   if $sid \notin sids(p)$ **then**
   send $(sid, pids, p, \mathsf{Plaintext}, \bot)$ **to** T
   else
   send $(pids, p, \mathsf{Dec}, c)$ **to** $\hat{T}$, **recv** $(pids, p, \mathsf{Plaintext}, m)$ **from** $\hat{T}$
   if m can be parsed as (sid, m') for some m' **then**
   send $(sid, pids, p, \mathsf{Plaintext}, m')$ **to** T
   else
   send $(sid, pids, p, \mathsf{Plaintext}, \bot)$ **to** T
5. Upon receiving $(sid, pids, \mathsf{Corrupted?})$ from $T \in \mathcal{T}_{\text{users}}$ where $\exists p \in pids\colon sid \in sids(p)$: Send $(pids, \mathsf{Corrupted?})$ to $\hat{T}$ and wait for receiving $(pids, \mathsf{CorruptionState}, corrupted)$ from $\hat{T}$ where $corrupted \in \{\mathsf{false}, \mathsf{true}\}$. Send $(sid, pids, \mathsf{CorruptionState}, corrupted)$ to T

Figure 6: Joint state realization $!\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ for symmetric encryption with long-term keys.

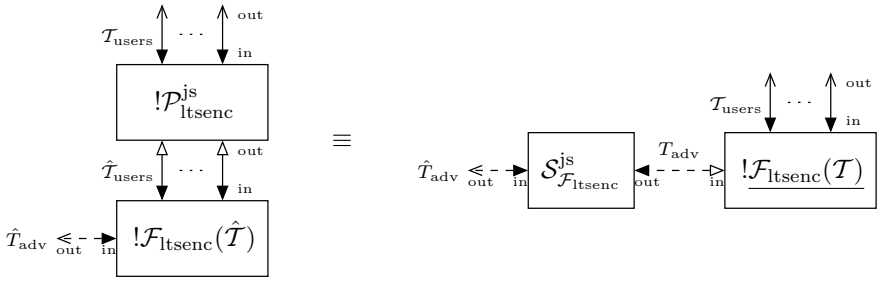

Figure 7: Theorem 6, $!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}}$ are indistinguishable. Solid lines represent I/O tapes and dashed lines network tapes. Filled arrow heads represent enriching and unfilled arrow heads consuming input tapes.

steps. Of course, for the simulation to work, they must have different runtime behavior. This is the reason why the session IDs have to be bounded polynomially in the security parameter. It is easy to see and was also shown for similar algorithms in [31] that for all polynomials q, q' there exists a polynomial q'' such that for all security parameters $\eta \in \mathbb{N}$, all session IDs sid with $|sid| \leq q(\eta)$, and all algorithms enc, dec with $|enc|, |dec| \leq q'(\eta)$ it holds that algorithms enc_{sid}, dec_{sid} can be constructed in polynomial time such that

1. $|enc_{sid}|, |dec_{sid}| \leq q''(\eta)$,
2. the probability distributions of $\text{sim}_{q''(\eta+|m|)}(enc_{sid}, m)$ and $\text{sim}_{q'(\eta+|(sid,m)|)}(enc, (sid, m))$ are equal for all $m \in \{0,1\}^*$, and
3. $\text{sim}_{q''(\eta+|c|)}(dec_{sid}, c) = m$ if $\text{sim-det}_{q'(\eta+|c|)}(dec, c) = (sid, m)$ for some $m \in \{0,1\}^*$, otherwise, $\text{sim}_{q''(\eta+|c|)}(dec_{sid}, c) = \perp$ for all $c \in \{0,1\}^*$.

Let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}})$. We define an injection α from runs of the runs of the ideal world $(\text{runs}^a_\eta(\mathcal{E} \mid !\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}}))$ to runs of JS world $(\text{runs}^a_\eta(\mathcal{E} \mid !\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}}))$ as follows. The state of $\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}}$ defines the state of $\mathcal{P}^{\text{js}}_{\text{ltsenc}}$ and the state of the instance of $\mathcal{F}_{\text{ltsenc}}$ for the parties $pids$ (we call this instance $\mathcal{F}_{\text{ltsenc}}[pids]$) is defined by the state of the instances of $\underline{\mathcal{F}_{\text{ltsenc}}}$ for the parties $pids$ for all session IDs sid (we call these instances $\underline{\mathcal{F}_{\text{ltsenc}}}[sid, pids]$). For example, the decryption table $decTable$ in $\mathcal{F}_{\text{ltsenc}}[pids]$ is set to the union of $\{((sid, m), c) \mid (m, c) \in decTable \text{ in } \underline{\mathcal{F}_{\text{ltsenc}}}[sid, pids]\}$ for all sid.

The crucial and simple observation is that there are no collisions between sessions, i.e., for all sid, sid', c, c', m, m' where $dec(c) = (sid, m)$ and $dec(c') = (sid', m')$ we have that $c \neq c'$ if $sid \neq sid'$. By the definition of the leakage algorithm L' we do not have collisions between sessions in the JS world. Now, it is easy to prove that for every run $\rho \in \text{runs}^a_\eta(\mathcal{E} \mid !\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}})$ of the JS world, we have that $\Pr[\rho] = \Pr[\alpha(\rho)]$. By Theorem 4 we obtain that $\mathcal{E} \mid !\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}} \equiv_0 \mathcal{E} \mid !\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{F}_{\text{ltsenc}}$. □

Note that the leakage algorithm L', as defined in the above theorem, is length preserving if the leakage algorithm L is length preserving. Hence, we can apply Theorem 5 and obtain that the protocol $!\mathcal{P}^{\text{js}}_{\text{ltsenc}} \mid !\mathcal{P}_{\text{ltsenc}}(\Sigma)$ realizes the multi-party multi-session functionality $!\underline{\mathcal{F}_{\text{ltsenc}}(L)}$ for any length preserving leakage algorithm L and IND-CCA (resp., IND-CPA and INT-CTXT) secure symmetric encryption scheme Σ.

4.2 Public Key Encryption

We use a slightly simplified version of the functionality $\mathcal{F}_{\text{pke}}$ for public key encryption as introduced in [32]. As shown in [32], the results transfer to the variant considered here, a public key encryption scheme realizes $\mathcal{F}_{\text{pke}}$ if and only if it is IND-CCA secure. Moreover, it is shown that there is a joint state realization of $\mathcal{F}_{\text{pke}}$.

The Functionality. One instance of $\mathcal{F}_{\text{pke}}$ can be used by one decryptor and arbitrary many encryptors. Intuitively, $\mathcal{F}_{\text{pke}}$ stands for one public/private key pair. The decryptor can use $\mathcal{F}_{\text{pke}}$ to (ideally) decrypt with the private key and the encryptors can use $\mathcal{F}_{\text{pke}}$ to (ideally) encrypt with the public key, as described above. The decryptor is first supposed to send a key generation command to $\mathcal{F}_{\text{pke}}$, upon which the

$\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}}(q, q', T_{\text{adv}})$

Tapes: enriching network in: $T^{\text{in}}_{\text{adv}}$ (denoted "from $\mathcal{F}_{\text{ltsenc}}$"), $\hat{T}^{\text{out}}_{\text{adv}}$ (denoted "from $\hat{T}_{\text{adv}}$")
network out: $T^{\text{out}}_{\text{adv}}$ (denoted "to $\mathcal{F}_{\text{ltsenc}}$"), $\hat{T}^{\text{in}}_{\text{adv}}$ (denoted "to $\hat{T}_{\text{adv}}$")

State: $pids \subseteq \textsc{Pids}$ (initially $\emptyset$)
$sids\colon \textsc{Pids} \dashrightarrow \textsc{Sids}$ (initially domain $= \emptyset$)
$state\colon \textsc{Pids} \dashrightarrow \{\mathsf{wait}\} \times \textsc{Sids} \cup \{\mathsf{ok}\}$ (initially domain $= \emptyset$)
$enc, dec\colon \textsc{Pids} \dashrightarrow \{0,1\}^*$ (initially domain $= \emptyset$)
$corrupted, incorruptible \in \{\mathsf{false}, \mathsf{true}\}$ (initially false)

ChkAdr: Upon first activation accept only messages of shape $(sid, pids', m)$ from $\mathcal{F}_{\text{ltsenc}}$ where $pids' \subseteq \textsc{Pids}$ and $sid \in \textsc{Sids}$. In mode $\mathsf{Compute}$, at first do $pids := pids'$.
Later accept only messages of shape $(sid, pids', m)$ from $\mathcal{F}_{\text{ltsenc}}$ or $(pids', m)$ from $\hat{T}_{\text{adv}}$ where $pids = pids'$ and (if applicable) $sid \in \textsc{Sids}$ and $|sid| \leq q(\eta)$.

1. Upon receiving $(sid, pids, p, \mathsf{KeyExchange})$ from $\mathcal{F}_{\text{ltsenc}}$:
   if $state(p) = \mathsf{ok}$ **then**
   $sids(p) := sids(p) \cup \{sid\}$, **send** $(sid, pids, p, \mathsf{Complete}, enc(p)_{sid}, dec(p)_{sid})$ **to** $\mathcal{F}_{\text{ltsenc}}$
   else
   $state(p) := (\mathsf{wait}, sid)$, **send** $(pids, p, \mathsf{KeyExchange})$ **to** $\hat{T}_{\text{adv}}$
2. Upon receiving $(pids, p, \mathsf{Complete}, enc, dec)$ from $\hat{T}_{\text{adv}}$ where $state(p) = (\mathsf{wait}, sid)$, for some sid, and $|enc|, |dec| \leq q'(\eta)$:
   $state(p) := \mathsf{ok}$, $sids(p) := \{sid\}$, $enc(p) := enc$, $dec(p) := dec$, $incorruptible := \mathsf{true}$
   send $(sid, pids, p, \mathsf{Complete}, enc(p)_{sid}, dec(p)_{sid})$ **to** $\mathcal{F}_{\text{ltsenc}}$
3. Upon receiving $(pids, \mathsf{Corrupt})$ from $\hat{T}_{\text{adv}}$ where not $incorruptible$, not $corrupted$, and $|\text{dom}(state)| > 0$:
   $corrupted := \mathsf{true}$
   for all $p \in pids$ **and** $sid \in sids(p)$ **do**
   send $(sid, pids, \mathsf{Corrupt})$ **to** $\mathcal{F}_{\text{ltsenc}}$, **recv** $(sid, pids, \mathsf{Corrupted})$ **from** $\mathcal{F}_{\text{ltsenc}}$
   send $(pids, \mathsf{Corrupted})$ **to** $\hat{T}_{\text{adv}}$

Figure 8: Simulator $!\mathcal{S}^{\text{js}}_{\mathcal{F}_{\text{ltsenc}}}$ for joint state realization of $!\underline{\mathcal{F}_{\text{ltsenc}}}$.

adversary is asked to send encryption and decryption algorithms as well as a public key. The public key is given to the decryptor, who can distribute it.

We formulate $\mathcal{F}_{\text{pke}}$ with a modeling of corruption which is similar to the one we introduced for $\mathcal{F}_{\text{ltsenc}}$. See Figure 9 for a definition of $\mathcal{F}_{\text{pke}}$. The proofs of the theorems in [32] can be easily adapted to the definition of $\mathcal{F}_{\text{pke}}$ we consider in this paper. For completeness we restate the results from [32], see below.

The functionalities $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{senc}}^{\text{auth}}$ will use the multi-party version $!\underline{\mathcal{F}_{\text{pke}}}$ of $\mathcal{F}_{\text{pke}}$, which provides public key encryption functionalities for an unbounded number of parties.

Realizing the Functionality. Every asymmetric encryption scheme Σ induces a protocol (system) $\mathcal{P}_{\text{pke}}(\Sigma, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$ that has the same interfaces as $\mathcal{F}_{\text{pke}}$ and provides encryption and decryption abilities for the parties. Furthermore, the adversary is able to statically corrupt the keys generated within $\mathcal{P}_{\text{pke}}$.

Theorem 7. *Let L be a leakage algorithm, which leaks exactly the length of a message, and let $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ be an asymmetric encryption scheme with domain $\text{dom}(L)$. Then, for all sufficiently large polynomials q (i.e., where all algorithms in Σ are bounded by q) it holds*

$$\Sigma \text{ is IND-CCA secure iff } \mathcal{P}_{\text{pke}}(\Sigma, \hat{\mathcal{T}}) \leq^{SS} \mathcal{F}_{\text{pke}}(q, L, \mathcal{T})$$

for all disjoint (sets of) tape names $\mathcal{T}_{\text{users}}, T_{\text{adv}}, \hat{T}_{\text{adv}}$ where $\mathcal{T} = (\mathcal{T}_{\text{users}}, T_{\text{adv}})$ and $\hat{\mathcal{T}} = (\mathcal{T}_{\text{users}}, \hat{T}_{\text{adv}})$.

The directions from left to right hold for any length preserving leakage algorithm L.

Joint State Realization. The joint state realization of [32] can be adapted in an obvious way to obtain a joint state realization $\mathcal{P}_{\text{pke}}^{\text{js}}$ for the formulation of $\mathcal{F}_{\text{pke}}$ in this paper. As in [32] and for long-term symmetric key encryption, $\mathcal{P}_{\text{pke}}^{\text{js}}$ is parameterized by a polynomial and a set of tape names. The polynomial is used to bound the length of the session identifiers. We can prove the following joint state theorem.

Theorem 8. *For all polynomials q, q' and disjoint (sets of) tape names $\mathcal{T}_{\text{users}}, T_{\text{adv}}$ there exists a polynomial q'' such that for every leakage algorithm L we have*

$$!\mathcal{P}_{\text{pke}}^{\text{js}}(q, \mathcal{T}_{\text{users}}) \mid !\mathcal{F}_{\text{pke}}(q', L', \hat{\mathcal{T}}) \leq^{SS} !\underline{\mathcal{F}_{\text{pke}}(q'', L, \mathcal{T})}$$

where $L'(1^\eta, (sid, m)) = (sid, L(1^\eta, m))$ for all $sid \in \text{SIDS}$ and $m \in \text{dom}(L)_\eta$, i.e., $\text{dom}(L') = (\{(sid, m) \mid sid \in \text{SIDS}, m \in \text{dom}(L)_\eta\})_{\eta \in \mathbb{N}}$, $\mathcal{T} = (\mathcal{T}_{\text{users}}, T_{\text{adv}})$, and $\hat{\mathcal{T}} = (\hat{\mathcal{T}}_{\text{users}}, \hat{T}_{\text{adv}})$ ($\hat{\mathcal{T}}_{\text{users}} = \{\hat{T} \mid T \in \mathcal{T}_{\text{users}}\}$).

5 The Symmetric Key Encryption Functionality

The main purpose of this functionality is for parties to be able to generate short-term keys and to provide (ideal) encryption and decryption under these keys as well as under public keys and long-term symmetric key, as described in Section 4. Short-term keys may themselves be part of the encrypted messages. 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. The functionality can be used to encrypt arbitrary messages (bit strings). We do not put a bound on the length and number of these messages. These messages may contain pointers of 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 Key is a tag and k the actual key. Again, the tag 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 emphasize that, apart from keys and pointers, we do not interpret messages; messages may be arbitrary bit strings, representing payload data, party names, nonces, ciphertexts (including ciphertexts previously generated by the functionality), digital signatures, non-interactive zero-knowledge proofs etc. Instead of using tags for pointers and keys we could parametrize the functionalities with any encoding and decoding function, 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,

$\mathcal{F}_{\text{pke}}(q, L, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$

Tapes: in: T^{in} for all $T \in \mathcal{T}_{\text{users}}$ (enriching I/O tapes), $T^{\text{in}}_{\text{adv}}$ (network tape)
out: T^{out} for all $T \in \mathcal{T}_{\text{users}}$ (I/O tapes), $T^{\text{out}}_{\text{adv}}$ (network tape)

State: $\mathit{stateDec} \in \{\mathsf{init}, \mathsf{wait}\} \times \mathcal{T}_{\text{users}} \cup \{\mathsf{ok}\}$ (initially init)
$\mathit{stateEnc}$: PIDS $\dashrightarrow \{\mathsf{wait}\} \times \mathcal{T}_{\text{users}} \cup \{\mathsf{ok}\}$ (partially defined, initially domain $= \emptyset$)
$\mathit{enc}, \mathit{dec}, \mathit{pk} \in \{0,1\}^* \cup \{\bot\}$ (initially $\bot$)
$\mathit{decTable} \subseteq \{0,1\}^* \times \{0,1\}^*$ (initially $\emptyset$)
$\mathit{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ (initially false)

1. KEY GENERATION: Upon receiving $(\mathsf{Algorithms}, \mathit{enc}', \mathit{dec}', \mathit{pk}', \mathit{corrupt})$ from T_{adv} where $\mathit{enc} = \mathit{dec} = \mathit{pk} = \bot$, $|\mathit{enc}'|, |\mathit{dec}'|, |\mathit{pk}'| \leq q(\eta)$, and $\mathit{corrupt} \in \{\mathsf{false}, \mathsf{true}\}$: $\mathit{enc} := \mathit{enc}'$, $\mathit{dec} := \mathit{dec}'$, $\mathit{pk} := \mathit{pk}'$, $\mathit{corrupted} := \mathit{corrupt}$, send (Ack) to T_{adv}.
2. INITIALIZATION DECRYPTOR: Upon receiving (KeyGen) from $T \in \mathcal{T}_{\text{users}}$: $\mathit{stateDec} := (\mathsf{wait}, T)$ and send (KeyGen) to T_{adv}.
3. COMPLETE DECRYPTOR: Upon receiving $(\mathsf{Complete})$ from T_{adv} where $\mathit{pk} \neq \bot$ and $\mathit{stateDec} = (\mathsf{wait}, T)$: $\mathit{stateDec} := \mathsf{ok}$ and send $(\mathsf{PublicKey}, \mathit{pk})$ to T.
4. INITIALIZATION ENCRYPTOR: Upon receiving $(p, \mathsf{Initialize})$ from $T \in \mathcal{T}_{\text{users}}$: $\mathit{stateEnc}(p) := (\mathsf{wait}, T)$ and send $(p, \mathsf{Initialize})$ to T_{adv}.
5. COMPLETE ENCRYPTOR: Upon receiving $(p, \mathsf{Complete})$ from T_{adv} where $\mathit{pk} \neq \bot$ and $\mathit{stateEnc}(p) = (\mathsf{wait}, T)$: $\mathit{stateEnc}(p) := \mathsf{ok}$ and send $(p, \mathsf{Completed})$ to T.
6. ENCRYPTION: Upon receiving $(p, \mathsf{Enc}, \mathit{pk}', m)$ from $T \in \mathcal{T}_{\text{users}}$:
Send $(p, \mathsf{Ciphertext}, \bot)$ to T if $\mathit{stateEnc}(p) \neq \mathsf{ok}$ or $m \notin \mathrm{dom}(L)_\eta$, otherwise:

   if not *corrupted* **and** $\mathit{pk} = \mathit{pk}'$ **then**
      $\overline{m} \leftarrow L(1^\eta, m)$, $c \leftarrow \mathrm{sim}_{q(\eta+|\overline{m}|)}(\mathit{enc}, (\mathit{pk}, \overline{m}))$
      if $\overline{m} = \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, c)$ **then**
         $\mathit{decTable} := \mathit{decTable} \cup \{(m, c)\}$, **send** $(p, \mathsf{Ciphertext}, c)$ **to** T
      else
         send $(p, \mathsf{Ciphertext}, \bot)$ **to** T
   else
      $c \leftarrow \mathrm{sim}_{q(\eta+|m|)}(\mathit{enc}, (\mathit{pk}', m))$, **send** $(p, \mathsf{Ciphertext}, c)$ **to** T

7. DECRYPTION: Upon receiving (Dec, c) from $T \in \mathcal{T}_{\text{users}}$:
Send $(\mathsf{Plaintext}, \bot)$ to T if $\mathit{stateDec} \neq \mathsf{ok}$, otherwise:

   if not *corrupted* **then**
      $$m := \begin{cases} m' & \text{if } \exists! m' \colon (m', c) \in \mathit{decTable} \\ \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, c) & \text{if } \forall m' \colon (m', c) \notin \mathit{decTable} \\ \bot & \text{otherwise} \end{cases}$$
      send $(\mathsf{Plaintext}, m)$ to T
   else $\{\mathit{corrupted}\}$
      $m := \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, c)$, **send** $(\mathsf{Plaintext}, m)$ to T

8. CORRUPTION REQUEST: Upon receiving $(\mathsf{Corrupted?})$ from $T \in \mathcal{T}_{\text{users}}$ where $\mathit{pk} \neq \bot$:
Send $(\mathsf{CorruptionState}, \mathit{corrupted})$ to T.

Figure 9: Functionality $\mathcal{F}_{\text{pke}}$ for asymmetric encryption.

in applications in which the positions of pointers/keys in a message are known, pointers and keys could be extracted without relying on tags.

In Section 5.2, we show that the functionality can be realized by every authenticated encryption scheme. We also present, in Section 6, a relaxed version $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ of $\mathcal{F}_{\text{senc}}$ which does not capture the properties of an authenticated encryption scheme and is realizable by every IND-CCA secure encryption scheme. To explicitly distinguish between the two variants we refer to the functionality presented in this section by $\mathcal{F}_{\text{senc}}^{\text{auth}}$.

5.1 The Functionality

The ideal functionality $\mathcal{F}_{\text{senc}}$ handles the key generation, encryption, and decryption requests of multiple parties. It also provides an interface to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, for bootstrapping symmetric key encryption (see Section 4). Just as these two functionalities, $\mathcal{F}_{\text{senc}}$ is parametrized by a polynomial q, a leakage algorithm L and tape names $\mathcal{T}_{\text{users}}$ and T_{adv}. The polynomial is used to bound the algorithm and keys for encryption and decryption provided by the adversary, see discussion in Section 3.1. The tape names specify the interface, i.e., the input and output tapes, see also Figure 16 for an overview.

1. *User interface* (enriching I/O tapes): The input tapes T^{in} and output tapes T^{out} (for all $T \in \mathcal{T}_{\text{users}}$) are for users to generate short-term keys and to encrypt and decrypt messages with these short-term keys. Users may connect to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ via $\mathcal{F}_{\text{senc}}$ to set up long-term keys or public keys, respectively, and to encrypt and decrypt messages with these long-term keys or public keys, respectively. Furthermore, the environment might request whether keys (short-term, long-term or public keys) are corrupted.
2. *Interface to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$* (consuming I/O tapes): As mentioned above, $\mathcal{F}_{\text{senc}}$ connects to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, therefore, the tapes ${T^{\text{lt}}}^{\text{in}}$, ${T^{\text{lt}}}^{\text{out}}$, ${T^{\text{pke}}}^{\text{in}}$, and ${T^{\text{pke}}}^{\text{out}}$ (for all $T \in \mathcal{T}_{\text{users}}$) are used to connect to $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, respectively.

   Although these tapes are consuming $\mathcal{F}_{\text{senc}}$ still can forward all messages received on these tapes because we only consider $\mathcal{F}_{\text{senc}}$ running together with $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ and they only receive enriching input via $\mathcal{F}_{\text{senc}}$.
3. *Adversarial interface* (consuming network tapes): The adversary connects to $T_{\text{adv}}^{\text{in}}$ and $T_{\text{adv}}^{\text{out}}$.

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, and a set of corrupted keys $\mathcal{K}_{\text{corrupt}} \subseteq \mathcal{K}_{\text{known}}$. The set $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ represents the unknown keys. A partial function *key* yields the key $key(p, ptr) \in \mathcal{K}$ pointer ptr points to for party p. Pointers are natural numbers, the first pointer a party receives is 0, the second 1 and so on. The information about the next pointer of party p is stored in $nextpointer(p)$. For ideal encryption and decryption, a table $decTable(k)$ is kept for every key $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ which records pairs (m, c) of ciphertexts c and their corresponding plaintext m. For generation of short-term keys, we need to store information which is done in variable $stateKeyGen(p)$ for party p, see below. In *enc* and *dec* the algorithms for encryption and decryption with short-term keys are stored, see below. To keep track of known keys, $\mathcal{F}_{\text{senc}}$ maintains sets of ciphertexts $\mathcal{C}_{\text{lt}}(pids)$ and $\mathcal{C}_{\text{pke}}(p)$ that where encrypted ideally by $\mathcal{F}_{\text{ltsenc}}$ for parties *pids* or $\mathcal{F}_{\text{pke}}$ for party p, respectively, see below. To tell, whether a ciphertext was encrypted ideally by $\mathcal{F}_{\text{pke}}$ for party p, it is important to know whether the right public key was used, therefore, $\mathcal{F}_{\text{senc}}$ stores the public key of party p in $pk(p)$. In mode CheckAddress all messages are accepted.

We now explain how $\mathcal{F}_{\text{senc}}$ works in more detail, see Figure 10, 11 and 12 for full details:

Obtaining Encryption and Decryption Algorithms. Before encryption and decryption can be performed, $\mathcal{F}_{\text{senc}}$ expect to receive an encryption *enc* and decryption *dec* algorithm from the adversary (the simulator). 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.

$\mathcal{F}_{\text{senc}}^{\text{auth}}(q, L, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$

Tapes: in: T^{in} (enriching), ${T^{\text{lt}}}^{\text{out}}$, ${T^{\text{pke}}}^{\text{out}}$ for all $T \in \mathcal{T}_{\text{users}}$ (I/O tapes), $T_{\text{adv}}^{\text{in}}$ (network tape)
out: T^{out}, ${T^{\text{lt}}}^{\text{in}}$, ${T^{\text{pke}}}^{\text{in}}$ for all $T \in \mathcal{T}_{\text{users}}$ (I/O tapes), $T_{\text{adv}}^{\text{out}}$ (network tape)

State: $enc, dec \in \{0,1\}^* \cup \{\bot\}$ (initially $\bot$)
$nextpointer\colon \textsc{Pids} \to \mathbb{N}$ (initially $p \mapsto 0$ for all p)
$stateKeyGen\colon \textsc{Pids} \dashrightarrow \{\mathsf{wait}\} \times \mathcal{T}_{\text{users}}$ (partial function, initially domain $= \emptyset$)
$decTable\colon \{0,1\}^* \to 2^{\{0,1\}^* \times \{0,1\}^*}$ (initially $k \mapsto \emptyset$ for all k)
$key\colon \textsc{Pids} \times \mathbb{N} \dashrightarrow \{0,1\}^*$ (partial function, initially domain $= \emptyset$)
$\mathcal{K}, \mathcal{K}_{\text{known}}, \mathcal{K}_{\text{corr}} \subseteq \{0,1\}^*$ (initially $\emptyset$)
$\mathcal{C}_{\text{lt}}\colon 2^{\textsc{Pids}} \to 2^{\{0,1\}^*}$ (initially $pids \mapsto \emptyset$ for all $pids$)
$\mathcal{C}_{\text{pke}}\colon \textsc{Pids} \to 2^{\{0,1\}^*}$ (initially $p \mapsto \emptyset$ for all p)
$pk\colon \textsc{Pids} \dashrightarrow \{0,1\}^*$ (partial function, initially domain $= \emptyset$)

1. Algorithms: Upon receiving $(\mathsf{Algorithms}, e, d)$ from T_{adv} where $|e|, |d| \le q(\eta)$ and $enc = dec = \bot$:
   $enc := e$, $dec := d$, **send** (Ack) **to** T_{adv}
2. Key generation: Upon receiving (p, KeyGen) from $T \in \mathcal{T}_{\text{users}}$:
   $stateKeyGen(p) := (\mathsf{wait}, T)$, **send** (p, KeyGen) **to** T_{adv}
3. Key generation: Upon receiving $(p, \mathsf{KeyGen}, corrupt, k)$ from T_{adv} where $stateKeyGen(p) = (\mathsf{wait}, T)$, for some T, $|k| \le q(\eta)$, $corrupt \in \{\mathsf{false}, \mathsf{true}\}$, and ($k \notin \mathcal{K}$ or ($corrupt = \mathsf{true}$ and $k \in \mathcal{K}_{\text{known}}$)):
   $stateKeyGen(p) := \bot$, $\mathcal{K} := \mathcal{K} \cup \{k\}$
   if $corrupt$ **then** $\mathcal{K}_{\text{corr}} := \mathcal{K}_{\text{corr}} \cup \{k\}$, $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k\}$
   if not $\exists! ptr\colon key(p, ptr) = k$ **then** {this is always the case if $corrupt = \mathsf{false}$}
   $ptr := nextpointer(p)\mathsf{++}$, $key(p, ptr) := k$
   send $(p, \mathsf{KeyGen}, ptr)$ **to** T
4. Store: Upon receiving (p, Store, k) from $T \in \mathcal{T}_{\text{users}}$ do: If $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ then $ptr := \bot$, otherwise, if exists ptr' s.t. $key(p, ptr') = k$ then $ptr := ptr'$, otherwise, $ptr := nextpointer(p)\mathsf{++}$, $key(p, ptr) := k$, and $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k\}$. Finally, send (p, Store, ptr) to T.
5. Reveal: Upon receiving $(p, \mathsf{Reveal}, ptr)$ from $T \in \mathcal{T}_{\text{users}}$ do: If $k := key(p, ptr) = \bot$ then send $(p, \mathsf{Reveal}, \bot)$ to T. Otherwise, $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k\}$ and send (p, Reveal, k) to T.
6. Corruption request: **recv** $(p, \mathsf{Corrupted?}, ptr)$ **from** $T \in \mathcal{T}_{\text{users}}$, $(p, ptr) \in \text{dom}(key)$:
   send $(\mathsf{CorruptionState}, \text{“}key(p, ptr) \in \mathcal{K}_{\text{corr}}\text{”})$ **to** T
7. Forwarding: Forward all messages for $\mathcal{F}_{\text{ltsenc}}$ (resp., $\mathcal{F}_{\text{pke}}$) except for encryption and decryption requests between T and T^{lt} (resp., T^{pke}) for all $T \in \mathcal{T}_{\text{users}}$. If received $(p, \mathsf{PublicKey}, k)$ from T^{pke} then $pk(p) := k$.

Figure 10: Functionality $\mathcal{F}_{\text{senc}}^{\text{auth}}$ for symmetric encryption with short-term keys.

$\mathcal{F}_{\text{senc}}^{\text{auth}}(q, L, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$ (continued)

8. ENCRYPTION, SHORT-TERM KEY: **recv** $(p, \mathsf{Enc}, ptr, m)$ **from** $T \in \mathcal{T}_{\text{users}}$:

   TRANSLATE POINTERS IN m TO KEYS, OBTAIN m': If m is an invalid user plaintext for p (i.e., contains invalid pointer) then $m' := \bot$, otherwise, obtain m' from m by replacing $(\mathsf{Key}, ptr) \in m$ by $(\mathsf{Key}, key(p, ptr))$.

   ENCRYPT m', OBTAIN CIPHERTEXT c:

   if ptr is invalid **or** $m' = \bot$ **or** $m' \notin \text{dom}(L)_\eta$ **or** $enc = dec = \bot$ **then**
   $c := \bot$
   else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
   $\overline{m} \leftarrow L(1^\eta, m')$, $c \leftarrow \text{sim}_{q(\eta+|\overline{m}|)}(enc, (k, \overline{m}))$
   if $\text{sim-det}_{q(\eta+|c|)}(dec, (k, c)) = \overline{m}$ **then**
   $decTable(k) := decTable(k) \cup \{(m', c)\}$
   else
   $c := \bot$
   else {key k is known}
   $c \leftarrow \text{sim}_{q(\eta+|m'|)}(enc, (k, m'))$

   UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if $c \neq \bot$ and encryption was not ideal, i.e., if $key(p, ptr) \in \mathcal{K}_{\text{known}}$.

   RETURN CIPHERTEXT c: Send $(p, \mathsf{Ciphertext}, c)$ to T.

9. ENCRYPTION, LONG-TERM KEY: **recv** $(pids, p, \mathsf{Enc}, m)$ **from** $T \in \mathcal{T}_{\text{users}}$ and $p \in pids$:

   TRANSLATE POINTERS IN m TO KEYS, OBTAIN m': As encryption with short-term keys.

   ENCRYPT m', OBTAIN CIPHERTEXT c: Send $(pids, p, \mathsf{Enc}, m')$ to T^{lt} and wait for receiving $(pids, p, \mathsf{Ciphertext}, c)$ from T^{lt} (i.e., request $\mathcal{F}_{\text{ltsenc}}[pids]$)

   UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if $c \neq \bot$ and encryption was not ideal, i.e., if $\mathcal{F}_{\text{ltsenc}}[pids]$ is corrupted (determined by requesting $\mathcal{F}_{\text{ltsenc}}[pids]$).

   STORE CIPHERTEXT c: $\mathcal{C}_{\text{lt}}(pids) := \mathcal{C}_{\text{lt}}(pids) \cup \{c\}$ if $c \neq \bot$ and encryption was ideal.

   RETURN CIPHERTEXT c: Send $(pids, p, \mathsf{Ciphertext}, c)$ to T.

10. ENCRYPTION, PUBLIC KEY: **recv** $(p', p, \mathsf{Enc}, k, m)$ **from** $T \in \mathcal{T}_{\text{users}}$:

    TRANSLATE POINTERS IN m TO KEYS, OBTAIN m': As encryption with short-term keys.

    ENCRYPT m', OBTAIN CIPHERTEXT c: Send $(p', p, \mathsf{Enc}, k, m')$ to T^{pke} and wait for receiving $(p', p, \mathsf{Ciphertext}, c)$ from T^{pke} (i.e., request $\underline{\mathcal{F}_{\text{pke}}}[p']$).

    UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if $c \neq \bot$ and encryption was not ideal, i.e., if $\underline{\mathcal{F}_{\text{pke}}}[p']$ is corrupted (determined by requesting $\underline{\mathcal{F}_{\text{pke}}}[p']$) or $k \neq pk(p')$ (wrong public key).

    STORE CIPHERTEXT c: $\mathcal{C}_{\text{pke}}(p') := \mathcal{C}_{\text{pke}}(p') \cup \{c\}$ if $c \neq \bot$ and encryption was ideal.

    RETURN CIPHERTEXT c: Send $(p', p, \mathsf{Ciphertext}, c)$ to T.

Figure 11: Functionality $\mathcal{F}_{\text{senc}}^{\text{auth}}$ for symmetric encryption with short-term keys. (continued)

$\mathcal{F}_{\text{senc}}^{\text{auth}}(q, L, (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$ (continued)

11. DECRYPTION, SHORT-TERM KEY: **recv** $(p, \mathsf{Dec}, ptr, m)$ **from** $T \in \mathcal{T}_{\text{users}}$:

DECRYPT c, OBTAIN PLAINTEXT m':

if ptr is invalid (i.e., $(p, ptr) \notin \text{dom}(key)$) **or** $enc = dec = \bot$ **then**
$m' := \bot$
else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
if $\exists! m'' \in \{0,1\}^*$: $(m'', c) \in decTable(k)$ **then** {exists unique m''}
$m' := m''$
else {decryption of c is ambiguous or c does not exist in $decTable(k)$}
$m' := \bot$
else {key k is known}
$m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$

PREVENT KEY GUESSING: If decryption was not ideal (i.e., $key(p, ptr) \in \mathcal{K}_{\text{known}}$) and m' contains (Key, k') with $k' \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ then $m' := \bot$.

UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if decryption was not ideal.

TRANSLATE KEYS IN m' TO POINTERS, OBTAIN m: Obtain m from m' by replacing $(\mathsf{Key}, k') \in m$ by (Key, ptr) where $k' = key(p, ptr)$ (if such ptr exists it is unique), otherwise, create new pointer $ptr := nextpointer(p)\mathsf{++}$, $key(p, ptr) := k'$

RETURN PLAINTEXT m: Send $(p, \mathsf{Plaintext}, m)$ to T.

12. DECRYPTION, LONG-TERM KEY: **recv** $(pids, p, \mathsf{Dec}, c)$ **from** $T \in \mathcal{T}_{\text{users}}$ and $p \in pids$:

DECRYPT c, OBTAIN PLAINTEXT m': Send $(pids, p, \mathsf{Dec}, c)$ to T^{lt} and wait for receiving $(pids, p, \mathsf{Plaintext}, m')$ from T^{lt} (i.e., request $\mathcal{F}_{\text{ltsenc}}[pids]$)

PREVENT KEY GUESSING: As decryption with short-term keys except that here "decryption was not ideal" means that $\mathcal{F}_{\text{ltsenc}}[pids]$ is corrupted or $c \notin \mathcal{C}_{\text{lt}}(pids)$.

UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if decryption was not ideal.

TRANSLATE KEYS IN m' TO POINTERS, OBTAIN m: As decryption with short-term keys.

RETURN PLAINTEXT m: Send $(pids, p, \mathsf{Plaintext}, m)$ to T.

13. DECRYPTION, PUBLIC KEY: **recv** (p, Dec, c) **from** $T \in \mathcal{T}_{\text{users}}$:

DECRYPT c, OBTAIN PLAINTEXT m': Send (p, Dec, c) to T^{pke} and wait for receiving $(p, \mathsf{Plaintext}, m')$ from T^{pke} (i.e., request $\underline{\mathcal{F}_{\text{pke}}}[p]$)

PREVENT KEY GUESSING: As decryption with short-term keys except that here "decryption was not ideal" means that $\underline{\mathcal{F}_{\text{pke}}}[p]$ is corrupted or $c \notin \mathcal{C}_{\text{pke}}(p)$.

UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if decryption was not ideal.

TRANSLATE KEYS IN m' TO POINTERS, OBTAIN m: As decryption with short-term keys.

RETURN PLAINTEXT m: Send $(p, \mathsf{Plaintext}, m)$ to T.

Figure 12: Functionality $\mathcal{F}_{\text{senc}}^{\text{auth}}$ for symmetric encryption with short-term keys. (continued)

(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 $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ (no key guessing). We emphasize that the difference between $\mathcal{K}_{\text{known}}$ and $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ is not whether or not an adversary knows the value of a key; the adversary knows this value anyway, since she provides these values in the ideal world. The point is that if $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$, 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 $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ 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. Note that if k is not marked corrupted then a new pointer will always be created because k is fresh in this case.

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. For public/private key generation, if the public key is returned by $\mathcal{F}_{\text{pke}}$ then it is stored in $pk(p)$ for party p.

Store and Reveal. A party p can ask $\mathcal{F}_{\text{senc}}$ to store some bit string k under some pointer. If k belongs to $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ then $\mathcal{F}_{\text{senc}}$ will return an error message (no key guessing). Otherwise, it creates a pointer to this key for party p, if there does not exist such a pointer already, and this pointer is given to p. The key k is added to $\mathcal{K}_{\text{known}}$.

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

Encryption Requests. We first consider encryption with short-term keys. Such a request contains a message m to be encrypted, the name of the party p who wants to encrypt m, and a pointer ptr 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, we call such pointers *valid* (for party p). Also, this is checked for all pointers (Key, ptr') in m. We call such an m a *valid user plaintext* (for party p) If these checks are successful, these pointers are replaced by their corresponding keys (Key, k'), resulting in a message m'. Then, if $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$, the *leakage* $L(1^\eta, m')$ of m' is encrypted under k using the encryption algorithm provided by the adversary, which is simulated for a polynomial number of steps. 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 long-term symmetric key encryption and public key encryption are handled similarly. The main difference is that the encryption of m' is handled by (an instance of) $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, respectively. If one of these functionalities is corrupted (this can be checked by simply asking the functionalities about their corruption status), the keys stored in m' are marked as known in $\mathcal{F}_{\text{senc}}$. For public key encryption, these keys are also marked as known if the wrong public key has been used, this can be checked because upon key generation the public key for party p was recorded in $pk(p)$. If the encryption was ideal, i.e., if the keys in m' where not marked known, then the ciphertext produced by $\mathcal{F}_{\text{ltsenc}}$ for parties $pids$ or $\mathcal{F}_{\text{pke}}$ for party p, respectively, is stored in $\mathcal{C}_{\text{lt}}(pids)$ or $\mathcal{C}_{\text{pke}}(p)$, respectively. This information is needed for decryption, see below.

Decryption Requests. We first consider encryption with short-term keys. Such a request contains a ciphertext c, the name of the party p who wants to decrypt c and a pointer ptr to the key with which p wants to decrypt m. Similar to the case of encryption, it is first checked whether ptr is valid for p (i.e., $k = key(p, ptr)$ is defined). If $k \in \mathcal{K}_{\text{known}}$, c is decrypted under k with the decryption algorithm provided by the adversary. If the resulting plaintext m' contains a key (Key, k') with $k' \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$, 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. If $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$, 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 are none or more than one m' with $(m', c) \in decTable(k)$, an error is returned.

Decryption requests for long-term symmetric key encryption and public key encryption are handled similarly. The main difference is that the decryption of c is handled by (an instance of) $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, respectively. If the decryption is not ideally, i.e., if one of these functionalities is corrupted (this can be checked by simply asking the functionalities about their corruption status) or if the ciphertext was not created ideally by honest parties (this can be checked because we stored honestly created ciphertexts in $\mathcal{C}_{\text{lt}}(pids)$ and $\mathcal{C}_{\text{pke}}(p)$), then, as for short-term key decryption, an error message is returned if the resulting plaintext m' contains a key (Key, k') with $k' \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ modeling that this should not happen (no key guessing).

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 $\mathcal{K}_{\text{corrupt}}$. Similar questions for long-term symmetric keys and public/private keys are forwarded by $\mathcal{F}_{\text{senc}}$ to (instances of) $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$.

This concludes the description of $\mathcal{F}_{\text{senc}}$. As explained for encryption requests, if a message m is encrypted under a known key (or by a corrupted $\mathcal{F}_{\text{ltsenc}}$ or $\mathcal{F}_{\text{pke}}$), then all keys in m are marked known in $\mathcal{F}_{\text{senc}}$. Yet, if in an application the ciphertext c for m is encrypted again under an *unknown* key and c is *always* encrypted under an unknown key, the keys in m might not be revealed. However, at some point in a cryptographic protocol, say, a ciphertext will typically be sent without being encrypted itself. So, $\mathcal{F}_{\text{senc}}$ seems to suffice in most applications. We are, for example, not aware of any authentication or key exchange protocol where the above situation occurs (see, e.g., [13] for a collection of such protocols) and even if it occurred, it might not be relevant for the kind of security properties one wishes to prove.

As for $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$, if the functionality is used with a leakage that has high entropy, then it guarantees that unknown ciphertext cannot be guessed. Let us explain: Assume that, e.g., due to nested encryption, a ciphertext c was generated by $\mathcal{F}_{\text{senc}}$ and that c is not known to the adversary because it was never output to the adversary. If the leakage has high entropy, the following is easy to see: The adversary has only negligible guessing probability for all ciphertexts that are stored in *decTable* in $\mathcal{F}_{\text{senc}}$ and which are formally unknown to the adversary. The proof idea is to exploit that the ciphertext has to contain as much information as $L(1^\eta, m)$, because of the decryption test during encryption. Since the leakage has high entropy, $L(1^\eta, m)$ is sufficiently random and can be guessed only with negligible probability.

Useful Invariants for Reasoning with the Functionality. Encryption with unknown and uncorrupted keys is ideal:

Lemma 2. *Let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}})$. In a run of $\mathcal{E} \mid \mathcal{F}_{\text{senc}}(L) \mid !\mathcal{F}_{\text{ltsenc}}(L) \mid !\underline{\mathcal{F}_{\text{pke}}(L)}$, every ciphertext returned by $\mathcal{F}_{\text{senc}}$ upon corruption of some plaintext m with some uncorrupted and unknown key (i.e., a key in $\mathcal{K} \setminus \mathcal{K}_{\text{known}}$ or an uncorrupted long-term key or public key) depends only on $L(m)$.*

Proof. Trivially this is an immediate consequence of the definition of $\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$. □

The following lemma states that if a key is generated honestly and never encrypted by a corrupted or known key then the key is always unknown. This, together with Lemma 2, is useful for reasoning about protocols, see Section 7.

Lemma 3. *Let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}})$ and ρ be a run of $\mathcal{E} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a)$ for some η, a. Let $k \in \{0,1\}^*$ be a key such that:*

1. *k is* generated honestly*, i.e., there is a configuration C in ρ where $\mathcal{E}$ sent $(p, \mathsf{KeyGen}, \mathsf{false}, k)$ (false indicates that this key is not corrupted) to $\mathcal{F}_{\text{senc}}$ and $k \notin \mathcal{K}$ in the configuration of $\mathcal{F}_{\text{senc}}$ in C,*
2. *k is* never encrypted by a corrupted or known key*, i.e., in no configuration C in ρ, $\mathcal{E}$ sent $(p, \mathsf{Enc}, ptr, m)$, $(pids, p, \mathsf{Enc}, m)$, or $(p', p, \mathsf{Enc}, pk, m)$ where $(\mathsf{Key}, ptr') \in m$, $k = key(p, ptr')$ and $key(p, ptr) \in \mathcal{K}_{\text{known}}$, $\mathcal{F}_{\text{ltsenc}}[pids].corrupted = \mathsf{true}$, or $\mathcal{F}_{\text{pke}}[p'].corrupted = \mathsf{true}$ or pk is the wrong public key (i.e., $pk \neq pk(p')$), respectively, and*
3. *k is* never revealed*, i.e., in no configuration C in ρ, $\mathcal{E}$ sent $(p, \mathsf{Reveal}, ptr)$ where $key(p, ptr) = k$.*

Then, the key k is always unknown, i.e., $k \notin \mathcal{K}_{\text{known}}$ for any configuration in ρ.

Proof. Because k is generated honestly upon generation we have $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$. After that, it can only be marked known upon encryption with a corrupted or known key or upon a reveal command which is excluded by assumption. Upon decryption k cannot be marked as known because of the "PREVENT KEY GUESSING" rule. Hence, k will always be unknown. □

Handling of Uninterpreted Messages. The functionality $\mathcal{F}_{\text{senc}}$ always interprets the plaintext and replaces pointers by keys. Typically this is not a big restriction, however, sometimes one might want to encrypt a bit string uninterpreted, e.g. if one wants to encrypt a Nonce which by accident happens to be of shape (Key, x) for some bit string x.

Here, we explain how $\mathcal{F}_{\text{senc}}$ can be used to encrypt messages without interpreting them by using the commands store and reveal.

If we want to encrypt a message m without interpreting it, instead of sending m directly to $\mathcal{F}_{\text{senc}}$ we first parses m for occurrences of (Key, x) for any bit string x. For each such x we store x in $\mathcal{F}_{\text{senc}}$ (by sending the message (p, Store, x)) obtaining a pointer ptr and replace (Key, x) by (Key, ptr) in m. This guarantees that exactly the plaintext intended to be encrypted gets encrypted because $\mathcal{F}_{\text{senc}}$ is undoing this replacement before encryption.

Similarly, if we want to decrypt a ciphertext c and obtain the uninterpreted plaintext, we decrypt c obtaining an interpreted plaintext m. Then, we retrieve the corresponding bit strings x (by sending the reveal command $(p, \mathsf{Reveal}, ptr)$ to $\mathcal{F}_{\text{senc}}$) for every $(\mathsf{Key}, ptr) \in m$ and replace (Key, ptr) by (Key, x) in m.

5.2 Realizing the Functionality

In this section, we show that an authenticated encryption scheme (IND-CPA and INT-CTXT secure) together with $!\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ realizes $\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$.

5.2.1 Protocol for Symmetric Key Encryption

A symmetric encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ induces a realization $\mathcal{P}_{\text{senc}}(\Sigma) = \mathcal{P}_{\text{senc}}(\Sigma, \mathcal{T})$ of $\mathcal{F}_{\text{senc}}$ in the obvious way: In $\mathcal{P}_{\text{senc}}(\Sigma, \mathcal{T})$, i) the tapes, are defined as for $\mathcal{F}_{\text{senc}}$ by the parameter $\mathcal{T}$, ii) ideal encryption and decryption is replaced by real encryption and decryption, iii) key guessing is not a prior prevented anymore, and iv) the bookkeeping for unknown and known keys is removed. Upon key generation, the adversary is asked whether she wants to corrupt the key, in which case she provides the key. Otherwise, the key is generated honestly within $\mathcal{P}_{\text{senc}}(\Sigma)$ using $\text{gen}(1^\eta)$. Keeping track of corrupted keys is necessary because the environment has the ability to ask which keys are corrupted. See Figure 13 for a precise definition of $\mathcal{P}_{\text{senc}}$.

5.2.2 Restricting the Environment

We would like to prove that $\mathcal{P}_{\text{senc}}(\Sigma)$ realizes $\mathcal{F}_{\text{senc}}$ for standard assumptions about the authenticated symmetric encryption scheme Σ, namely IND-CPA and INT-CTXT secure. 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 [11, 6]. 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., [4, 27]). As already mentioned in the introduction, similarly to [4], 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 pointed out in [4], that once a key has been used in a protocol to encrypt a message, this key is typically not encrypted anymore in the rest of the protocol. Let us call these protocols *standard*.

$\mathcal{P}_{\text{senc}}(\Sigma = (\text{gen}, \text{enc}, \text{dec}), (\mathcal{T}_{\text{users}}, T_{\text{adv}}))$

Tapes: As $\mathcal{F}_{\text{senc}}$ (see Figure 10).

State: *nextpointer*: PIDS $\to \mathbb{N}$ (initially $p \mapsto 0$ for all p)
stateKeyGen: PIDS $\dashrightarrow \{\mathsf{wait}\} \times \mathcal{T}_{\text{users}}$ (partial function, initially domain $= \emptyset$)
key: PIDS $\times \mathbb{N} \dashrightarrow \{0,1\}^*$ (partial function, initially domain $= \emptyset$)
$\mathcal{K}_{\text{corr}} \subseteq \{0,1\}^*$ (initially $\emptyset$)

1. KEY GENERATION: Upon receiving (p, KeyGen) from $T \in \mathcal{T}_{\text{users}}$:
   $\mathit{stateKeyGen}(p) := (\mathsf{wait}, T)$, **send** (p, KeyGen) **to** T_{adv}

2. KEY GENERATION: Upon receiving $(p, \mathsf{KeyGen}, \mathit{corrupt}, k')$ from T_{adv} where $\mathit{stateKeyGen}(p) = (\mathsf{wait}, T)$, for some T, $\mathit{corrupt} \in \{\mathsf{false}, \mathsf{true}\}$, and $|k'| \leq q(\eta)$ (where q is the polynomial associated with Σ that bounds the runtime of the algorithms):
   $\mathit{stateKeyGen}(p) := \bot$
   if *corrupt* **then**
   $k := k'$, $\mathcal{K}_{\text{corr}} := \mathcal{K}_{\text{corr}} \cup \{k\}$
   else
   $k \leftarrow \text{gen}(1^\eta)$
   if not $\exists! \mathit{ptr}\colon \mathit{key}(p, \mathit{ptr}) = k$ **then**
   $\mathit{ptr} := \mathit{nextpointer}(p)\texttt{++}$, $\mathit{key}(p, \mathit{ptr}) := k$
   send $(p, \mathsf{KeyGen}, \mathit{ptr})$ **to** T

3. STORE: Upon receiving (p, Store, k) from $T \in \mathcal{T}_{\text{users}}$ do: If exists ptr' s.t. $\mathit{key}(p, \mathit{ptr}') = k$ then $\mathit{ptr} := \mathit{ptr}'$, otherwise, $\mathit{ptr} := \mathit{nextpointer}(p)\texttt{++}$ and $\mathit{key}(p, \mathit{ptr}) := k$. Send $(p, \mathsf{Store}, \mathit{ptr})$ to T.

4. REVEAL: Upon receiving $(p, \mathsf{Reveal}, \mathit{ptr})$ from $T \in \mathcal{T}_{\text{users}}$ do: If $k := \mathit{key}(p, \mathit{ptr}) = \bot$ then send $(p, \mathsf{Reveal}, \bot)$ to T. Otherwise, send (p, Reveal, k) to T.

5. CORRUPTION REQUEST: Upon receiving $(p, \mathsf{Corrupted?}, \mathit{ptr})$ from $T \in \mathcal{T}_{\text{users}}$ where $(p, \mathit{ptr}) \in \text{dom}(\mathit{key})$ do: Send $(\mathsf{CorruptionState}, \text{“}\mathit{key}(p, \mathit{ptr}) \in \mathcal{K}_{\text{corr}}\text{”})$ to T.

6. FORWARDING: Forward all messages for $\mathcal{F}_{\text{ltsenc}}$ (resp., $\mathcal{F}_{\text{pke}}$) except for encryption and decryption requests between T and T^{lt} (resp., T^{pke}) for all $T \in \mathcal{T}_{\text{users}}$.

7. ENCRYPTION, SHORT-TERM KEY: Upon receiving $(p, \mathsf{Enc}, \mathit{ptr}, m)$ from $T \in \mathcal{T}_{\text{users}}$:

   TRANSLATE POINTERS IN m TO KEYS, OBTAIN m': As $\mathcal{F}_{\text{senc}}$ (8.), see Figure 11.
   ENCRYPT m', OBTAIN CIPHERTEXT c: $c := \bot$ if ptr is invalid (i.e., $\mathit{ptr} \notin \text{dom}(\mathit{key})$), $m' = \bot$ or $m' \notin \text{dom}(\Sigma)_\eta$, otherwise, $c \leftarrow \text{enc}(\mathit{key}(p, \mathit{ptr}), m')$.
   RETURN CIPHERTEXT c: Send $(p, \mathsf{Ciphertext}, c)$ to T.

8. ENCRYPTION, LONG-TERM AND PUBLIC KEY: As $\mathcal{F}_{\text{senc}}$ (9.) and (10), respectively, see Figure 11, but without “UPDATE $\mathcal{K}_{\text{known}}$” and “STORE CIPHERTEXT”.

9. DECRYPTION, SHORT-TERM KEY: Upon receiving $(p, \mathsf{Dec}, \mathit{ptr}, c)$ from $T \in \mathcal{T}_{\text{users}}$:

   DECRYPT c, OBTAIN PLAINTEXT m': $m' := \bot$ if ptr is invalid (i.e., $\mathit{ptr} \notin \text{dom}(\mathit{key})$), otherwise, $m' := \text{dec}(\mathit{key}(p, \mathit{ptr}), c)$.
   TRANSLATE KEYS IN m' TO POINTERS, OBTAIN m: As $\mathcal{F}_{\text{senc}}$ (11.), see Figure 12.
   RETURN PLAINTEXT m: Send $(p, \mathsf{Plaintext}, m)$ to T.

10. DECRYPTION, LONG-TERM AND PUBLIC KEY: As $\mathcal{F}_{\text{senc}}$ (12.) and (13), resp., see Figure 12, but without “PREVENT KEY GUESSING” and “UPDATE $\mathcal{K}_{\text{known}}$”.

Figure 13: Protocol $\mathcal{P}_{\text{senc}}$ for symmetric encryption with short-term keys.

This observation can be generalized to *used order respecting environments*, which we formulate based on $\mathcal{F}_{\text{senc}}$: In what follows, we say that an unknown key k, i.e., $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$, has been *used (for encryption)*, if $\mathcal{F}_{\text{senc}}$ has been instructed to encrypt a message using 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. Clearly, used order respecting environments produce key cycles (among unknown keys) with at most negligible probability.

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 in the run, i.e., is not added to $\mathcal{K}_{\text{known}}$. Assuming static corruption of keys, it is easy to see that for standard protocols, as introduced above, the commitment problem does not occur.

Instead of explicitly restricting the class of environments in our main theorem, we describe a functionality $\mathcal{F}^*$ that provides exactly the same I/O interface as $\mathcal{F}_{\text{senc}}$ (and hence, $\mathcal{P}_{\text{senc}}$), but before forwarding requests to $\mathcal{F}_{\text{senc}}$ checks whether the used ordering 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. See Figure 14 and 15 for a precise definition of $\mathcal{F}^*$.

Definition 9. An environmental system $\mathcal{E}$ that is environmentally connectible with $\mathcal{F}_{\text{senc}} \mid\ !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ is *non-committing and used order respecting* if for every simulator $\mathcal{S}$ that is adversarial connectible to $\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ and where $\mathcal{E}$ is connectible to $\mathcal{S} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ the probability that $\mathcal{F}^*$ sends (CommitProblem) or (UsedOrderViolated) in a run of $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}^* \mid\ \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ is negligible in the security parameter.

A protocol (system) $\mathcal{P}$ which is environmentally connectible with $\mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ is *non-committing and used order respecting* if $\mathcal{E} \mid \mathcal{P}$ is non-committing and used order respecting for all environments $\mathcal{E}$ that are environmentally connectible with $\mathcal{P} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$.

Note that these restrictions could also be expressed as a condition in the sense of [2].

Recall that a protocol (system) $\mathcal{P}$ that is environmentally connectible with $\mathcal{F}_{\text{senc}} \mid\ !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ is called *standard* if for every environment $\mathcal{E}$ that is connectible with $\mathcal{P} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid\ !\underline{\mathcal{F}_{\text{pke}}}$ in every run of $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ all short-term keys are never encrypted (by short-term, long-term or public keys) after they have been first used for encryption.

Lemma 4. *Every standard protocol (system) is non-committing and used order respecting.*

Proof. Because unknown and used short-term keys are never encrypted by any short-term, long-term or public key the commitment problem does not occur. Also, used order violations only occur if unknown and used keys are encrypted. □

5.2.3 Main Results

Before stating the main theorem, we fix the interfaces, i.e. the tapes, for the rest of this section as follows: Let $\mathcal{T}_{\text{users}}$ be a set of tape names and let T_{adv} be a tape name such that $T_{\text{adv}} \notin \mathcal{T}_{\text{users}}$. We define $\mathcal{T} = (\mathcal{T}_{\text{users}}, T_{\text{adv}})$, $\mathcal{T}^{\text{lt}} = (\mathcal{T}^{\text{lt}}_{\text{users}}, T^{\text{lt}}_{\text{adv}})$, $\mathcal{T}_{\text{pke}} = (\mathcal{T}^{\text{pke}}_{\text{users}}, T^{\text{pke}}_{\text{adv}})$, $\hat{\mathcal{T}} = (\mathcal{T}_{\text{users}}, \hat{T}_{\text{adv}})$, $\hat{\mathcal{T}}^{\text{lt}} = (\mathcal{T}^{\text{lt}}_{\text{users}}, \hat{T}^{\text{lt}}_{\text{adv}})$, $\hat{\mathcal{T}}^{\text{pke}} = (\mathcal{T}^{\text{pke}}_{\text{users}}, \hat{T}^{\text{pke}}_{\text{adv}})$ where $\mathcal{T}^{\text{lt}}_{\text{users}} = \{T^{\text{lt}} \mid T \in \mathcal{T}_{\text{users}}\}$ and $\mathcal{T}^{\text{pke}}_{\text{users}} = \{T^{\text{pke}} \mid T \in \mathcal{T}_{\text{users}}\}$. Furthermore, let $\mathcal{F}_{\text{senc}}(q, L) = \mathcal{F}_{\text{senc}}(q, L, \mathcal{T})$, $\mathcal{P}_{\text{senc}}(\Sigma) = \mathcal{P}_{\text{senc}}(\Sigma, \hat{\mathcal{T}})$, and $\mathcal{F}^* = \mathcal{F}^*(\mathcal{T}_{\text{users}})$. See Figure 16 for the connection between the systems.

Theorem 9. *Let L be a leakage algorithm which leaks exactly the length of a message and Σ be a symmetric encryption scheme with domain $\text{dom}(L)$. Then for all polynomials q_{st}, q_{lt} and q_{pke} where q_{st} is sufficiently large (such that Σ is bounded by q_{st}) it holds that Σ is IND-CPA and INT-CTXT secure if and only if*

$$\mathcal{F}^* \mid \mathcal{P}_{\text{senc}}(\Sigma) \mid !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \hat{\mathcal{T}}^{\text{lt}}) \mid !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \hat{\mathcal{T}}^{\text{pke}})} \leq^{SS} \mathcal{F}^* \mid \mathcal{F}_{\text{senc}}(q_{\text{st}}, L) \mid !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \mathcal{T}^{\text{lt}}) \mid !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \mathcal{T}^{\text{pke}})}\ .$$

This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}^{\text{auth}}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}^{\text{unauth}}_{\text{ltsenc}}$. Also, this holds if the leakage algorithms used by $\mathcal{F}_{\text{senc}}$, $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ differ.

$\mathcal{F}^*(\mathcal{T}_{\text{users}})$

Tapes: I/O in: $\hat{T}^{\text{in}}$ (enriching, from environment), T^{out} (from $\mathcal{F}_{\text{senc}}$) for all $T \in \mathcal{T}_{\text{users}}$,
I/O out: $\hat{T}^{\text{out}}$ (to environment), T^{in} (to $\mathcal{F}_{\text{senc}}$) for all $T \in \mathcal{T}_{\text{users}}$

State: $\mathit{nextkeyid}, \mathit{nextused} \in \mathbb{N}$ (initially 1)
$\mathit{keyid}\colon \text{PIDS} \times \mathbb{N} \dashrightarrow \mathbb{N}$ (partially defined, initially domain $= \emptyset$)
$\mathit{used}\colon \mathbb{N} \dashrightarrow \mathbb{N}$ (partially defined, initially domain $= \emptyset$)
$\mathcal{K}_{\text{known}} \subseteq \mathbb{N}$ (initially $\{0\}$)
$\mathit{decTable}_{\text{st}}\colon \mathbb{N} \to 2^{\{0,1\}^* \times \{0,1\}^*}$ (initially $i \mapsto \emptyset$ for every key ID i)
$\mathit{decTable}_{\text{lt}}\colon 2^{\text{PIDS}} \to 2^{\{0,1\}^* \times \{0,1\}^*}$ (initially $\mathit{pids} \mapsto \emptyset$ for every set of PIDs pids)
$\mathit{decTable}_{\text{pke}}\colon \text{PIDS} \to 2^{\{0,1\}^* \times \{0,1\}^*}$ (initially $p \mapsto \emptyset$ for every PID p)

1. KEY GENERATION: Upon receiving $(p, \mathsf{KeyGen}, \mathit{ptr})$ from $T \in \mathcal{T}_{\text{users}}$: If ptr of p is corrupted (determined by sending $(p, \mathsf{Corrupted?}, \mathit{ptr})$ to T) then $\mathit{keyid}(p, \mathit{ptr}) := 0$ if $(p, \mathit{ptr}) \notin \text{dom}(\mathit{keyid})$, otherwise, $\mathit{keyid}(p, \mathit{ptr}) := \mathit{nextkeyid}\text{++}$. Send $(p, \mathsf{KeyGen}, \mathit{ptr})$ to $\hat{T}$.
2. STORE: Upon receiving $(p, \mathsf{Store}, \mathit{ptr})$ from $T \in \mathcal{T}_{\text{users}}$: If $\mathit{ptr} \neq \bot$ and $(p, \mathit{ptr}) \notin \text{dom}(\mathit{keyid})$ then $\mathit{keyid}(p, \mathit{ptr}) := 0$. Send $(p, \mathsf{Store}, \mathit{ptr})$ to $\hat{T}$.
3. REVEAL: Upon receiving $(p, \mathsf{Reveal}, \mathit{ptr})$ from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$: If ptr is unknown and used (i.e., $\mathit{keyid}(p, \mathit{ptr}) \in \text{dom}(\mathit{used}) \setminus \mathcal{K}_{\text{known}}$) then send $(\mathsf{CommitProblem})$ to $\hat{T}$ and terminate. Otherwise, $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{\mathit{keyid}(p, \mathit{ptr})\}$ and send $(p, \mathsf{Reveal}, \mathit{ptr})$ to T.
4. ENCRYPTION, SHORT-TERM KEY: Upon receiving $(p, \mathsf{Enc}, \mathit{ptr}, m)$ from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$:

   FORWARD m, OBTAIN CIPHERTEXT c: Send $(p, \mathsf{Enc}, \mathit{ptr}, m)$ to T and wait for receiving $(p, \mathsf{Ciphertext}, c)$ from T.

   PREVENT USED ORDER VIOLATION: If $c \neq \bot$, *encryption was ideal* (i.e., $\mathit{keyid}(p, \mathit{ptr}) \notin \mathcal{K}_{\text{known}}$) and m *contains an unknown and used key* i' (i.e., m contains $(\mathsf{Key}, \mathit{ptr}')$ for some ptr' s.t. $i' = \mathit{keyid}(p, \mathit{ptr}') \in \text{dom}(\mathit{used}) \setminus \mathcal{K}_{\text{known}}$) and $(i = \mathit{keyid}(p, \mathit{ptr}) \notin \text{dom}(\mathit{used})$ or $\mathit{used}(i) \geq \mathit{used}(i'))$ then send $(\mathsf{UsedOrderViolated})$ to $\hat{T}$ and terminate (i.e., block all further messages).

   PREVENT COMMITMENT PROBLEM: If $c \neq \bot$, encryption was not ideal and m contains an unknown and used key then send $(\mathsf{CommitProblem})$ to $\hat{T}$ and terminate.

   UPDATE $\mathcal{K}_{\text{known}}$ AND $\mathit{decTable}$: If $c \neq \bot$ and encryption was not ideal then $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{\mathit{keyid}(p, \mathit{ptr}') \mid (\mathsf{Key}, \mathit{ptr}') \in m\}$.
   If $c \neq \bot$ and encryption was ideal then create the list keyids of key IDs in m (i.e., translate pointers in m to key IDs) and add (keyids, c) to $\mathit{decTable}_{\text{st}}(\mathit{keyid}(p, \mathit{ptr}))$. Furthermore, if $i = \mathit{keyid}(p, \mathit{ptr}) \notin \text{dom}(\mathit{used})$ then $\mathit{used}(i) := \mathit{nextused}\text{++}$.

   FORWARD CIPHERTEXT c: Send $(p, \mathsf{Ciphertext}, c)$ to $\hat{T}$.
5. ENCRYPTION, LONG-TERM AND PUBLIC KEY: Upon receiving $(\mathit{pids}, p, \mathsf{Enc}, m)$ or $(p', p, \mathsf{Enc}, k, m)$ from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$:

   FORWARD m, OBTAIN CIPHERTEXT c: Send $(\mathit{pids}, p, \mathsf{Enc}, m)$ (resp., $(p', p, \mathsf{Enc}, k, m)$) to T and wait for receiving $(\mathit{pids}, p, \mathsf{Ciphertext}, c)$ (resp., $(p', p, \mathsf{Ciphertext}, c)$) from T.

   PREVENT COMMITMENT PROBLEM: If $c \neq \bot$, *encryption was not ideal* (i.e., if $\mathcal{F}_{\text{ltsenc}}[\mathit{pids}]$ is corrupted or $\underline{\mathcal{F}_{\text{pke}}}[p']$ is corrupted or k is not the public key returned by $\underline{\mathcal{F}_{\text{pke}}}[p']$, respectively) and m contains an unknown and used key then send $(\mathsf{CommitProblem})$ to $\hat{T}$ and terminate.

   UPDATE $\mathcal{K}_{\text{known}}$ AND $\mathit{decTable}$: As encryption with short-term keys. Here, add (keyids, c) to $\mathit{decTable}_{\text{lt}}(\mathit{pids})$ or $\mathit{decTable}_{\text{pke}}(p')$, respectively.

   FORWARD CIPHERTEXT c: Send $(\mathit{pids}, p, \mathsf{Ciphertext}, c)$ or $(p', p, \mathsf{Ciphertext}, c)$, respectively, to $\hat{T}$.

Figure 14: Functionality $\mathcal{F}^*$ for restricting the environment.

$\mathcal{F}^*(\mathcal{T}_{\text{users}})$ (continued)

4. DECRYPTION, SHORT-TERM KEY: Upon receiving $(p, \mathsf{Dec}, ptr, c)$ from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$:

   FORWARD c, OBTAIN PLAINTEXT m: Send $(p, \mathsf{Dec}, ptr, c)$ to T and wait for receiving $(p, \mathsf{Plaintext}, m)$ from T.

   UPDATE $\mathcal{K}_{\text{known}}$: If $c \neq \bot$ and *decryption was not ideal* (i.e., $keyid(p, ptr) \in \mathcal{K}_{\text{known}}$) then $keyid(p, ptr') := 0$ for all $(\mathsf{Key}, ptr') \in m$ where $(p, ptr') \notin \text{dom}(keyid)$ and $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{keyid(p, ptr') \mid (\mathsf{Key}, ptr') \in m\}$.

   If $c \neq \bot$ and decryption was ideal then let *keyids* be s.t. $(keyids, c) \in decTable_{\text{st}}(keyid(p, ptr))$ (when running with $\mathcal{F}_{\text{senc}}^{\text{auth}}$ this always exist and is unique) and set $keyid(p, ptr) := i'$ for all $(\mathsf{Key}, ptr') \in m$ and corresponding key ID $i' \in keyids$. (When not running with $\mathcal{F}_{\text{senc}}^{\text{auth}}$ such *keyids* does not need to exist, in that case produce empty output and terminate.)

   FORWARD PLAINTEXT m: Send $(p, \mathsf{Plaintext}, m)$ to $\hat{T}$.

5. DECRYPTION, LONG-TERM AND PUBLIC KEY: Upon receiving $(pids, p, \mathsf{Dec}, c)$ or (p, Dec, c), respectively, from $\hat{T}$ with $T \in \mathcal{T}_{\text{users}}$:

   FORWARD c, OBTAIN PLAINTEXT m: Send $(pids, p, \mathsf{Dec}, c)$ (resp., (p, Dec, c)) to T and wait for receiving $(pids, p, \mathsf{Plaintext}, m)$ (resp., $(p, \mathsf{Plaintext}, m)$) from T.

   UPDATE $\mathcal{K}_{\text{known}}$: As decryption with short-term keys except that here *decryption was not ideal* means that $\mathcal{F}_{\text{ltsenc}}[pids]$ is corrupted or c does not exist in $decTable_{\text{lt}}(pids)$ or $\underline{\mathcal{F}_{\text{pke}}}[p]$ is corrupted or c does not exist in $decTable_{\text{pke}}(p)$, respectively)

   FORWARD PLAINTEXT m: Send $(pids, p, \mathsf{Plaintext}, m)$ or $(p, \mathsf{Plaintext}, m)$, respectively, to $\hat{T}$.

6. Forward all other messages.

Figure 15: Functionality $\mathcal{F}^*$ for restricting the environment. (continued)

Recall that the bootstrapping component $!\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ on the left-hand side of $\leq^{SS}$ can be replaced by its realization or even joint state realization, due to Theorems 5, 6, 7, and 8.

If for realizability we only consider a restricted class of environments, namely non-committing and used order respecting environments, then we also obtain realizability. We define that $\mathcal{P} \leq^{SS(*)} \mathcal{F}$ if there exists a simulator $\mathcal{S}$ such that for all non-committing and used order respecting environments $\mathcal{E}$ it holds that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$.

Corollary 2. *Let L be a leakage algorithm which leaks exactly the length of a message and Σ be a symmetric encryption scheme with domain* $\text{dom}(L)$*. Then for all polynomials q_{st}, q_{lt} and q_{pke} where q_{st} is sufficiently large (such that Σ is bounded by q_{st}) it holds that Σ is IND-CPA and INT-CTXT secure if and only if*

$$\mathcal{P}_{\text{senc}}(\Sigma) \mid !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \hat{\mathcal{T}}^{\text{lt}}) \mid !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \hat{\mathcal{T}}^{\text{pke}})} \leq^{SS(*)} \mathcal{F}_{\text{senc}}(q_{\text{st}}, L) \mid !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \mathcal{T}^{\text{lt}}) \mid !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \mathcal{T}^{\text{pke}})} \ .$$

This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$. Also, this holds if the leakage algorithms used by $\mathcal{F}_{\text{senc}}$, $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ differ.

Proof of Corollary 2. The direction from right to left follows immediately from the proof of Theorem 9 because the constructed environment is non-committing and used order respecting.

Next, we consider the direction from left to right: By Theorem 9, we find a simulator $\mathcal{S}$ for $\mathcal{F}^* \mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}} \leq^{SS} \mathcal{F}^* \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$. Let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}})$ such that $\mathcal{E}$ is non-committing and used order respecting. Let $\mathcal{E}'$ be obtained from $\mathcal{E}$ by renaming the I/O tapes connecting $\mathcal{E}$ and $\mathcal{F}_{\text{senc}}$ such that $\mathcal{E}'$ connects to $\mathcal{F}^*$. Then $\mathcal{E}' \mid \mathcal{F}^* \mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}} \equiv \mathcal{E}' \mid \mathcal{S} \mid \mathcal{F}^* \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$. Since $\mathcal{E}$ (and hence $\mathcal{E}'$) is non-committing and used order respecting we have that $\mathcal{E}' \mid \mathcal{S} \mid \mathcal{F}^* \mid \mathcal{F}_{\text{senc}} \mid !\underline{\mathcal{F}_{\text{ltsenc}}} \mid !\underline{\mathcal{F}_{\text{pke}}} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$. Consequently, the probability that $\mathcal{F}^*$ outputs $(\mathsf{CommitProblem})$ or $(\mathsf{UsedOrderViolated})$ in a run of $\mathcal{E}' \mid \mathcal{F}^* \mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ is

negligible too. Thus, $\mathcal{E}' \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \equiv \mathcal{E} \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$. By transitivity of $\equiv$, we conclude $\mathcal{E} \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$, as desired. □

We directly obtain the following corollary for non-committing and used order respecting protocols. As mentioned above, most protocols have this property and this can typically be easily checked by inspection of the protocol (see Section 7.1 for an example).

Corollary 3. *Let L be a leakage algorithm which leaks exactly the length of a message and Σ be an IND-CPA and INT-CTXT secure symmetric encryption scheme with domain* $\text{dom}(L)$*. Let $\mathcal{P}$ be a non-committing and used order respecting protocol. Then for all polynomials q_{st}, q_{lt} and q_{pke} where q_{st} is sufficiently large (such that Σ is bounded by q_{st}) it holds that*

$$\mathcal{P} \,|\, \mathcal{P}_{\text{senc}}(\Sigma) \,|\, !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \hat{\mathcal{T}}^{\text{lt}}) \,|\, !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \hat{\mathcal{T}}^{\text{pke}})} \\ \leq^{SS} \mathcal{P} \,|\, \mathcal{F}_{\text{senc}}(q_{\text{st}}, L) \,|\, !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \mathcal{T}^{\text{lt}}) \,|\, !\underline{\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \mathcal{T}^{\text{pke}})} \ .$$

This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$. Also, this holds if the leakage algorithms used by $\mathcal{F}_{\text{senc}}$, $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ differ.

5.2.4 Proof of the Main Results

To prove the Theorem 9 we first consider the direction from right to left. Given any IND-CCA, IND-CPA or INT-CTXT adversary A we can construct an environment $\mathcal{E}$ such that the advantage of A is bound by the advantage of $\mathcal{E}$ distinguishing between $\mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}}(\Sigma) \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$ and $\mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$ for any simulator $\mathcal{S}$. The environment $\mathcal{E}$ generates an uncorrupted short-term key for some party p and simulates A by using this key for encryption and decryption to simulate the oracles of A. To do so A has to encrypt messages uninterpreted (i.e. without replacing pointers by keys) which can be done as explained in Section 5.1 (handling of uninterpreted messages).

To prove the direction from left to right we first give the simulator and define hybrid systems $\mathcal{F}_{\text{senc}}^{(r)}$ for every $r \in \mathbb{N}$ where all keys that have used order less than r are treated as in $\mathcal{F}_{\text{senc}}$ while the others are treated as in $\mathcal{P}_{\text{senc}}$. Then, we relate the hybrid with $r = 0$ to the real system and the hybrid with $r = p(\eta)$ (where p is a polynomial that bounds the runtime of the environment) to the ideal system. Finally, we relate the r's hybrid to the $(r+1)$'s. To simplify the last step we introduce hybrids $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ which behave like $\mathcal{F}_{\text{senc}}^{(r)}$ but use Oracle (as defined in Section 3.3) to perform the encryption and decryption with the r-th key, i.e., the key of order r. By Lemma 1, to relate $\mathcal{F}_{\text{senc}}^{(r)}$ and $\mathcal{F}_{\text{senc}}^{(r+1)}$ we then only need to show that $\widehat{\mathcal{F}}_{\text{senc}}^{(r)} \,|\, \text{Oracle}(\mathsf{real})$ is related to $\mathcal{F}_{\text{senc}}^{(r)}$ and that $\widehat{\mathcal{F}}_{\text{senc}}^{(r)} \,|\, \text{Oracle}(\mathsf{auth})$ is related to $\mathcal{F}_{\text{senc}}^{(r+1)}$. Note that these systems are already very close because every key is treated (real or ideal) in the same way. The only difference occurs upon key collisions (for honestly generated keys) or if the environment is able to guess a key that is ideally not known to it. But because these keys where only encrypted ideally we can show that this probability is negligible.

Formulation of the Simulator. The simulator $\mathcal{S}_{\mathcal{F}_{\text{senc}}} = \mathcal{S}_{\mathcal{F}_{\text{senc}}}(\Sigma, T_{\text{adv}})$ is defined in Figure 17. On the first activation it provides the encryption and decryption algorithms to $\mathcal{F}_{\text{senc}}$. Then, key generation request from $\mathcal{F}_{\text{senc}}$ are forwarded to the environment. Key generation complete messages from the environment are forwarded to $\mathcal{F}_{\text{senc}}$ but if the key is uncorrupted then $\mathcal{S}_{\mathcal{F}_{\text{senc}}}$ generates a key with $\text{gen}(1^\eta)$ and sends this key to $\mathcal{F}_{\text{senc}}$. All messages between environment and $\mathcal{F}_{\text{ltsenc}}$ or $\mathcal{F}_{\text{pke}}$ are forwarded.

Formulation of Hybrid Systems. We define the hybrid systems $\mathcal{F}_{\text{senc}}^{(r)}$ and $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ for all $r \in \mathbb{N}$, see below.

$\mathcal{F}_{\text{senc}}^{(r)}$ behaves like $\mathcal{F}_{\text{senc}}$ except that the order in which unknown keys are used is tracked, as in $\mathcal{F}^*$. All keys with order $< r$ are treated ideally (as in $\mathcal{F}_{\text{senc}}$) but keys with order $\geq r$ are treated as in the real world. In $\mathcal{F}_{\text{senc}}$ the adversary was not able to insert keys (upon key generation, store, or decryption with corrupted or known keys) that collide with unknown keys (guessing of keys that are ideally not known). Here, this is only guaranteed for keys of order $\leq r$ or as long as there are no keys of order $> r$ (i.e., *nextused* $\leq r$).

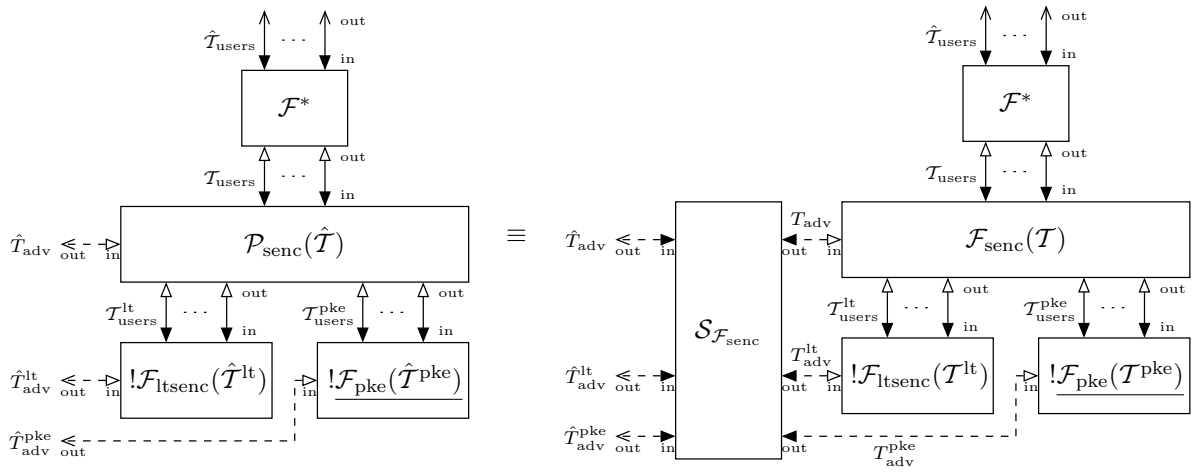

Figure 16: Theorem 9,$\mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$ and $\mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$ are indistinguishable. Solid lines represent I/O tapes and dashed lines network tapes. Filled arrow heads represent enriching and unfilled arrow heads consuming input tapes.

$\mathcal{S}_{\mathcal{F}_{\text{senc}}}(\Sigma = (\text{gen}, \text{enc}, \text{dec}), T_{\text{adv}})$

Tapes: network in: $T^{\text{out}}_{\text{adv}}$ (denoted by "**from** $\mathcal{F}_{\text{senc}}$"), $T^{\text{lt}\,\text{out}}_{\text{adv}}$ ("**from** $\mathcal{F}_{\text{ltsenc}}$"), $T^{\text{pke}\,\text{out}}_{\text{adv}}$ ("**from** $\mathcal{F}_{\text{pke}}$"), $\hat{T}^{\text{in}}_{\text{adv}}$ ("**from** $\hat{T}_{\text{adv}}$"), $\hat{T}^{\text{lt}\,\text{in}}_{\text{adv}}$ ("**from** $\hat{T}^{\text{lt}}_{\text{adv}}$"), $\hat{T}^{\text{pke}\,\text{in}}_{\text{adv}}$ ("**from** $\hat{T}^{\text{pke}}_{\text{adv}}$")
network out: $T^{\text{in}}_{\text{adv}}$ (denoted by "**to** $\mathcal{F}_{\text{senc}}$"), $T^{\text{lt}\,\text{in}}_{\text{adv}}$ ("**to** $\mathcal{F}_{\text{ltsenc}}$"), $T^{\text{pke}\,\text{in}}_{\text{adv}}$ ("**to** $\mathcal{F}_{\text{pke}}$"), $\hat{T}^{\text{out}}_{\text{adv}}$ ("**to** $\hat{T}_{\text{adv}}$"), $\hat{T}^{\text{lt}\,\text{out}}_{\text{adv}}$ ("**to** $\hat{T}^{\text{lt}}_{\text{adv}}$"), $\hat{T}^{\text{pke}\,\text{out}}_{\text{adv}}$ ("**to** $\hat{T}^{\text{pke}}_{\text{adv}}$")

0. Upon first activation from $\mathcal{F}_{\text{senc}}$:
   send (Algorithms, enc, dec) **to** $\mathcal{F}_{\text{senc}}$, **recv** (Ack) **from** $\mathcal{F}_{\text{senc}}$
   proceed as on normal activation (Step 1 to 3)
1. Upon receiving (p, KeyGen) from $\mathcal{F}_{\text{senc}}$: Send (p, KeyGen) to $\hat{T}_{\text{adv}}$.
2. Upon receiving $(p, \mathsf{KeyGen}, \mathit{corrupt}, k')$ from $\hat{T}_{\text{adv}}$, $\mathit{corrupt} \in \{\mathsf{false}, \mathsf{true}\}$, $|k'| \leq q(\eta)$ (where q is the polynomial associated with Σ that bounds the runtime of the algorithms): If $\mathit{corrupt} = \mathsf{false}$ then $k \leftarrow \text{gen}(1^\eta)$, otherwise, $k := k'$. Send $(p, \mathsf{KeyGen}, \mathit{corrupt}, k)$ to $\mathcal{F}_{\text{senc}}$.
3. Forward all messages between $\hat{T}^{\text{lt}}_{\text{adv}}$, $\hat{T}^{\text{pke}}_{\text{adv}}$ and $\mathcal{F}_{\text{ltsenc}}$, $\mathcal{F}_{\text{pke}}$, respectively.

Figure 17: Simulator $\mathcal{S}_{\mathcal{F}_{\text{senc}}}$ for the realization of $\mathcal{F}_{\text{senc}}$.

More formally: The functionality $\mathcal{F}_{\text{senc}}^{(r)}$ has two additional variables $nextused \in \mathbb{N}$ (initially 1) and $used\colon \{0,1\}^* \dashrightarrow \mathbb{N}$ (partially defined, initially domain $= \emptyset$). The condition "$k \notin \mathcal{K}$ or ($corrupt = \mathsf{true}$ and $k \in \mathcal{K}_{\text{known}}$)" of receive rule 3. (key generation) of $\mathcal{F}_{\text{senc}}$ changes to "$k \notin \mathcal{K}$ or ($corrupt = \mathsf{true}$ and not $\text{KeyGuess}^{(r)}(k)$)" where $\text{KeyGuess}^{(r)}(k) = \mathsf{true}$ if and only if $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ and ($\bot \neq used(k) \leq r$ or $nextused \leq r$). For encryption and decryption with short-term keys the way the ciphertext (rule 8.) and plaintext (rule 11.), respectively, is computed changes as follows. For store and decryption of short-term, long-term and public keys, the key guessing prevention changes as well:

STORE: Upon receiving (p, Store, k) from $T \in \mathcal{T}_{\text{users}}$ do: If $\text{KeyGuess}^{(r)}(k)$ then $ptr := \bot$, otherwise, if exists ptr' s.t. $key(p, ptr') = k$ then $ptr := ptr'$, otherwise, $ptr := nextpointer(p)$++, $key(p, ptr) := k$, and $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k\}$. Finally, send (p, Store, ptr) to T.

ENCRYPT m', OBTAIN CIPHERTEXT c:

if ptr is invalid **or** $m' = \bot$ **or** $m' \notin \text{dom}(L)_\eta$ **or** $enc = dec = \bot$ **then**
  $c := \bot$
else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
  if $k \notin \text{dom}(used)$ **then** $used(k) := nextused$++
  if $used(k) < r$ **then**
    $\overline{m} \leftarrow L(1^\eta, m')$, $c \leftarrow \text{sim}_{q(\eta+|\overline{m}|)}(enc, (k, \overline{m}))$
    if $\text{sim-det}_{q(\eta+|c|)}(dec, (k, c)) = \overline{m}$ **then**
      $decTable(k) := decTable(k) \cup \{(m', c)\}$
    else
      $c := \bot$
  else {$used(k) \geq r$}
    $c \leftarrow \text{sim}_{q(\eta+|m'|)}(enc, (k, m'))$
else {key k is known}
  $c \leftarrow \text{sim}_{q(\eta+|m'|)}(enc, (k, m'))$

DECRYPT c, OBTAIN PLAINTEXT m':

if ptr is invalid (i.e., $(p, ptr) \notin \text{dom}(key)$) **or** $enc = dec = \bot$ **then**
  $m' := \bot$
if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
  if $\bot \neq used(k) < r$ **or** $nextused \leq r$ **then**
    if $\exists! m'' \in \{0,1\}^*\colon (m'', c) \in decTable(k)$ **then**
      $m' := m''$
    else {decryption of c is ambiguous or c does not exist in $decTable(k)$}
      $m' := \bot$
  else
    $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$
else {key k is known}
  $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$

PREVENT KEY GUESSING: If the decryption was not ideal and the plaintext m' contains (Key, k') with $\text{KeyGuess}^{(r)}(k') = \mathsf{true}$ then $m' := \bot$.

$\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ behaves like $\mathcal{F}_{\text{senc}}^{(r)}$ except that it connects to Oracle and the key with order r is relayed out and handled by calls to Oracle.

More formally: $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ has the additional I/O output tape $T_{\text{oracle}}^{\text{in}}$ and I/O input tape $T_{\text{oracle}}^{\text{out}}$. Furthermore, for encryption and decryption with short-term keys the way the ciphertext (rule 8.) and plaintext (rule 11.), respectively, is computed changes as follows (compared to $\mathcal{F}_{\text{senc}}^{(r)}$):

ENCRYPT m', OBTAIN CIPHERTEXT c:

if ptr is invalid **or** $m' = \bot$ **or** $m' \notin \text{dom}(L)_\eta$ **or** $enc = dec = \bot$ **then**
  $c := \bot$
else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
  if $k \notin \text{dom}(used)$ **then**
    $used(k) := nextused$++
    if $used(k) = r$ **then**

send (KeyGen) **to** T_{oracle} , **recv** (Ack) **from** T_{oracle}
if $used(k) < r$ **then**
$\overline{m} \leftarrow L(1^\eta, m')$, $c \leftarrow \text{sim}_{q(\eta+|\overline{m}|)}(enc, (k, \overline{m}))$
if $\text{sim-det}_{q(\eta+|c|)}(dec, (k, c)) = \overline{m}$ **then**
$decTable(k) := decTable(k) \cup \{(m', c)\}$
else
$c := \bot$
else if $used(k) = r$ **then**
send (Enc, m') **to** T_{oracle} , **recv** $(\mathsf{Ciphertext}, c)$ **from** T_{oracle}
else {$used(k) > r$}
$c \leftarrow \text{sim}_{q(\eta+|m'|)}(enc, (k, m'))$
else {key k is known}
$c \leftarrow \text{sim}_{q(\eta+|m'|)}(enc, (k, m'))$

DECRYPT c, OBTAIN PLAINTEXT m':
if ptr is invalid (i.e., $(p, ptr) \notin \text{dom}(key)$) **or** $enc = dec = \bot$ **then**
$m' := \bot$
else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
if $\bot \neq used(k) < r$ **or** $nextused \leq r$ **then**
if $\exists! m'' \in \{0,1\}^*$: $(m'', c) \in decTable(k)$ **then**
$m' := m''$
else {decryption of c is ambiguous or c does not exist in $decTable(k)$}
$m' := \bot$
else if $used(k) = r$ **then**
send (Dec, c) **to** T_{oracle} , **recv** $(\mathsf{Plaintext}, m')$ **from** T_{oracle}
else
$m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$
else {key k is known}
$m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$

We can prove the following invariants for $\mathcal{F}^* \mid \mathcal{F}_{\text{senc}}^{(r)} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$: If a short-term key k' is encrypted by some unknown short-term key k then k' is known, k' has not yet been used for encryption, or the used order of k is smaller than the one of k'. In particular this implies that the used order is never violated. Furthermore, if a short-term key has used order $\leq r$ then it is always unknown.

Lemma 5. *For all $r \in \mathbb{N}$ let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{F}^* \mid \mathcal{F}_{\text{senc}}^{(r)} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}})$ and ρ be a run of $\mathcal{E} \mid \mathcal{F}^* \mid \mathcal{F}_{\text{senc}}^{(r)} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a)$ for any η, a. In the following, for a configuration C in ρ, by $C.x$ we denote the value of variable x in the configuration of $\mathcal{F}_{\text{senc}}^{(r)}$ in C.*

1. *If a key k' is encrypted by some unknown short-term key k (i.e., in some configuration C in ρ, $\mathcal{E}$ sends $(p, \mathsf{Enc}, ptr, m)$ to $\mathcal{F}_{\text{senc}}$ and receives $(p, \mathsf{Ciphertext}, c)$ where $c \neq \bot$, $(\mathsf{Key}, ptr') \in m$, $C.key(p, ptr) = k \notin \mathcal{K}_{\text{known}}$ and $C.key(p, ptr') = k'$) then k' is known (i.e., $k' \in C.\mathcal{K}_{\text{known}}$), k' has not yet been used for encryption (i.e., $k' \notin \text{dom}(C.used)$), or the used order of k is smaller than the one of k' (i.e., $\bot \neq C.used(k) < C.used(k') \neq \bot$).*
2. *If a key k has used order $\leq r$ (i.e., there is a configuration C in ρ where $\bot \neq C.used(k) \leq r$) then k is always unknown (i.e., $k \notin C.\mathcal{K}_{\text{known}}$ for any C in ρ).*

Proof. ad 1. It is easy to see that this is guaranteed by $\mathcal{F}^*$.

ad 2. Let C be the configuration in ρ where a key k is first used and gets used order $\leq r$. By definition, k is unknown in C. Later, in some configuration C', k cannot be encrypted by a known or corrupted key because $\mathcal{F}^*$ would trigger (CommitProblem) in that case. Hence, k can only be encrypted by unknown or uncorrupted keys. It is easy to see that Lemma 3 also holds for $\mathcal{F}_{\text{senc}}^{(r)}$ for keys with used order $\leq r$ because of the definition of $\text{KeyGuess}^{(r)}$, hence, k is always unknown. □

Proof of Theorem 9. Let $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{F}^* \mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}})$ and $p_{\mathcal{E}}$ be a polynomial such that the overall length of all messages output by $\mathcal{E}$ in any run of $\mathcal{E} \mid \mathcal{Q}(1^\eta, a)$ for any system $\mathcal{Q}$, security parameter

$\eta \in \mathbb{N}$ and initial input $a \in \{0,1\}^*$ is bound by $p_{\mathcal{E}}(\eta + |a|)$. Since $\mathcal{E}$ is an environmental system (all input tapes are consuming) such a polynomial always exists.

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

$$\mathcal{C}^{(r)} = \mathcal{E} \,|\, \mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}}^{(r)} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$$
$$\widehat{\mathcal{C}}_b^{(r)} = \mathcal{E} \,|\, \mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \widehat{\mathcal{F}}_{\text{senc}}^{(r)} \,|\, \text{Oracle}(b) \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \ .$$

Next, we define an error set (i.e., a negligible set of runs we do not want to consider) for collisions of honestly generated keys. Let $B_{\text{coll}}^{(r)}(1^\eta, a)$ be the set of runs of $\mathcal{C}^{(r)}(1^\eta, a)$ where the simulator generates a key that collides with some key in $\mathcal{F}_{\text{senc}}^{(r)}$. More formally: Where at some point during the run the simulator sends a message of the shape $(p, \mathsf{KeyGen}, \mathsf{false}, k)$ for some $p \in \text{PIDS}$ and $k \in \mathcal{F}_{\text{senc}}^{(r)}.\mathcal{K}$.

The following lemma is used in the proofs of Lemma 7 and 8.

Lemma 6. *There exists a negligible function f_{coll} such that for all $r \in \mathbb{N}$, $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$*

$$\Pr\left[B_{\text{coll}}^{(r)}(1^\eta, a)\right] \leq f_{\text{coll}}(1^\eta, a) \ . \tag{3}$$

Proof. Let $f_{\text{coll}}(\eta) = \max\{\text{gen}(1^\eta) = k \mid k \in \{0,1\}^*\}$ (because gen is polynomial the set of keys generated by $\text{gen}(1^\eta)$ is finite, so, this maximum exists). Since $\mathcal{E}$'s output is polynomially bounded by $p_{\mathcal{E}}$, in every run of $\mathcal{C}^{(r)}(1^\eta, a)$ the number of keys in $\mathcal{F}_{\text{senc}}^{(r)}$ ($|\mathcal{K}|$) is polynomially bounded. Hence, $\Pr\left[B_{\text{coll}}^{(r)}(1^\eta, a)\right]$ is bounded by a polynomial in $\eta + |a|$ times $f_{\text{coll}}(\eta)$. Thus, it suffice to show that $f_{\text{coll}}(\eta)$ is negligible (as a function in η). Note that this bound does not depend on r, thus, we obtain a uniform bound for all $r \in \mathbb{N}$.

Let $A^{O(\cdot,\cdot)}(1^\eta)$ be the following IND-CPA adversary for Σ. First, A computes $k^* \leftarrow \text{gen}(1^\eta)$ and chooses plaintexts $m_0 \neq m_1, |m_0| = |m_1|$. Then A request the oracle for $c \leftarrow O(m_0, m_1)$ and outputs 0 if $\text{dec}(k^*, c) = m_0$, and 1 otherwise.

If the key k^* generated by A is equal to the key k generated in the IND-CPA game, which happens with probability at least $f_{\text{coll}}^2(\eta)$, then A always chooses the right bit. Hence, $f_{\text{coll}}(\eta)$ is negligible as Σ is IND-CPA secure. □

Lemma 7. *There exists a negligible function f_0 such that*

$$\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \equiv_{f_0} \mathcal{C}^{(0)} \tag{4}$$

and

$$\Pr\left[\mathcal{E} \,|\, \mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a) \rightsquigarrow 1\right] = \Pr\left[\mathcal{C}^{(p_{\mathcal{E}}(\eta+|a|))}(1^\eta, a) \rightsquigarrow 1\right] \tag{5}$$

for all $\eta \in \mathbb{N}$, $a \in \{0,1\}^*$.

Proof. ad (4): Let $f_0(1^\eta, a) = \Pr\left[B_{\text{coll}}^{(0)}(1^\eta, a)\right]$ (see Lemma 6). By Lemma 6 we have that f_0 is negligible. Now, we show that $\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \equiv_{f_0} \mathcal{C}^{(0)}$.

Note that in every run of $\mathcal{C}^{(0)}(1^\eta, a)$ it always holds that $\mathit{nextused} \geq 1$ and $\mathit{used}(k) = \perp$ or $\mathit{used}(k) > 0$. In particular this implies that $\text{KeyGuess}^{(0)}(k) = \mathsf{false}$ for all k. One then easily verifies that every run of $\mathcal{C}^{(0)}(1^\eta, a)$ where $B_{\text{coll}}^{(0)}(1^\eta, a)$ does not occur corresponds, i.e., can be injectively mapped, to a run of $\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a)$ with the same overall output and probability. By Theorem 4 we conclude

$$\left|\Pr\left[\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a) \rightsquigarrow 1\right] - \Pr\left[\mathcal{C}^{(0)}(1^\eta, a) \rightsquigarrow 1\right]\right| \leq f_0(1^\eta, a) \ .$$

ad (5): In every run of $\mathcal{C}^{(p_{\mathcal{E}}(\eta+|a|))}(1^\eta, a)$ it always holds that $\mathit{nextused} \leq p_{\mathcal{E}}(\eta + |a|)$ and for all k we have $\text{KeyGuess}^{(p_{\mathcal{E}}(\eta+|a|))}(k) = \mathsf{true}$ iff $k \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$. Hence, one easily verifies by code inspection that the behavior of the two systems $\mathcal{C}^{(p_{\mathcal{E}}(\eta+|a|))}$ and $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$ does not differ at all upon security parameter η and initial input a. □

Lemma 8. *There exist negligible functions $f_{\mathsf{real}}, f_{\mathsf{ideal}}$ such that*

$$\mathcal{C}^{(r)} \equiv_{f_{\mathsf{real}}} \widehat{\mathcal{C}}_{\mathsf{real}}^{(r)} \quad \textit{for all } r \in \mathbb{N} \textit{ and} \tag{6}$$
$$\mathcal{C}^{(r+1)} \equiv_{f_{\mathsf{ideal}}} \widehat{\mathcal{C}}_{\mathsf{auth}}^{(r)} \quad \textit{for all } r \in \mathbb{N} \ . \tag{7}$$

The proof of this lemma can be found in the appendix.

Finally, we prove Theorem 9:

Proof of Theorem 9. Let $\mathcal{E}'$ be the system that upon input of security parameter η and initial input a first chooses $r \in \{0, \ldots, p_{\mathcal{E}}(\eta + |a|) - 1\}$ uniformly at random and then behaves exactly like $\mathcal{E} \,|\, \mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \widehat{\mathcal{F}}^{(r)}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a)$. Clearly, $\mathcal{E}' \in \text{Con}_{\mathbf{E}}(\text{Oracle})$ and by Lemma 1 we find a negligible function f_{O} such that

$$\mathcal{E}' \mid \text{Oracle}(\mathsf{real}) \equiv_{f_{\text{O}}} \mathcal{E}' \mid \text{Oracle}(\mathsf{auth}) \ . \tag{8}$$

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

$$\begin{aligned} \Pr\left[\widehat{\mathcal{C}}^{(r)}_b(1^\eta, a) \rightsquigarrow 1\right] &= \Pr\left[(\mathcal{E}' \,|\, \text{Oracle}(b))(1^\eta, a) \rightsquigarrow 1 \,\middle|\, \mathcal{E}' \text{ chooses } r\right] \\ &= p_{\mathcal{E}}(\eta + |a|) \cdot \Pr\left[(\mathcal{E}' \,|\, \text{Oracle}(b))(1^\eta, a) \rightsquigarrow 1 \text{ and } \mathcal{E}' \text{ chooses } r\right] \ . \end{aligned} \tag{9}$$

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

$$\begin{aligned} &\left|\Pr[\mathcal{E} \,|\, \mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}] - \Pr[\mathcal{E} \,|\, \mathcal{S}_{\mathcal{F}_{\text{senc}}} \,|\, \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}]\right| \\ &\overset{(4),(5)}{\leq} \left|\Pr\left[\mathcal{C}^{(0)}\right] - \Pr\left[\mathcal{C}^{(p_{\mathcal{E}})}\right]\right| + f_0 = \left|\sum_{r<p_{\mathcal{E}}} \Pr\left[\mathcal{C}^{(r)}\right] - \Pr\left[\mathcal{C}^{(r+1)}\right]\right| + f_0 \\ &\overset{(6),(7)}{\leq} \left|\sum_{r<p_{\mathcal{E}}} \Pr\left[\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}\right] - \Pr\left[\widehat{\mathcal{C}}^{(r)}_{\mathsf{auth}}\right]\right| + f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &\overset{(9)}{=} p_{\mathcal{E}}\left|\sum_{r<p_{\mathcal{E}}} \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{real}) \text{ and } \mathcal{E}' \text{ chooses } r\right] - \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{auth}) \text{ and } \mathcal{E}' \text{ chooses } r\right]\right| \\ &\qquad + f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &= p_{\mathcal{E}}\left|\Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{real})\right] - \Pr\left[\mathcal{E}' \mid \text{Oracle}(\mathsf{auth})\right]\right| + f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}}) \\ &\overset{(8)}{\leq} f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}} + f_{\text{O}}) \ . \end{aligned}$$

Since $f_0 + p_{\mathcal{E}}(f_{\mathsf{real}} + f_{\mathsf{ideal}} + f_{\text{O}})$ is negligible, $\mathcal{F}^* \,|\, \mathcal{P}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}} \leq^{SS} \mathcal{F}^* \,|\, \mathcal{F}_{\text{senc}} \,|\, !\mathcal{F}_{\text{ltsenc}} \,|\, !\underline{\mathcal{F}_{\text{pke}}}$. This concludes the proof. □

6 Unauthenticated Symmetric Key Encryption

In this section we show how to relax the functionality $\mathcal{F}^{\text{auth}}_{\text{senc}}$ to obtain a functionality $\mathcal{F}_{\text{senc}}$ which is realizable by an IND-CCA secure encryption scheme, instead of an authenticated encryption scheme.

6.1 The Functionality

We define $\mathcal{F}^{\text{unauth}}_{\text{senc}}$ as $\mathcal{F}_{\text{senc}} = \mathcal{F}^{\text{auth}}_{\text{senc}}$ except that upon decryption if the ciphertext was not honestly generate then we do not refuse decryption but we decrypt it with the decryption algorithm. Furthermore, we treat the decryption as a decryption with a known key because this ciphertext was not generated by an honest party. That is, for decryption with short-term keys, we change $\mathcal{F}_{\text{senc}}$ as follows:

DECRYPT c, OBTAIN PLAINTEXT m':
  if ptr is invalid (i.e., $(p, ptr) \notin \text{dom}(key)$) **or** $enc = dec = \bot$ **then**
    $m' := \bot$
  if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
    if $\exists! m'' \in \{0,1\}^*$: $(m'', c) \in decTable(k)$ **then** {exists unique m''}
      $m' := m''$
    else if $\forall m' \in \{0,1\}^*$: $(m', c) \notin decTable(k)$ **then** {c does not exist in $decTable(k)$}
      $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$
    else {decryption of c is ambiguous}

$m' := \bot$
else {key k is known}
$m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$

PREVENT KEY GUESSING: If decryption was *not ideal* (i.e., $key(p, ptr) \in \mathcal{K}_{\text{known}}$ or c does not exist in $decTable(key(p, ptr))$) and m' contains (Key, k') with $k' \in \mathcal{K} \setminus \mathcal{K}_{\text{known}}$ then $m' := \bot$.

UPDATE $\mathcal{K}_{\text{known}}$: $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$ if decryption was not ideal, as defined above.

6.2 Realizing the Functionality

Recall $\mathcal{P}_{\text{senc}}$ from Section 5.2 and the notation and tape names from Section 5.2.3.

In the proof of Theorem 9, we needed, among other error sets, an error set $B_{\text{int}}^{(r+1)}$ which is the set of all runs where the environment produces a ciphertext that decrypts under an unknown and unused key before the $(r+1)$-th key has been used, see Appendix A.2. Recall that the used order, as defined by $\mathcal{F}^*$, records keys only when they are first used for encryption, not for decryption. The reason why we needed this error set is because upon decryption with an unknown and unused key k before the $(r+1)$-th key has been used, to be more precise where $nextused = r+1$, $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ treats the decryption non-ideal but $\mathcal{F}_{\text{senc}}^{(r+1)}$ treats it ideal and returns $\bot$.

One could think to easily circumvent this problem by changing the definition of performing decryptions ideally (i.e., returning $\bot$) for unused keys as long $nextused \leq r+1$ and instead perform them non-ideally (i.e, actually decrypt with the key). But this would not work because one of these keys might be the r-th key, but $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ does not know this yet because it has not yet been used for encryption and would have to use Oracle for decryption, otherwise the simulation would fail.

Fortunately, for INT-CTXT secure encryption schemes $B_{\text{int}}^{(r+1)}$ is negligible, as we have shown in Lemma 12. For only IND-CCA secure encryption schemes this is not true in general. In fact, given an IND-CCA secure encryption scheme, it is easy to construct an IND-CCA secure encryption scheme where $B_{\text{int}}^{(r+1)}$ is not negligible.

One solution would be to require that the encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ is not only IND-CCA secure but also satisfies the following property. For all probabilistic polynomial time adversaries A it is almost impossible for A to produce a ciphertext c that decrypts under a randomly generated key k where A has no knowledge of k at all. More formally, where

$$\Pr[k \leftarrow \text{gen}(1^\eta), c \leftarrow A(1^\eta, a) : \text{dec}(k, c) \neq \bot]$$

is negligible (as a function in η, a). For example, this is fulfilled by an IND-CCA secure encryption scheme where the ciphertexts are "connected" to the keys they where produced with, i.e., where every ciphertext decrypts under at most one key.

However, requiring this additional assumption seems artificial and indeed we can proof realizability without it if we slightly change the used order. The used order, as defined by $\mathcal{F}^*$, is insufficient for proving that one hybrid system is close to the next hybrid system because for unauthenticated encryption we need to know how to handle (real or ideal) keys upon decryption, as argued above. Thus, we change the used order such that we also count the use of a key for decryption as *used*. However, commitment problems still only occur if a key used ideally for encryption (not for decryption) later becomes known. Because we want $\mathcal{F}^*$ to be as less restrictive as possible we do not change this aspect. This induces the following: A key that is used for decryption where it is unknown might later become known, i.e., the invariant that all keys that have used order $\leq r$ are unknown is no longer true. See the proof of Theorem 10 on how this can be dealt with.

Definition of $\mathcal{F}_{\text{unauth}}^*$. As motivated above, we define $\mathcal{F}_{\text{unauth}}^*$ as $\mathcal{F}^*$ with the following exceptions:

1. The commitment problem prevention (for short-term, long-term and public keys) changes:

   PREVENT COMMITMENT PROBLEM: If $c \neq \bot$ and m contains an unknown key with key ID i such that $decTable_{\text{st}}(i) \neq \emptyset$ (i.e., i has been used for encryption before) then send $(\mathsf{CommitProblem})$ and terminate.

2. For decryption with short-term keys we have the following additional rule: If $c \neq \bot$, $keyid(p, ptr) \notin \mathcal{K}_{\text{known}}$, and $i = keyid(p, ptr) \notin \text{dom}(used)$ then $used(i) := nextused$++.

We are now able to state the main theorem for $\mathcal{F}_{\text{senc}}^{\text{unauth}}$:

Theorem 10. *Let L be a leakage algorithm which leaks exactly the length of a message and Σ be a symmetric encryption scheme with domain $\text{dom}(L)$. Then for all polynomials q_{st}, q_{lt} and q_{pke} where q_{st} is sufficiently large (such that Σ is bounded by q_{st}) it holds that Σ is IND-CCA secure if and only if*

$$\mathcal{F}_{\text{unauth}}^{*} \,|\, \mathcal{P}_{\text{senc}}(\Sigma) \,|\, !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \hat{\mathcal{T}}^{\text{lt}}) \,|\, \underline{!\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \hat{\mathcal{T}}^{\text{pke}})} \leq^{SS} \mathcal{F}_{\text{unauth}}^{*} \,|\, \mathcal{F}_{\text{senc}}^{\text{unauth}}(q_{\text{st}}, L) \,|\, !\mathcal{F}_{\text{ltsenc}}(q_{\text{lt}}, L, \mathcal{T}^{\text{lt}}) \,|\, \underline{!\mathcal{F}_{\text{pke}}(q_{\text{pke}}, L, \mathcal{T}^{\text{pke}})} \ .$$

This holds for both $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{auth}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$. Also, this holds if the leakage algorithms used by $\mathcal{F}_{\text{senc}}^{\text{unauth}}$, $\mathcal{F}_{\text{ltsenc}}$ and $\mathcal{F}_{\text{pke}}$ differ.

This theorem yields corresponding corollaries for non-committing and used order respecting environments and protocols as in Section 5.2.3.

Proof of Theorem 10. We only sketch the proof of the direction from left to right and highlight the differences compared to the proof of Theorem 9. We use exactly the same simulator $\mathcal{S}_{\mathcal{F}_{\text{senc}}}$.

The hybrid system $\mathcal{F}_{\text{senc}}^{(r)}$ changes on how ideal decryption is done in the way $\mathcal{F}_{\text{senc}}^{\text{unauth}}$ changed compared to $\mathcal{F}_{\text{senc}}^{\text{auth}}$. Furthermore, upon decryption with unknown and unused short-term keys the short-term key gets assigned a used value. For completeness, we give the full definition of the decryption rule with short-term keys of $\mathcal{F}_{\text{senc}}^{(r)}$:

DECRYPT c, OBTAIN PLAINTEXT m':
  if *ptr* is invalid (i.e., $(p, ptr) \notin \text{dom}(key)$) **or** $enc = dec = \bot$ **then**
    $m' := \bot$
  else if $(k := key(p, ptr)) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
    if $k \notin \text{dom}(used)$ **then** $used(k) := nextused$++
    if $used(k) < r$ **then**
      if $\exists! m'' \in \{0,1\}^*$: $(m'', c) \in decTable(k)$ **then** {exists unique m''}
        $m' := m''$
      else if $\forall m' \in \{0,1\}^*$: $(m', c) \notin decTable(k)$ **then** {c not in $decTable(k)$}
        $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$
      else {decryption of c is ambiguous}
        $m' := \bot$
    else
      $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$
  else {key k is known}
    $m' := \text{sim-det}_{q(\eta+|c|)}(dec, (k, c))$

Now, the r-th key might become known after it has been used as an unknown key for decryption. When $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ performs encryptions with the r-th key it relays to Oracle. Now, it has to have the ability to treat the key in Oracle as a known key which of course is only possible if $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ actually knows the bit string. By $\mathcal{F}_{\text{unauth}}^{*}$ this change only needs to be possible until Oracle has been first used for encryption. Therefore, in Figure 18, we define Oracle$^{\text{unauth}}$ which is identical to Oracle except that it is corruptible and reveals the key upon corruption. It is incorruptible once it has been used for encryption.

As for Oracle, the real variant realizes the ideal variant if the encryption scheme is IND-CCA secure.

Lemma 9. *Let L be a length preserving leakage algorithm and Σ an IND-CCA secure symmetric encryption scheme with domain $\text{dom}(L)$. Then,* $\text{Oracle}^{\text{unauth}}(\Sigma, \mathsf{real}, L) \leq^{SS} \text{Oracle}^{\text{unauth}}(\Sigma, \mathsf{ideal}, L)$.

Note: For systems $\mathcal{P}, \mathcal{Q}$ without network tapes, like $\text{Oracle}^{\text{unauth}}$, *$\leq^{SS}$ is symmetric, i.e., $\mathcal{P} \leq^{SS} \mathcal{Q}$ if and only if $\mathcal{Q} \leq^{SS} \mathcal{P}$.*

Proof. $\text{Oracle}^{\text{unauth}}$ in mode ideal differs from Oracle in mode unauth only by the ability to be corruptible. As long as $decTable = \emptyset$ the behavior of $\text{Oracle}^{\text{unauth}}$ in mode ideal and real is identical. Because it can be corrupted only as long as $decTable = \emptyset$ an environment trying to distinguish between mode ideal and real has no advantage if it corrupts $\text{Oracle}^{\text{unauth}}$. The proof proceeds as the proof of Lemma 1. □

$\text{Oracle}^{\text{unauth}}(\Sigma = (\text{gen}, \text{enc}, \text{dec}), \mathit{mode} \in \{\mathsf{ideal}, \mathsf{real}\}, L)$

Tapes: enriching I/O input tape $T^{\text{in}}_{\text{oracle}}$, I/O output tape $T^{\text{out}}_{\text{oracle}}$

State: $\mathit{state} \in \{\mathsf{init}, \mathsf{ok}, \mathsf{terminated}\}$ (initially init), $k \in \{0,1\}^* \cup \{\bot\}$ (initially $\bot$)
$\mathit{decTable} \subseteq \{0,1\}^* \times \{0,1\}^*$ (initially $\emptyset$)

1. Upon receiving (KeyGen) from T_{oracle} where $\mathit{state} = \mathsf{init}$:
   $k \leftarrow \text{gen}(1^\eta)$, $\mathit{state} := \mathsf{ok}$, **send** (Ack) **to** T_{oracle}
2. Upon receiving (Enc, m) from T_{oracle} where $\mathit{state} = \mathsf{ok}$, $m \in \text{dom}(L)_\eta$:
   if $\mathit{mode} = \mathsf{real}$ **then**
   $c \leftarrow \text{enc}(k, m)$
   else $\{\mathit{mode} = \mathsf{ideal}\}$
   $\overline{m} \leftarrow L(1^\eta, m)$, $c \leftarrow \text{enc}(k, \overline{m})$, $\mathit{decTable} := \mathit{decTable} \cup \{(m, c)\}$
   send $(\mathsf{Ciphertext}, c)$ **to** T_{oracle}
3. Upon receiving (Dec, c) from T_{oracle} where $\mathit{state} = \mathsf{ok}$:
   if $\mathit{mode} = \mathsf{real}$ **then**
   $m := \text{dec}(k, c)$
   else $\{\mathit{mode} = \mathsf{ideal}\}$
   $$m := \begin{cases} m' & \text{if } \exists! m' \colon (m', c) \in \mathit{decTable} \quad (\text{exists unique } m') \\ \text{dec}(k, c) & \text{if } \forall m' \colon (m', c) \notin \mathit{decTable} \\ \bot & \text{otherwise} \end{cases}$$
   send $(\mathsf{Plaintext}, m)$ **to** T_{oracle}
4. Upon receiving $(\mathsf{Corrupt})$ from T_{oracle} where $\mathit{state} = \mathsf{ok}$ and $\mathit{decTable} = \emptyset$:
   $\mathit{state} := \mathsf{terminated}$, **send** $(\mathsf{Corrupted}, k)$ **to** T_{oracle}

Figure 18: The IITM $\text{Oracle}^{\text{unauth}}$ is parameterized by an encryption scheme Σ, $\mathit{mode} \in \{\mathsf{real}, \mathsf{ideal}\}$ and a leakage algorithm L. The mode specifies whether the behavior is real or ideal.

Now, the definition of the hybrid system $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ changes on how ideal decryption is done in the way $\mathcal{F}^{(r)}_{\text{senc}}$ changed. Furthermore, if the r-key becomes known then Oracle$^{\text{unauth}}$ is corrupted and the key that used to be the r-th key is replaced by the key obtained from Oracle$^{\text{unauth}}$. Note that the r-th key can only become known upon encryption with a corrupted or known key and by $\mathcal{F}^{*}_{\text{unauth}}$ this will never happen after it has been used for encryption. The update $\mathcal{K}_{\text{known}}$ rule for encryption with short-term, long-term and public keys and the decryption rule with short-term keys of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ is as follows:

UPDATE $\mathcal{K}_{\text{known}}$: If $c \neq \bot$, encryption was not ideal and exists $(\mathsf{Key}, k') \in m'$ with $\mathit{used}(k') = r$ and $k' \notin \mathcal{K}_{\text{known}}$ then send $(\mathsf{Corrupt})$ to T_{oracle} and wait for receiving $(\mathsf{Corrupted}, k'')$ from T_{oracle}. Then, replace k' by k'' everywhere (i.e., in $\mathcal{K}, \mathcal{K}_{\text{known}}, \mathit{key}, \mathit{decTable}$ and used).

If $c \neq \bot$ and encryption was not ideal then $\mathcal{K}_{\text{known}} := \mathcal{K}_{\text{known}} \cup \{k' \mid (\mathsf{Key}, k') \in m'\}$.

DECRYPT c, OBTAIN PLAINTEXT m':

if ptr is invalid (i.e., $(p, \mathit{ptr}) \notin \mathrm{dom}(\mathit{key})$) **or** $\mathit{enc} = \mathit{dec} = \bot$ **then**
  $m' := \bot$
else if $(k := \mathit{key}(p, \mathit{ptr})) \notin \mathcal{K}_{\text{known}}$ **then** {key k is unknown}
  if $k \notin \mathrm{dom}(\mathit{used})$ **then**
    $\mathit{used}(k) := \mathit{nextused}$++
    if $\mathit{used}(k) = r$ **then**
      send (KeyGen) **to** T_{oracle} , **recv** (Ack) **from** T_{oracle}
  if $\mathit{used}(k) < r$ **then**
    if $\exists! m'' \in \{0,1\}^*\colon (m'', c) \in \mathit{decTable}(k)$ **then** {exists unique m''}
      $m' := m''$
    else if $\forall m' \in \{0,1\}^*\colon (m', c) \notin \mathit{decTable}(k)$ **then** {c not in $\mathit{decTable}(k)$}
      $m' := \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, (k, c))$
    else {decryption of c is ambiguous}
      $m' := \bot$
  else if $\mathit{used}(k) = r$ **then**
    send (Dec, c) **to** T_{oracle} , **recv** $(\mathsf{Plaintext}, m')$ **from** T_{oracle}
  else
    $m' := \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, (k, c))$
else {key k is known}
  $m' := \text{sim-det}_{q(\eta+|c|)}(\mathit{dec}, (k, c))$

To prove Theorem 10, we establish the same error sets: $B^{(r)}_{\text{coll}}$ and $B^{(r)}_{\text{guess-unused}}$. The error set $B^{(r),r'}_{\text{guess}}$ only changes slightly: we require that the guessed key (the r'-th key) is unknown. It is easy to adapt the proofs that these error sets are negligible to this setting. Instead of the INT-CTXT adversary an IND-CCA adversary can be used.

By adapting the proofs of Lemma 7 and 8, we can prove that there are negligible functions f_0, f_{real} and f_{ideal} such that

$$\mathcal{C}^{(0)} \equiv_{f_0} \mathcal{E} \mid \mathcal{F}^{*}_{\text{unauth}} \mid \mathcal{P}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}} \ , \tag{10}$$

$$\Pr[\mathcal{C}^{(p_{\mathcal{E}}(\eta+|a|))}(1^\eta, a) \rightsquigarrow 1] = \Pr[\mathcal{E} \mid \mathcal{S}_{\mathcal{F}_{\text{senc}}} \mid \mathcal{F}^{*}_{\text{unauth}} \mid \mathcal{F}^{\text{unauth}}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}(1^\eta, a) \rightsquigarrow 1] \tag{11}$$

for all $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$, and

$$\mathcal{C}^{(r)} \equiv_{f_{\mathsf{real}}} \widehat{\mathcal{C}}^{(r)}_{\mathsf{real}} \quad \text{for all } r \in \mathbb{N} \ , \tag{12}$$

$$\mathcal{C}^{(r+1)} \equiv_{f_{\mathsf{ideal}}} \widehat{\mathcal{C}}^{(r)}_{\mathsf{ideal}} \quad \text{for all } r \in \mathbb{N} \ . \tag{13}$$

Note that the proof of (13) does not require the error set $B^{(r)}_{\text{int}}$. It was used to prevent some bad behavior upon decryption with unused keys. By the definition of the new used order, this case never occurs because even keys only used for decryption are used.

The proof proceeds as for the authenticated case. □

7 Applications

As mentioned in the introduction, $\mathcal{F}_{\text{senc}}$ has applications both in the simulation- and game-based setting. We now illustrate the usefulness of $\mathcal{F}_{\text{senc}}$ in these settings by two examples. In what follows, let $\mathcal{F}_{\text{enc}} = \mathcal{F}_{\text{senc}} \mid !\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$, with $!\mathcal{F}_{\text{ltsenc}} \mid !\underline{\mathcal{F}_{\text{pke}}}$ being the bootstrapping component (see Theorem 9). We write $\mathcal{F}_{\text{enc}}^{\text{unauth}}$, if in $\mathcal{F}_{\text{enc}}$ we set $\mathcal{F}_{\text{senc}} = \mathcal{F}_{\text{senc}}^{\text{unauth}}$ and $\mathcal{F}_{\text{ltsenc}} = \mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$; analogously for $\mathcal{F}_{\text{enc}}^{\text{auth}}$. We write $\mathcal{P}_{\text{enc}}$ for a realization of $\mathcal{F}_{\text{enc}}$, as obtained in the previous sections. We write $\mathcal{P}_{\text{enc}}^{\text{unauth}}$ if $\mathcal{P}_{\text{enc}}$ is based on an IND-CCA secure symmetric encryption scheme and $\mathcal{P}_{\text{enc}}^{\text{auth}}$ in case of authenticated encryption.

7.1 Simulation-based Analysis of a Key Exchange Protocol

In this section, we analyze a variant of the Amended Needham Schroeder Symmetric Key (ANSSK) protocol [34]. 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. We show that this protocol realizes an ideal key exchange functionality $\mathcal{F}_{\text{ke}} = \mathcal{F}_{\text{ke}}(\text{gen})$ in case authenticated encryption is used. (We show that this is not true if the encryption scheme is merely IND-CCA secure, see below.) Due to our main result (Section 5.2.3), it suffices to show that ANSSK' realizes $\mathcal{F}_{\text{ke}}$ when encryption and decryption is performed based on $\mathcal{F}_{\text{enc}}^{\text{auth}}$. As we will see, the use of $\mathcal{F}_{\text{enc}}^{\text{auth}}$ 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 $\mathcal{F}_{\text{ke}}$.

The ANSSK' Protocol. The protocol can informally be described as follows. There are three roles, the initiator B, the responder A and the server S. In the following description, Na is a nonce, Kb is a short-term symmetric key generated by B and only used in one session, Kas and Kbs are long-term symmetric keys shared between A (resp., B) and S, and Kab is a short-term symmetric key chosen by S to be used as a session key by A and B. See Figure 19.

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. $A \to B$: $\{Kab, A\}_{Kb}$

Figure 19: The ANSSK' Protocol.

The initiator B generates a short-term key Kb, encrypts it together with A's identity under the key Kbs shared with the server, and sends this ciphertext to A. Then, A generates a nonce Na and sends it together with its own identity, B's identity and the received ciphertext to the server S. The server S decrypts the ciphertext with the key shared with B and checks if A's identity is included in the ciphertext and extracts the key Kb. Then S generates a fresh key Kab and creates the following two ciphertexts: i) The first is encrypted with the key Kb and contains the freshly generated key Kab and A's identity. ii) The second is encrypted with the key Kas shared with A and contains A's nonce, B's identity, Kab and the first ciphertext. This second ciphertext is sent to A. Now, A decrypts the ciphertext, checks the containing nonce and B's identity, accepts with Kab as a the session key exchanged with B and sends the containing ciphertext to B. Then B decrypts this ciphertext with Kb, checks that A's identity is contained and accepts with Kab as the session key exchanged with A.

Compared to the ANSSK protocol [34], ANSSK' differs in two aspects. First, the key Kb was originally a nonce of B and instead the server uses Kb for encryption it uses the long-term key Kbs. We made this modification to make the protocol more interesting to analyze as otherwise there would be no short-term keys that are used but only long-term keys. Second, we omitted the last two messages which are used for key confirmation. There, the exchanged key Kab is used to encrypt some nonces. It is easy to show that this implies that Kab is not indistinguishable from random (see e.g. [26, 5] where this problem is studied for other protocols), i.e., the ANSSK protocol does not satisfy cryptographic key indistinguishability and would not realize any ideal key exchange protocol which is based on this. If one would remove the key confirmation in the ANSSK protocol then it would realize the ideal key exchange functionality. The proof is even simpler than the one for the ANSSK' protocol.

The Ideal Key Exchange Functionality. We restate the ideal key exchange functionality $\mathcal{F}_{\text{ke}} = \mathcal{F}_{\text{ke}}(\text{gen})$ from [20] in the setting of the IITM-model. The parameter gen is a probabilistic key generation algorithm. The functionality describes one session of an ideal key exchange between two parties. The adversary has the ability to corrupt the functionality by sending a *corrupt* message. As usual in the definition of our functionalities and protocols, the environment can ask $\mathcal{F}_{\text{ke}}$ (via a special tape) whether it is corrupted. If the functionality is uncorrupted its behavior is as follows: After a party A (the initiator) has send some initial message that it is willing to exchange a key with another party B (the responder), the adversary can send a *complete* message for A upon which the functionality generates a key by $\text{gen}(1^\eta)$ (if not done so before) and outputs this key to A. Similar for the responder B, B has to send an initial message. Upon the second complete message, the already generated key is sent to the party. In the corrupted case, the adversary can decide which key (in every complete message she can choose a different key if she likes) is sent to the party. A complete is possible even if the peer has not sent its initial message, i.e., it is in that sense weaker than authenticated key exchange.

Note that this functionality only captures the single session and single party case, i.e., there is exactly one key exchange between two parties. The multi-party, multi-session version $!\underline{\mathcal{F}_{\text{ke}}}$ is the desired functionality for multi-party, multi-session ideal key exchange. By the composition theorem it can be realized if we have a realization of $\mathcal{F}_{\text{ke}}$.

The ANSSK' Protocol Realizes the Ideal KE Protocol. It is straightforward to specify the ANSSK' protocol as a protocol system $\mathcal{P}_{\text{anssk'}}$ in the IITM-model which relies on $\mathcal{F}_{\text{enc}}$ for encryption and decryption. Next, we only mention the main issues. $\mathcal{P}_{\text{anssk'}}$ describes a single session of the protocol (just as $\mathcal{F}_{\text{ke}}$), i.e., it consists of one IITM for each participant role: initiator, responder and server. We assume, similar as in formulations by Canetti et al. (see, e.g., [20]), that the machines in $\mathcal{P}_{\text{anssk'}}$ 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. In this first activation with the pair (A, B) every party exchanges its long-term keys with the server.

The send/receive behavior of these machines follows directly from the informal definition of the protocol (Figure 19). For encryption and decryption the functionality $\mathcal{F}_{\text{enc}}$ is called. Nonces are computed by choosing a value uniformly at random from $\{0,1\}^\eta$. The responder B generates the session key Kb by using $\mathcal{F}_{\text{enc}}$. The server computes Kab as $\text{gen}(1^\eta)$. Note that in this protocol Kab is never used as a key but rather as a secret nonce and, so, does not need to be a key in $\mathcal{F}_{\text{enc}}$.

For the IITMs for the initiator, responder and server we define static Byzantine corruption behavior, i.e., just after initialization, upon a corrupt message from the network interface they output their internal state to the adversary and will from then on forward all messages between the I/O interface and the adversary, giving the adversary complete control over this instance of the protocol. We require that if a party is corrupted all its short-term and long-term keys are corrupted as well. The environment has the ability to ask whether $\mathcal{F}_{\text{ke}}$ is corrupted, since, $\mathcal{P}_{\text{anssk'}}$ has to provide the same I/O interface as $\mathcal{F}_{\text{ke}}$, upon such a corrupted request by the environment $\mathcal{P}_{\text{anssk'}}$ returns false if and only if all the parties and all their keys are uncorrupted.

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 11. $\mathcal{P} \mid \mathcal{F}_{\text{enc}}^{\text{auth}} \leq^{SS} \mathcal{F}_{\text{ke}}$.

Before proving this theorem, we note that by the results of Section 5.2.3, 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 4. $\mathcal{P} \mid \mathcal{P}_{\text{enc}}^{\text{auth}} \leq^{SS} \mathcal{F}_{\text{ke}}$.

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, see below for the discussion of multiple sessions.

Proof sketch of Theorem 11. We first need to define a simulator: The simulator $\mathcal{S}$ simulates $\mathcal{P}_{\text{anssk'}} \mid \mathcal{F}_{\text{enc}}$ and sends a completion request to $\mathcal{F}_{\text{ke}}$ if $\mathcal{P}_{\text{anssk'}}$ 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 $\mathcal{E} \mid \mathcal{P}_{\text{anssk'}} \mid \mathcal{F}_{\text{enc}}^{\text{auth}}$ (without corruption), we have: i) Since $\mathcal{P}_{\text{anssk'}}$ 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 $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{enc}}^{\text{auth}}$ and the session key generated and output by $\mathcal{F}_{\text{ke}}$ in runs of $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{ke}}$. Therefore, it is easy to establish a one-to-one correspondence between the runs of $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{enc}}^{\text{auth}}$ and those of $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{ke}}$. Thus, we obtain: $\mathcal{E} \mid \mathcal{P} \mid \mathcal{F}_{\text{enc}}^{\text{auth}} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{ke}}$. □

We emphasize that due to the use of $\mathcal{F}_{\text{enc}}^{\text{auth}}$, we did not need to reason about IND-CPA and INT-CTXT games. We also remark that we did not use the nonce Na in our argumentation. 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 3, we obtain from Theorem 11 that $!\underline{\mathcal{P}_{\text{anssk'}}} \mid !\underline{\mathcal{F}_{\text{enc}}^{\text{auth}}} \leq^{SS} !\underline{\mathcal{F}_{\text{ke}}}$, i.e., the multi-session version of $\mathcal{P}_{\text{anssk'}}$ realizes ideal key exchanges in multiple sessions. In this realization, $\mathcal{P}_{\text{anssk'}}$ uses fresh long-term keys in every session. However, we can apply the joint state theorem (Theorem 6) 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

$$!\underline{\mathcal{P}_{\text{anssk'}}} \mid !\underline{\mathcal{F}_{\text{senc}}^{\text{auth}}} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{js}} \mid !\mathcal{F}_{\text{ltsenc}}^{\text{auth}} \leq^{SS} !\underline{\mathcal{F}_{\text{ke}}} \ .$$

By the composition theorems (Theorem 2 and 3), Theorem 5, Corollary 3, and transitivity of $\leq^{SS}$ we finally obtain

$$!\underline{\mathcal{P}_{\text{anssk'}}} \mid !\underline{\mathcal{P}_{\text{senc}}(\Sigma)} \mid !\mathcal{P}_{\text{ltsenc}}^{\text{js}} \mid !\mathcal{P}_{\text{ltsenc}}(\Sigma) \leq^{SS} !\underline{\mathcal{F}_{\text{ke}}}$$

for any authenticated encryption scheme Σ. As explained in Section 4.1, in this realization SIDs are embedded into plaintexts before encryption with long-term keys. This is why $\mathcal{P}_{\text{anssk'}}$ realizes $\mathcal{F}_{\text{ke}}$ also for multiple concurrent sessions, even if the nonce Na is dropped. 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.

We note that one could prove Theorem 11 directly for the case that $\mathcal{P}_{\text{anssk'}}$ 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 5, 9 and 10).

Remarks on Unauthenticated Encryption. We note that the ANSSK' protocol using unauthenticated encryption does not realize $\mathcal{F}_{\text{ke}}$, i.e., $\mathcal{P}_{\text{anssk'}} \mid \mathcal{F}_{\text{enc}}^{\text{unauth}} \not\leq^{SS} \mathcal{F}_{\text{ke}}$. An attack, which can directly be formulated as a distinguishing environment, is the following: Since encryption is not authenticated, we have to assume that for the considered encryption scheme the adversary can produce ciphertexts that decrypt, e.g. under Kas, to any plaintext she likes (for $\mathcal{F}_{\text{ltsenc}}^{\text{unauth}}$ it is clear that she has this ability). Because the nonce Na is not secret, the adversary can fake the message sent from the server to A and send some message that decrypts to (Na, B, k, c) under Kas where k and c are chosen as she likes. Then, A accepts with k as a the session key exchanged with B. The same flaw exists in the original ANSSK protocol.

7.2 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, in [37, 36] Roy et al. 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. [10, 7]. 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 [26] 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 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 3, the proofs of these theorems now are very simple, they do not require to reason about IND-CCA, IND-CPA, or INT-CTXT games. While Roy et al. consider protocols which may use only symmetric encryption, our theorems immediately extend to protocols that in addition use public key encryption. These theorems can, for example, be applied to the protocol discussed in Section 7.1 and also the original ANSSK protocol.

Let $\mathcal{P}$ be a protocol system that relies solely on public key and symmetric encryption, uses $\mathcal{P}_{\text{enc}}$ for this purpose, and specifies an unbounded number of sessions of a key exchange protocol. For key indistinguishability, we assume that the session key is never used as a key in the protocol. Therefore, this key does not have to be handled by $\mathcal{P}_{\text{enc}}$; it is simply a bit string generated within (a run of) $\mathcal{P}$ itself. For key usability the session key might have been used as a key in the protocol, therefore, this key is handled as a short-term key by $\mathcal{P}_{\text{enc}}$. For ciphertext integrity it might be either of the above variants.

Definition 10. 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 $\mathcal{P} \mid \mathcal{F}_{\text{enc}}$, it holds with overwhelming probability that the session key in some uncorrupted session picked by $\mathcal{E}$ is always marked *unknown* in $\mathcal{F}_{\text{enc}}$ or, for key indistinguishability (where the session key is not a short-term key in $\mathcal{F}_{\text{enc}}$), has never been encrypted by a corrupted instance of $\mathcal{F}_{\text{ltsenc}}$ or $\mathcal{F}_{\text{pke}}$, or keys marked *known* in $\mathcal{F}_{\text{enc}}$.

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 $\mathcal{P}_{\text{enc}}$, they can rather be modeled as bit strings outside of $\mathcal{P}_{\text{enc}}$, generated within (a run of) $\mathcal{P}$ 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 $\mathcal{P} \mid \mathcal{P}_{\text{enc}}$ 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 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 $\mathcal{F}_{\text{enc}}$, 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 12. *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 $\mathcal{P}_{\text{enc}}^{\text{unauth}}$ and on $\mathcal{P}_{\text{enc}}^{\text{auth}}$ for encryption and decryption, i.e., for both IND-CCA secure and authenticated encryption.*

Proof. Let $\mathcal{E}$ be an environmental system as described for key indistinguishability. Since $\mathcal{P}_{\text{enc}}$ is a realization of $\mathcal{F}_{\text{enc}}$, by Theorem 2, Corollary 3, and the transitivity of $\leq^{SS}$ we know that there exists a simulator $\mathcal{S}$ such that $(*)$: $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{P}_{\text{enc}} \equiv \mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}$. Since $\mathcal{P}$ is secretive and $\mathcal{E} \,|\, \mathcal{S}$ can be considered to be an environment for $\mathcal{P} \,|\, \mathcal{F}_{\text{enc}}$, it follows that in runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}$, the session key, say k, output by the party picked by $\mathcal{A}$ has never been encrypted by a corrupted instance of $\mathcal{F}_{\text{pke}}$ or $\mathcal{F}_{\text{ltsenc}}$, or keys marked *known* in $\mathcal{F}_{\text{senc}}$. Moreover, by assumption, we know that k is not used to encrypt other messages. But then, from the definition of $\mathcal{F}_{\text{enc}}$ it follows that k has always been encrypted ideally. Hence, the view of $\mathcal{A}$ in runs of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}$ is independent of the value of k. Consequently, the probability that $\mathcal{T}$ outputs 1 is exactly 1/2. By $(*)$, the probability that $\mathcal{T}$ outputs 1 in a run of $\mathcal{E} \,|\, \mathcal{P} \,|\, \mathcal{P}_{\text{enc}}$ is bounded above 1/2 by a negligible function, as desired. □

Key Usability. Let $\mathcal{P}$ be a protocol as above that uses $\mathcal{P}_{\text{enc}}$ 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 handled as short-term keys within $\mathcal{P}_{\text{enc}}$.

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 negligible chance of winning the game. We note that in our setting we may w.l.o.g. assume that encryption and decryption in the IND-CCA game is performed by invoking $\mathcal{P}_{\text{enc}}$.

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 $\mathcal{P}' \,|\, \mathcal{P}_{\text{enc}}$ consisting of two subsystems. One subsystem, call it $\mathcal{A}$, interacts with $\mathcal{P}' \,|\, \mathcal{P}_{\text{enc}}$ (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 message 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 (see Section 5.1 about handling of uninterpreted messages 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 [26]. We obtain the following analog of the theorem by Roy et al. on key usability. In our theorem, we assume that a protocol does not use the 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 stick to this simpler assumption.

Theorem 13. *Let $\mathcal{P}$ be a secretive protocol where the session keys are never used to encrypt other keys. Then, $\mathcal{P}$ satisfies key usability. This holds both in case $\mathcal{P}$ relies on $\mathcal{P}_{\text{enc}}^{\text{unauth}}$ and in case $\mathcal{P}$ relies on $\mathcal{P}_{\text{enc}}^{\text{auth}}$ 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 usability. First we note that $\mathcal{E} \,|\, \mathcal{P}'$ is non-committing and used order respecting: Since $\mathcal{P}$ is non-committing and used order respecting this is the case in the first phase (where $\mathcal{A}$ interacts with $\mathcal{P}$). In the second phase, this follows from our assumption that session keys are never used to encrypt other keys. Upon encryption of a message m all short-term keys possibly contained in m are marked known in $\mathcal{F}_{\text{senc}}$ because m is encrypted uninterpreted and therefore all keys are inserted into $\mathcal{F}_{\text{senc}}$ by the store command which marks them known (see Section 5.1 about handling of uninterpreted messages). Hence, $(*)$ for all message ciphertext pairs stored in $\mathcal{F}_{\text{senc}}$ the messages only contain known keys (if any). For the first phase this is guaranteed by $\mathcal{P}$ because we assume that session keys are never used to encrypt other keys. In particular, this implies that the used order is always respected. Recall that for the used order only unknown keys are considered. Furthermore, the only case where the non-committing property might be violated in the second phase is upon decryption where the resulting plaintext contains some unknown key. In this case, the reveal command (because we decrypt uninterpreted) would mark an unknown key known. But this case can never occur because, by $(*)$, all keys upon decryption are already marked known. We conclude that $\mathcal{P}'$ is non-committing and used order respecting.

Since $\mathcal{P}_{\text{enc}}$ is a realization of $\mathcal{F}_{\text{enc}}$, by Theorem 2, Corollary 3, and the transitivity of $\leq^{SS}$ we know that there exists a simulator $\mathcal{S}$ such that $(**)$: $\mathcal{E} \,|\, \mathcal{P}' \,|\, \mathcal{P}_{\text{enc}} \equiv \mathcal{E} \,|\, \mathcal{P}' \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}$.

Since $\mathcal{P}$ is secretive and $\mathcal{E} \,|\, \mathcal{S}$ can be considered to be an environment for $\mathcal{P}' \,|\, \mathcal{F}_{\text{enc}}$, it follows by the definition of $\mathcal{P}'$ that in the first phase in runs of $\mathcal{E} \,|\, \mathcal{P}' \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}$, 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 request 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 $(**)$, the probability that $\mathcal{T}$ outputs 1 in a run of $\mathcal{E} \,|\, \mathcal{P}' \,|\, \mathcal{P}_{\text{enc}}$ is bounded above 1/2 by a negligible function, as desired. □

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

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 $\mathcal{P} \,|\, \mathcal{P}_{\text{enc}}$ 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 $\mathcal{P}_{\text{int}}$ 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 $\mathcal{E} \,|\, \mathcal{P}_{\text{int}} \,|\, \mathcal{P}_{\text{enc}}$ the bit 1 is output is negligible in the security parameter.

Theorem 14. *Let $\mathcal{P}$ be a secretive protocol. Then, $\mathcal{P}$ satisfies ciphertext integrity under session keys. This holds in case $\mathcal{P}$ relies on $\mathcal{P}_{\text{enc}}^{\text{auth}}$ 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 $\mathcal{P}_{\text{int}}$.

Now, since $\mathcal{P}_{\text{enc}}$ is a realization of $\mathcal{F}_{\text{enc}}$, by Theorem 2, Corollary 3, and the transitivity of $\leq^{SS}$ we know that there exists a simulator $\mathcal{S}$ such that $(*)$: $\mathcal{E} \,|\, \mathcal{P}_{\text{int}} \,|\, \mathcal{P}_{\text{enc}} \equiv \mathcal{E} \,|\, \mathcal{P}_{\text{int}} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$.

Since $\mathcal{P}_{\text{int}}$ is secretive and $\mathcal{E} \,|\, \mathcal{S}$ can be considered to be an environment for $\mathcal{P}_{\text{int}} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$, it follows that in runs of $\mathcal{E} \,|\, \mathcal{P}_{\text{int}} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$, the session key picked by $\mathcal{E}$ is always marked "unknown" in $\mathcal{F}_{\text{enc}}^{\text{auth}}$. But then, from the definition of $\mathcal{F}_{\text{enc}}^{\text{auth}}$ 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 $\mathcal{P}_{\text{int}} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{enc}}^{\text{auth}}$. By $(*)$, the probability that $\mathcal{E}$ outputs 1 in a run with $\mathcal{P}_{\text{int}} \,|\, \mathcal{P}_{\text{enc}}^{\text{auth}}$ is negligible, as desired. □

Acknowledgment

The authors would like to thank Ran Canetti for helpful discussions on early versions of our functionalities.

References

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In J. van Leeuwen, O. Watanabe, M. Hagiya, P.D. Mosses, and T. 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, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional Reactive Simulatability. *International Journal of Information Security (IJIS)*, 7(2):155–169, April 2008.

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

[4] M. Backes and B. Pfitzmann. Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library. In *17th IEEE Computer Security Foundations Workshop (CSFW-17 2004)*, pages 204–218. IEEE Computer Society, 2004.

[5] M. Backes and B. Pfitzmann. On the Cryptographic Key Secrecy of the Strengthened Yahalom Protocol. In S. Fischer-Hübner, K. Rannenberg, L. Yngström, and S. Lindskog, editors, *SEC*, volume 201 of *IFIP*, pages 233–245. Springer, 2006.

[6] 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.

[7] 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.

[8] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In H. Krawczyk, editor, *Advances in Cryptology, 18th Annual International Cryptology Conference (CRYPTO 1998)*, volume 1462 of *Lecture Notes in Computer Science*, pages 549–570. Springer, 1998.

[9] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[10] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 57–66. ACM, 1995.

[11] 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.

[12] 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.

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

[14] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, 2001.

[15] R. Canetti. Universally Composable Signature, Certification, and Authentication. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW-17 2004)*, pages 219–233. IEEE Computer Society, 2004.

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

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

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

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

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

[21] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002.

[22] R. Canetti and T. Rabin. Universal Composition with Joint State. In *Advances in Cryptology, 23rd Annual International Cryptology Conference (CRYPTO 2003), Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

[23] 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.

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

[25] V. Cortier, R. Küsters, and B. Warinschi. A Cryptographic Model for Branching Time Security Properties – The Case of Contract Signing Protocols. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium On Research In Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2007.

[26] 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.

[27] Anupam Datta, Ante Derek, John C. Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2006.

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

[29] 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.

[30] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. Technical Report 2006/151, Cryptology ePrint Archive, 2006. Available at `http://eprint.iacr.org/`.

[31] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. Technical Report 2008/006, Cryptology ePrint Archive, April 2008. Available at `http://eprint.iacr.org/`.

[32] 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.

[33] 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.

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

[35] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *IEEE Symposium on Security and Privacy*, pages 184–201. IEEE Computer Society Press, 2001.

[36] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Proofs of Computational Secrecy. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium On Research In Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2007.

[37] A. Roy, A. Datta, A. Derek, and J. C. Mitchell. Inductive Trace Properties Imply Computational Security. In *Proceedings of the 7th International Workshop on Issues in the Theory of Security (WITS'07)*, 2007.

A Proof of Lemma 8

Lemma 8 (restated from Section 5.2.4)**.** *There exist negligible functions $f_{\mathsf{real}}, f_{\mathsf{ideal}}$ such that*

$$\mathcal{C}^{(r)} \equiv_{f_{\mathsf{real}}} \widehat{\mathcal{C}}^{(r)}_{\mathsf{real}} \quad \textit{for all } r \in \mathbb{N} \textit{ and} \tag{6}$$

$$\mathcal{C}^{(r+1)} \equiv_{f_{\mathsf{ideal}}} \widehat{\mathcal{C}}^{(r)}_{\mathsf{auth}} \quad \textit{for all } r \in \mathbb{N}\ . \tag{7}$$

Before proving (6) and (7) separately, we establish a negligible error set, i.e., a negligible set of runs we do not want to consider. Let $B^{(r),r'}_{\text{guess}}(1^\eta, a)$ for all $r, r' \in \mathbb{N}$ be the set of runs of $\mathcal{C}^{(r)}(1^\eta, a)$ where $\mathcal{E}$ "guesses" the r'-th key. More formally: Where at some point during the run:

1. $\mathcal{E}$ wants to use the r'-th key as a corrupted key, i.e., where, for some p, k, $\mathcal{E}$ sends a message of shape $(p, \mathsf{KeyGen}, \mathsf{true}, k)$ to the simulator where $\mathcal{F}^{(r)}_{\text{senc}}.used(k) = r'$ [3],
2. $\mathcal{E}$ wants to store the r'-th key, i.e., where $\mathcal{E}$ sends a message of shape (p, Store, k) for some p, k to $\mathcal{F}^{(r)}_{\text{senc}}$ where $\mathcal{F}^{(r)}_{\text{senc}}.used(k) = r'$,
3. $\mathcal{E}$ decrypts a ciphertext with a known short-term key or a corrupted long-term or public key and where the decryption contains the r'-th key, i.e., where
   (a) $\mathcal{E}$ sends a message of shape $(p, \mathsf{Dec}, ptr, c)$ for some p, ptr, c to $\mathcal{F}^{(r)}_{\text{senc}}$ where the key $k = \mathcal{F}^{(r)}_{\text{senc}}.key(p, ptr) \in \mathcal{F}^{(r)}_{\text{senc}}.\mathcal{K}_{\text{known}}$ and $\text{dec}(k, c)$ contains (Key, k') with $\mathcal{F}^{(r)}_{\text{senc}}.used(k') = r'$,
   (b) $\mathcal{E}$ sends a message of shape $(pids, p, \mathsf{Dec}, c)$ for some $pids, p, c$ to $\mathcal{F}^{(r)}_{\text{senc}}$ where it holds that $\mathcal{F}_{\text{ltsenc}}[pids].corrupted = \mathsf{true}$ and the result of $\mathcal{F}_{\text{ltsenc}}[pids].dec(p)$ applied to c contains (Key, k) with $\mathcal{F}^{(r)}_{\text{senc}}.used(k) = r'$ [4], or

[3] By $\mathcal{F}^{(r)}_{\text{senc}}.x$ we denote the value of the variable x in the configuration of $\mathcal{F}^{(r)}_{\text{senc}}$.

[4] By $\mathcal{F}_{\text{ltsenc}}[pids].x$ we denote the value of the variable x in the configuration of the instance of $\mathcal{F}_{\text{ltsenc}}$ which accepts messages of the shape $(pids, m)$ for any m.

(c) $\mathcal{E}$ sends a message of shape (p, Dec, c) for some p, c to $\mathcal{F}_{\text{senc}}^{(r)}$ where $\underline{\mathcal{F}_{\text{pke}}}[p].corrupted = \mathsf{true}$ and the result of $\underline{\mathcal{F}_{\text{pke}}}[p].dec$ applied to c contains (Key, k) with $\mathcal{F}_{\text{senc}}^{(r)}.used(k) = r'$. [5]

Later, we only need that these error sets are negligible for any r' and $r = r'$ and $r = r' + 1$ but we prove it more general.

Recall that we abbreviate $p_{\mathcal{E}} = p_{\mathcal{E}}(\eta + |a|)$ and $f = f(1^\eta, a)$ for negligible functions f.

Lemma 10. *There exists a negligible function f_{guess} such that for all $r, r' \in \mathbb{N}$ with $r' \leq r$ and for all $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$*

$$\Pr\left[B_{\text{guess}}^{(r),r'}(1^\eta, a)\right] \leq f_{\text{guess}}(1^\eta, a) \ .$$

Proof. The proof is a standard cryptographic reduction. We construct an INT-CTXT adversary who is successful, i.e. wins the INT-CTXT game, if the event occurs.

Let $r, r' \in \mathbb{N}$ with $r' \leq r$. First, look at when the event $B_{\text{guess}}^{(r),r'}$ occurs. It occurs when $\mathcal{E}$ *chooses* (or *inserts*) a key that collides with the r'-th key. By $\mathcal{E}$ chooses we mean that $\mathcal{E}$ sends messages of the shape defined in 1., 2. and 3. in the definition of $B_{\text{guess}}^{(r),r'}$. For example if $\mathcal{E}$ sends $(p, \mathsf{KeyGen}, \mathsf{true}, k)$ for some p then $\mathcal{E}$ chooses k. For another example if $\mathcal{E}$ sends $(p, \mathsf{Dec}, ptr, c)$ for some p, ptr, c where $\text{dec}(key(p, ptr), c)$ contains (Key, k_0) and (Key, k_1) then $\mathcal{E}$ chooses k_0 and k_1. Given a run of $\mathcal{C}^{(r)}$ we order the keys chosen by $\mathcal{E}$ as they appear and call the j-th key k_j. Note that this order is different from the used order.

For all $r, r' \in \mathbb{N}$ with $r' \leq r$ we construct an INT-CTXT adversary $A_{r,r'}^{O_1(\cdot),O_2(\cdot)}$ that behaves as follows. First $A_{r,r'}$ chooses $j \in \{0, \ldots, p_{\mathcal{E}}(\eta + |a|) - 1\}$ uniformly at random and then simulates $\mathcal{C}^{(r)}(1^\eta, a)$ by using O_1 and O_2 for encryptions and decryptions, respectively, with the r'-th key. Recall that the r'-th key is used non-ideally if $r' = r$ and used ideally if $r' < r$. The adversary stops the simulation when $\mathcal{E}$ chooses the j-th key (as defined above). The adversary assumes that this key is the key used by O_1/O_2 and challenges the game by computing $c^* \leftarrow \text{enc}(k_j, m^*)$ for some m^* that was never encrypted before and making a query $O_2(c^*)$. The adversary then stops. Recall that the adversary wins the INT-CTXT game if at some point he queried O_2 with a ciphertext that decrypts but was never returned by O_1.

If the event $B_{\text{guess}}^{(r),r'}$ occurs then with probability $1/p_{\mathcal{E}}(\eta + |a|)$ the adversary aborts the simulation at the right place, i.e., where the event occurs. Up to that point the simulation was perfect because the r'-th key was never encrypted non-ideally (by Lemma 5, $\mathcal{F}^*$ guarantees that the r'-th key can only be encrypted by unknown keys with used order $< r$ and these keys are treated ideally in $\mathcal{C}^{(r)}$) and the leakage leaks exactly the length of a message. Hence, the key k_j is in fact the key used by O_1/O_2 and $A_{r,r'}$ wins because c^* was never returned by O_1 as m^* was never requested to O_1. We obtain that

$$\Pr\left[B_{\text{guess}}^{(r),r'}(1^\eta, a)\right] \leq p_{\mathcal{E}}(\eta + |a|) \cdot \text{Adv}_{\Sigma, A_{r,r'}}^{\text{int-ctxt}}(1^\eta, a) \ . \tag{14}$$

Finally, since we are interested in a bound independent of r, we construct an adversary A which first chooses $r' \in \{0, \ldots, p_{\mathcal{E}}(\eta + |a|) - 1\}$ and $r \in \{r', \ldots, p_{\mathcal{E}}(\eta + |a|) - 1\}$ uniformly at random and then behaves like $A_{r,r'}$. Clearly,

$$p_{\mathcal{E}}(\eta + |a|) \cdot p_{\mathcal{E}}(\eta + |a|) \cdot \text{Adv}_{\Sigma, A}^{\text{int-ctxt}}(1^\eta, a) \geq \text{Adv}_{\Sigma, A_{r,r'}}^{\text{int-ctxt}}(1^\eta, a) \tag{15}$$

for all $r, r' < p_{\mathcal{E}}(\eta + |a|)$ with $r' \leq r$. For $r \geq p_{\mathcal{E}}(\eta + |a|)$ the system $\mathcal{C}^{(r)}$ behaves exactly like $\mathcal{C}^{(p_{\mathcal{E}}(\eta + |a|) - 1)}$ and hence (15) holds for all $r, r' \in \mathbb{N}$ with $r' \leq r$.

Since Σ is INT-CTXT secure $\text{Adv}_{\Sigma, A}^{\text{int-ctxt}}(1^\eta, a)$ is negligible and by (14) and (15) we conclude that there exists a negligible function f_{guess} such that $\Pr\left[B_{\text{guess}}^{(r),r'}(1^\eta, a)\right] \leq f_{\text{guess}}(1^\eta, a)$ for all $r, r' \in \mathbb{N}$ with $r' \leq r$, $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$. □

A.1 Proof of (6)

For all r, η, a we define a relation $R = R_{r,\eta,a}$ between runs of $\mathcal{C}^{(r)}(1^\eta, a)$ and runs of $\widehat{\mathcal{C}}_{\mathsf{real}}^{(r)}(1^\eta, a)$.

[5] By $\underline{\mathcal{F}_{\text{pke}}}[p].x$ we denote the value of the variable x in the configuration of the instance of $\underline{\mathcal{F}_{\text{pke}}}$ which accepts messages of the shape (p, m) for any m.

First we describe R informally. In runs of the system $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ we have two r-th keys, the one in $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ which we denote by the *payload key* and the one in Oracle(real) which we denote by the *encryption key*. The differences between the two systems $\mathcal{C}^{(r)}$ and $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ is not on how the keys are used. The r-th key in $\mathcal{C}^{(r)}$ and the encryption key in $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ are both used non-ideally. Also, the r-th key is never encrypted non-ideally (because of $\mathcal{F}^*$) and therefore it does not matter if the payload key is different from the encryption key. The only difference between the systems is if the environment guesses the r-th key because in $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ there is this difference between the payload and the encryption key. For the same reason collisions with the r-th key result in different behavior of the systems. Fortunately, we can push these events into negligible error sets, namely $B^{(r)}_{\text{coll}}$ (see Lemma 6) and $B^{(r),r}_{\text{guess}}$ (see Lemma 10).

The only difference between the systems to deal with is the moment when the r-th key is generated. When in $\mathcal{C}^{(r)}$ the r-th key is generated then in $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ the payload key is generated and when in $\mathcal{C}^{(r)}$ the r-th key is first used (at that moment it gets assigned the used order r) then in $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ the encryption key is generated and first used. The payload key is never really used, not even as a payload, because it is always encrypted ideally (this follows from the definition of $\mathcal{F}^*$). Therefore, we can relate a run ρ of $\mathcal{C}^{(r)}$ to several runs $\widehat{\rho}_k$ of $\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ for all possible payload keys k, i.e., the r-th key in ρ equals the encryption key in $\widehat{\rho}_k$ and the payload key in $\widehat{\rho}_k$ (which does not occur in ρ) is k. Then we can prove that

$$\Pr[\rho] - \Pr[\{\widehat{\rho}_k \mid k \text{ possible payload key}\}] \leq \Pr[\rho] \cdot f$$

for some negligible function f (which is independent of r). By Theorem 4 we can conclude (6).

We now define the relation R more formal. Let η, a be fixed and, to simplify notation, let $\mathcal{C} = \mathcal{C}^{(r)}$, $\widehat{\mathcal{C}} = \widehat{\mathcal{C}}^{(r)}_{\mathsf{real}}$ and $B = B^{(r),r}_{\text{guess}}(1^\eta, a) \cup B^{(r)}_{\text{coll}}(1^\eta, a)$.

In a run of the system $\widehat{\mathcal{C}}$, if $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ sends a message to Oracle(real) and directly receives an answer then in the run this results in three configurations which correspond to a single configuration in a run of $\mathcal{C}$. For simplification, we consider $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ and Oracle(real) to be a single IITM. This is valid as shown in [29].

For $\rho \in \text{runs}^a_\eta(\mathcal{C}) \setminus B$ let $\mathcal{K}_\rho$ be the set of all keys that occur in ρ, i.e. $\mathcal{K}_\rho = \mathcal{F}^{(r)}_{\text{senc}}.\mathcal{K}$ in the last configuration of ρ because $\mathcal{K}$ contains all keys and is monotone, i.e., no keys get deleted. We define $\overline{\mathcal{K}}_\rho = \{0,1\}^{\text{key-len}_\eta} \setminus \mathcal{K}_\rho$ to be the set of keys that can be generated by $\text{gen}(1^\eta)$ and which do not occur in ρ. Recall that we assume that all keys generated by $\text{gen}(1^\eta)$ have length key-len_η.

For all keys $k^* \in \overline{\mathcal{K}}_\rho$ (this corresponds to the payload key in the informal description) we define $\alpha_{k^*} \colon \text{runs}^a_\eta(\mathcal{C}) \setminus B \to \text{runs}^a_\eta(\widehat{\mathcal{C}})$. Let $\rho = C_0, \ldots, C_n \in \text{runs}^a_\eta(\mathcal{C}) \setminus B$ we define $\alpha_{k^*}(\rho)$ as follows:

1. If there exists no r-th key, i.e. no $\overline{k} \in \{0,1\}^*$ and $\overline{m} \leq n$ such that $\mathcal{F}^{(r)}_{\text{senc}}.\mathit{used}(\overline{k}) = r$ in $C_{\overline{m}}$ then we define $\alpha_{k^*}(\rho) = \widehat{C}_0, \ldots, \widehat{C}_n$ where for all $m \leq n$, $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}^{(r)}_{\text{senc}}$ by the configuration of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}} \mid \text{Oracle}(\mathsf{real})$ (recall that we consider $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ and Oracle(real) to be a single IITM) where Oracle is uninitialized and the state (i.e., the value of all variables) of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ is equal to the state of $\mathcal{F}^{(r)}_{\text{senc}}$.

   Note that in this case we have not used the payload key k^* because there is no r-th key.

2. If there exists an r-th key, i.e. we find $\overline{k} \in \{0,1\}^*$ and $\overline{m} \leq n$ such that $\mathcal{F}^{(r)}_{\text{senc}}.\mathit{used}(\overline{k}) = r$ in $C_{\overline{m}}$ then let $\overline{m}$ be the smallest such value (i.e., in configuration $C_{\overline{m}}$ the r-th key is first used).

   We define $\alpha_{k^*}(\rho) = \widehat{C}_0, \ldots, \widehat{C}_n$ as follows:

   1. For all $m < \overline{m}$, $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}^{(r)}_{\text{senc}}$ by the configuration of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}} \mid \text{Oracle}(\mathsf{real})$ where there is no running instance of Oracle and the state of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ is equal to the state of $\mathcal{F}^{(r)}_{\text{senc}}$ except that the key $\overline{k}$ (the encryption key) is replaced by k^* (the payload key). By replaced we mean that it is replaced everywhere, i.e. in $\mathcal{K}$, *decTable*, *key* and *used*.
   2. For all $\overline{m} \leq m \leq n$, $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}^{(r)}_{\text{senc}}$ by the configuration of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}} \mid \text{Oracle}(\mathsf{real})$ where there is a running instance of Oracle with $\mathit{state} = \mathsf{ok}$, $k = \overline{k}$ (i.e. Oracle uses the encryption key) and $\mathit{decTable} = \emptyset$ and the state of $\widehat{\mathcal{F}}^{(r)}_{\text{senc}}$ is equal to the state of $\mathcal{F}^{(r)}_{\text{senc}}$ except that the key $\overline{k}$ (the encryption key) is replaced by k^* (the payload key).

Clearly, $\alpha_{k^*}(\rho) \in \mathrm{runs}^a_\eta(\widehat{\mathcal{C}})$.

Now we define R. A run $\rho \in \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$ is in relation to $\alpha_{k^*}(\rho)$ for all $k^* \in \overline{\mathcal{K}}_\rho$, i.e., $(\rho, \widehat{\rho}) \in R$ if and only if $\widehat{\rho} = \alpha_{k^*}(\rho)$ for some $k^* \in \overline{\mathcal{K}}_\rho$. Clearly, $\mathrm{dom}(R) = \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$, i.e, $\Pr[\overline{\mathrm{dom}(R)}] = \Pr[B]$. The equivalence classes of $\sim_R$ are $\{\rho\} \cup \{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}$ for all $\rho \in \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$.

Finally, we prove that there exists a negligible function f such that

$$0 \leq \Pr[\rho] - \Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] \leq f(1^\eta, a) \cdot \Pr[\rho] \tag{16}$$

for all $\rho \in \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$. Let

$$f(1^\eta, a) = \max\{\Pr[k \leftarrow \mathrm{gen}(1^\eta) : k \in \mathcal{K}] \mid \mathcal{K} \subseteq \{0,1\}^*, |\mathcal{K}| \leq p_{\mathcal{E}}(\eta + |a|)\} \ .$$

Similar to the proof of Lemma 6 it can be shown that f is negligible because Σ is IND-CPA secure and $\mathcal{K}$ has polynomial size. Note that f does not depend on r.

Let $\rho = C_0, \ldots, C_n \in \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$. If there exists no r-th key in ρ (as in 1. in the definition of α_{k^*}) then one easily verifies that $\Pr[\rho] = \Pr[\alpha_{k^*}(\rho)]$ for any k^*. Also, we have that $\alpha_{k_1^*}(\rho) = \alpha_{k_2^*}(\rho)$ for any k_1^*, k_2^* (because k^* is never used in that case). Hence, $\Pr[\rho] = \Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}]$ in this case. If there exists an r-th key then let $\overline{k}$ and $\overline{m}$ be defined as in 2. in the definition of α_{k^*}. Recall that $\overline{m}$ is the (index of the) configuration where $\overline{k}$ is first used. Let $\overline{m}'$ be the configuration where the key $\overline{k}$ is generated, i.e., where the simulator generates $\overline{k}$ by computing $\mathrm{gen}(1^\eta)$ and sends it to $\mathcal{F}^{(r)}_{\mathrm{senc}}$. For every $k^* \in \overline{\mathcal{K}}_\rho$ let $\alpha_{k^*}(\rho) = C_0^{k^*}, \ldots, C_n^{k^*}$.

Recall that we assume that Oracle and $\widehat{\mathcal{F}}^{(r)}_{\mathrm{senc}}$ are considered as a single IITM. By definition of the probability of a run we have $\Pr[\rho] = \prod_{0<m\leq n} \Pr[C_{m-1} \to C_m]$ and $\Pr[\alpha_{k^*}(\rho)] = \prod_{0<m\leq n} \Pr[C^{k^*}_{m-1} \to C^{k^*}_m]$. Now, we relate these probabilities. The crucial point is that because $\rho \notin B$ and $k^* \notin \overline{\mathcal{K}}_\rho$ there is no difference upon guessing or collisions of unknown keys (i.e., in "KEY GENERATION", "STORE", and "PREVENT KEY GUESSING"). Furthermore, if party p decrypts c with pointer ptr where $key(p, ptr) = \overline{k}$ and $used(\overline{k}) = \bot$ then it is important to verify that the behavior is the same. Note that because $used(\overline{k}) = \bot$ we have that $nextused \leq r$, hence, $\mathcal{F}^{(r)}_{\mathrm{senc}}$ and $\widehat{\mathcal{F}}^{(r)}_{\mathrm{senc}}$ will both return $\bot$ (because $decTable(\overline{k}) = \emptyset$). Since the distribution of $L(1^\eta, \overline{k})$ equals the one of $L(1^\eta, k^*)$ (L leaks exactly the length of a message and $\overline{k}$ and k^* have the same length) and by Lemma 5 (the r-th key is never encrypted non-ideally), we conclude that the behavior upon encryption and decryption is always the same, i.e., we have that

$$\Pr\left[C_{m-1} \to C_m\right] = \Pr\left[C^{k^*}_{m-1} \to C^{k^*}_m\right] \qquad \text{for all } m \in \{1, \ldots, n\} \setminus \{\overline{m}', \overline{m}\} \ .$$

The runs differ at the transitions from $\overline{m}' - 1$ to $\overline{m}'$ and from $\overline{m} - 1$ to $\overline{m}$ because in $C_{\overline{m}'}$ the key $\overline{k}$ is generated while in $C^{k^*}_{\overline{m}'}$ the payload key k^* is generated. In $C_{\overline{m}}$ the key $\overline{k}$ is first used and no keys are generated while in $C^{k^*}_{\overline{m}}$ the encryption key $\overline{k}$ is generated and then used. Hence,

$$\begin{aligned}
\Pr\left[C^{k^*}_{\overline{m}'-1} \to C^{k^*}_{\overline{m}'}\right] &= \Pr\left[C_{\overline{m}'-1} \to C_{\overline{m}'}\right] \cdot \frac{\Pr[\mathrm{gen}(1^\eta) = k^*]}{\Pr[\mathrm{gen}(1^\eta) = \overline{k}]} \ , \\
\Pr\left[C^{k^*}_{\overline{m}-1} \to C^{k^*}_{\overline{m}}\right] &= \Pr\left[C_{\overline{m}-1} \to C_{\overline{m}}\right] \cdot \Pr[\mathrm{gen}(1^\eta) = \overline{k}] \ .
\end{aligned}$$

Therefore, for all $k^* \in \overline{\mathcal{K}}_\rho$ it holds

$$\begin{aligned}
\Pr[\alpha_{k^*}(\rho)] &= \prod_{0<m\leq n} \Pr[C^{k^*}_{m-1} \to C^{k^*}_m] \\
&= \Pr\left[C_{\overline{m}'-1} \to C_{\overline{m}'}\right] \cdot \frac{\Pr[\mathrm{gen}(1^\eta) = k^*]}{\Pr[\mathrm{gen}(1^\eta) = \overline{k}]} \cdot \Pr\left[C_{\overline{m}-1} \to C_{\overline{m}}\right] \cdot \Pr[\mathrm{gen}(1^\eta) = \overline{k}] \\
&\quad \cdot \prod_{m \in \{1,\ldots,n\} \setminus \{\overline{m}', \overline{m}\}} \Pr\left[C_{m-1} \to C_m\right] \\
&= \Pr[\mathrm{gen}(1^\eta) = k^*] \cdot \Pr[\rho] \ .
\end{aligned}$$

We conclude

$$\Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] = \sum_{k^* \in \overline{\mathcal{K}}_\rho} \Pr[\alpha_{k^*}(\rho)] = \Pr[\mathrm{gen}(1^\eta) \in \overline{\mathcal{K}}_\rho] \cdot \Pr[\rho] \leq \Pr[\rho] \ .$$

and because $|\mathcal{K}_\rho| \leq p_{\mathcal{E}}(\eta + |a|)$ we have that

$$\Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] = (1 - \Pr[\mathrm{gen}(1^\eta) \in \mathcal{K}_\rho]) \cdot \Pr[\rho] \geq \Pr[\rho] - f(1^\eta, a) \cdot \Pr[\rho]$$

from which we obtain (16).

By Lemma 6 and 10, we find a negligible f' such that for all $r, \eta \in \mathbb{N}$ and $a \in \{0,1\}^*$ we have $f'(1^\eta, a) \geq \Pr[B^{(r),r}_{\mathrm{guess}}(1^\eta, a) \text{ or } B^{(r)}_{\mathrm{coll}}(1^\eta, a)] = \Pr[\overline{\mathrm{dom}(R_{r,\eta,a})}]$. Hence, for all $r, \eta \in \mathbb{N}$ and $a \in \{0,1\}^*$ the relation $R_{r,\eta,a}$ is a $(f(1^\eta, a), 0, f'(1^\eta, a))$-probabilistic trace relation between $\mathrm{runs}^a_\eta(\mathcal{C}^{(r)})$ and $\mathrm{runs}^a_\eta(\widehat{\mathcal{C}}^{(r)}_{\mathsf{real}})$. Theorem 4 implies that (6) holds with $f_{\mathrm{real}} = f + f'$. Note that since f and f' are independent of r, so is f_{real}.

A.2 Proof of (7)

For all r, η, a we define a relation $R = R_{r,\eta,a}$ between runs of $\mathcal{C}^{(r+1)}(1^\eta, a)$ and runs of $\widehat{\mathcal{C}}^{(r)}_{\mathsf{auth}}(1^\eta, a)$.

The relation R and the proof are similar to the proof of (6). However, the systems are more different because of guessing of keys. In $\mathcal{C}^{(r+1)}$ we have to consider $\mathrm{KeyGuess}^{(r+1)}$ while in $\widehat{\mathcal{C}}^{(r)}_{\mathsf{auth}}$ we have to consider $\mathrm{KeyGuess}^{(r)}$. Therefore, we need two additional error sets which capture guessing of unknown and unused keys before the r-th key has been used.

Let $B^{(r)}_{\text{guess-unused}}(1^\eta, a)$ for all $r \in \mathbb{N}$ be the set of runs of $\mathcal{C}^{(r)}(1^\eta, a)$ where $\mathcal{E}$ "guesses" an unknown and unused key before the r-th key has been used. More formally: Where at some point during the run before the r-th key has been used, i.e., where $\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{nextused} \leq r$:

1. $\mathcal{E}$ wants to use an unknown and unused key k as a corrupted key, i.e., where $\mathcal{E}$ sends a message of shape $(p, \mathsf{KeyGen}, \mathsf{true}, k)$ for some p to the simulator where $k \in \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K} \setminus \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K}_{\mathrm{known}}$ and $k \notin \mathrm{dom}(\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{used})$,
2. $\mathcal{E}$ wants to store an unknown and unused key k, i.e., where $\mathcal{E}$ sends a message of shape (p, Store, k) for some p to $\mathcal{F}^{(r)}_{\mathrm{senc}}$ where $k \in \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K} \setminus \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K}_{\mathrm{known}}$ and $k \notin \mathrm{dom}(\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{used})$, or
3. $\mathcal{E}$ decrypts a ciphertext with a known short-term key or a corrupted long-term or public key and where the decryption contains an unknown and unused key k, i.e., where $k \in \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K} \setminus \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K}_{\mathrm{known}}$ and $k \notin \mathrm{dom}(\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{used})$ and
   (a) $\mathcal{E}$ sends a message of shape $(p, \mathsf{Dec}, \mathit{ptr}, c)$ for some p, ptr, c to $\mathcal{F}^{(r)}_{\mathrm{senc}}$ where the key $k' = \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{key}(p, \mathit{ptr}) \in \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K}_{\mathrm{known}}$ and $\mathrm{dec}(k', c)$ contains (Key, k),
   (b) $\mathcal{E}$ sends a message of shape $(\mathit{pids}, p, \mathsf{Dec}, c)$ for some pids, p, c to $\mathcal{F}^{(r)}_{\mathrm{senc}}$ where it holds that $\mathcal{F}_{\mathrm{ltsenc}}[\mathit{pids}].\mathit{corrupted} = \mathsf{true}$ and the result of $\mathcal{F}_{\mathrm{ltsenc}}[\mathit{pids}].\mathit{dec}(p)$ applied to c contains (Key, k), or
   (c) $\mathcal{E}$ sends a message of shape (p, Dec, c) for some p, c to $\mathcal{F}^{(r)}_{\mathrm{senc}}$ where $\underline{\mathcal{F}_{\mathrm{pke}}}[p].\mathit{corrupted} = \mathsf{true}$ and the result of $\underline{\mathcal{F}_{\mathrm{pke}}}[p].\mathit{dec}$ applied to c contains (Key, k).

Lemma 11. *There exists a negligible function $f_{\text{guess-unused}}$ such that for all $r \in \mathbb{N}$ and for all $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$*

$$\Pr[B^{(r)}_{\text{guess-unused}}(1^\eta, a)] \leq f_{\text{guess-unused}}(1^\eta, a) \ .$$

Proof. We only sketch the proof. All keys with used order $< r$ are used ideally. Furthermore, unknown keys are generated honestly and have not been encrypted by known or corrupted keys, i.e., only by keys with used order $< r$. Because they are unused, guessing one of these keys is like guessing a key generated by $\mathrm{gen}(1^\eta)$ without having any information about the key. Because Σ is IND-CPA secure and there are only polynomially many such keys $B^{(r)}_{\text{guess-unused}}(1^\eta, a)$ is negligible. We obtain a bound independent of r with the same technique as used in the proof of Lemma 10. □

Let $B^{(r)}_{\mathrm{int}}(1^\eta, a)$ for all $r \in \mathbb{N}$ be the set of runs of $\mathcal{C}^{(r)}(1^\eta, a)$ where $\mathcal{E}$ "produces" a ciphertext that successfully decrypts (to some plaintext $\neq \bot$) under an unknown and unused key before the r-th key has been used. More formally: Where at some point during the run before the r-th key has been used, i.e., where $\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{nextused} \leq r$, $\mathcal{E}$ decrypts a ciphertext c, i.e., where $\mathcal{E}$ sends a message of shape $(p, \mathsf{Dec}, \mathit{ptr}, c)$ to $\mathcal{F}^{(r)}_{\mathrm{senc}}$ for some p, ptr, c, where $k = \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{key}(p, \mathit{ptr}) \in \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K} \setminus \mathcal{F}^{(r)}_{\mathrm{senc}}.\mathcal{K}_{\mathrm{known}}$, $k \notin \mathrm{dom}(\mathcal{F}^{(r)}_{\mathrm{senc}}.\mathit{used})$ and $\mathrm{dec}(k, c) \neq \bot$.

Lemma 12. *There exists a negligible function f_{int} such that for all $r \in \mathbb{N}$ and for all $\eta \in \mathbb{N}$ and $a \in \{0,1\}^*$*

$$\Pr[B_{\text{int}}^{(r)}(1^\eta, a)] \leq f_{\text{int}}(1^\eta, a) \ .$$

Proof. We only sketch the proof. As for $B_{\text{guess-unused}}^{(r)}$, all keys with used order $< r$ are used ideally, unknown keys are generated honestly and have not been encrypted by known or corrupted keys. Because these keys are unused, producing a ciphertext that decrypts for these keys is like producing a ciphertext for a key generated by $\text{gen}(1^\eta)$ without having any information about the key. Because Σ is INT-CTXT secure and there are only polynomially many such keys $B_{\text{int}}^{(r)}(1^\eta, a)$ is negligible. We obtain a bound independent of r with the same technique as used in the proof of 10. □

Now, we proof (7). The differences compared to the proof of (6) are the following.

1. Here, we consider the error set $B = B_{\text{coll}}^{(r+1)} \cup B_{\text{guess}}^{(r+1),r} \cup B_{\text{guess}}^{(r+1),r+1} \cup B_{\text{guess-unused}}^{(r+1)} \cup B_{\text{int}}^{(r+1)}$ instead of $B_{\text{coll}}^{(r)} \cup B_{\text{guess}}^{(r),r}$.
2. We still relate the r-th key (not the $(r+1)$-th key) of $\mathcal{C}^{(r+1)}$ to the key in Oracle(auth). This time the key is used ideally, therefore, we also have to relate the decryption tables $\mathcal{F}_{\text{senc}}^{(r+1)}.decTable(\overline{k})$ and Oracle(auth).*decTable* (where $\overline{k}$ is the r-th key).

We now define the relation R more formal. Let η, a be fixed and, to simplify notation, let $\mathcal{C} = \mathcal{C}^{(r+1)}$, $\widehat{\mathcal{C}} = \widehat{\mathcal{C}}_{\mathsf{auth}}^{(r)}$ and $B = B_{\text{coll}}^{(r+1)}(1^\eta, a) \cup B_{\text{guess}}^{(r+1),r}(1^\eta, a) \cup B_{\text{guess}}^{(r+1),r+1}(1^\eta, a) \cup B_{\text{guess-unused}}^{(r+1)}(1^\eta, a) \cup B_{\text{int}}^{(r+1)}(1^\eta, a)$. As above, for simplification, we consider $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ and Oracle(auth) to be a single IITM.

Recall the definition of $\mathcal{K}_\rho$ and $\overline{\mathcal{K}}_\rho$ from above. For all keys $k^* \in \overline{\mathcal{K}}_\rho$ (this will be the payload key) we define $\alpha_{k^*} \colon \text{runs}_\eta^a(\mathcal{C}) \setminus B \to \text{runs}_\eta^a(\widehat{\mathcal{C}})$. Let $\rho = C_0, \ldots, C_n \in \text{runs}_\eta^a(\mathcal{C}) \setminus B$ we define $\alpha_{k^*}(\rho)$ as follows:

1. If there exists no r-th key, i.e. no $\overline{k} \in \{0,1\}^*$ and $\overline{m} \leq n$ such that $\mathcal{F}_{\text{senc}}^{(r+1)}.used(\overline{k}) = r$ in $C_{\overline{m}}$ then we define $\alpha_{k^*}(\rho) = \widehat{C}_0, \ldots, \widehat{C}_n$ where for all $m \leq n$ $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}_{\text{senc}}^{(r+1)}$ by the configuration of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)} \mid$ Oracle(auth) (recall that we consider $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ and Oracle(auth) to be a single IITM) where Oracle is uninitialized and the state of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ is equal to the state of $\mathcal{F}_{\text{senc}}^{(r+1)}$.

   Note that in this case we have not used the payload key k^* because there is no r-th key.

2. If there exists an r-th key, i.e. we find $\overline{k} \in \{0,1\}^*$ and $\overline{m} \leq n$ such that $\mathcal{F}_{\text{senc}}^{(r+1)}.used(\overline{k}) = r$ in $C_{\overline{m}}$ then let $\overline{m}$ be the smallest such value (i.e., in configuration $C_{\overline{m}}$ the r-th key is first used).

   We define $\alpha_{k^*}(\rho) = \widehat{C}_0, \ldots, \widehat{C}_n$ as follows:

   1. For all $m < \overline{m}$, $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}_{\text{senc}}^{(r+1)}$ by the configuration of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)} \mid$ Oracle(auth) where there is no running instance of Oracle and the state of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ is equal to the state of $\mathcal{F}_{\text{senc}}^{(r+1)}$ except that the key $\overline{k}$ (the encryption key) is replaced by k^* (the payload key). By replaced we mean that it is replaced everywhere, i.e. in $\mathcal{K}$, *decTable*, *key* and *used*.
   2. For all $\overline{m} \leq m \leq n$, $\widehat{C}_m$ is obtained from C_m by replacing the configuration of $\mathcal{F}_{\text{senc}}^{(r+1)}$ by the configuration of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)} \mid$ Oracle(auth) where there is a running instance of Oracle with $state = \mathsf{ok}$, $k = \overline{k}$ (i.e. Oracle uses the encryption key) and $decTable = \mathcal{F}_{\text{senc}}^{(r+1)}.decTable(\overline{k})$ and the state of $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}$ is equal to the state of $\mathcal{F}_{\text{senc}}^{(r+1)}$ except that the key $\overline{k}$ (the encryption key) is replaced by k^* (the payload key) and $\widehat{\mathcal{F}}_{\text{senc}}^{(r)}.decTable(k^*) = \emptyset$. By replaced we mean that it is replaced everywhere, i.e. in $\mathcal{K}$, *decTable* and *key*.

Clearly, $\alpha_{k^*}(\rho) \in \text{runs}_\eta^a(\widehat{\mathcal{C}})$.

Now we define R. A run $\rho \in \text{runs}_\eta^a(\mathcal{C}) \setminus B$ is in relation to $\alpha_{k^*}(\rho)$ for all $k^* \in \overline{\mathcal{K}}_\rho$, i.e., $(\rho, \widehat{\rho}) \in R$ if and only if $\widehat{\rho} = \alpha_{k^*}(\rho)$ for some $k^* \in \overline{\mathcal{K}}_\rho$. Clearly, $\text{dom}(R) = \text{runs}_\eta^a(\mathcal{C}) \setminus B$, i.e, $\Pr[\overline{\text{dom}(R)}] = \Pr[B]$. The equivalence classes of $\sim_R$ are $\{\rho\} \cup \{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}$ for all $\rho \in \text{runs}_\eta^a(\mathcal{C}) \setminus B$.

Finally, we prove that there exists a negligible function f such that

$$0 \leq \Pr[\rho] - \Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] \leq f(1^\eta, a) \cdot \Pr[\rho] \ . \tag{17}$$

The proof of (17) is similar to the proof of (16), in fact, we can use the same f but we have to argue using the different error set.

Let $\rho = C_0, \ldots, C_n \in \mathrm{runs}^a_\eta(\mathcal{C}) \setminus B$. If there exists no r-th key in ρ (as in 1. in the definition of α_{k^*}) then we always have that *nextused* $\leq r$, so, there is no difference between $\mathrm{KeyGuess}^{(r+1)}$ and $\mathrm{KeyGuess}^{(r)}$. One verifies that $\Pr[\rho] = \Pr[\alpha_{k^*}(\rho)]$ for any k^*. Also, we have that $\alpha_{k_1^*}(\rho) = \alpha_{k_2^*}(\rho)$ for any k_1^*, k_2^* (because k^* is never used in that case). Hence, $\Pr[\rho] = \Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}]$ in this case. If there exists an r-th key then let $\overline{k}$ and $\overline{m}$ be defined as in 2. in the definition of α_{k^*}. Recall that $\overline{m}$ is the (index of the) configuration where $\overline{k}$ is first used. Let $\overline{m}'$ be the configuration where the key $\overline{k}$ is generated, i.e., where the simulator generates $\overline{k}$ by computing $\mathrm{gen}(1^\eta)$ and sends it to $\mathcal{F}^{(r)}_{\mathrm{senc}}$. For every $k^* \in \overline{\mathcal{K}}_\rho$ let $\alpha_{k^*}(\rho) = C_0^{k^*}, \ldots, C_n^{k^*}$.

Recall that we assume that Oracle and $\widehat{\mathcal{F}}^{(r)}_{\mathrm{senc}}$ are considered as a single IITM. By definition of the probability of a run we have $\Pr[\rho] = \prod_{0<m\leq n} \Pr[C_{m-1} \to C_m]$ and $\Pr[\alpha_{k^*}(\rho)] = \prod_{0<m\leq n} \Pr[C^{k^*}_{m-1} \to C^{k^*}_m]$. Now, we relate these probabilities. The two crucial points are the following:

1. Consider key guessing. If $\mathcal{E}$ guesses a key $k \in \mathcal{K} \setminus \mathcal{K}_{\mathrm{known}}$ then because of the difference between $\mathrm{KeyGuess}^{(r+1)}$ and $\mathrm{KeyGuess}^{(r)}$ there might be difference if *nextused* $= r+1$. We show that this is not the case. If $\bot \neq \mathit{used}(k) < r$ or $\bot \neq \mathit{used}(k) > r+1$ then there is no difference between $\mathrm{KeyGuess}^{(r+1)}$ and $\mathrm{KeyGuess}^{(r)}$. The case $\mathit{used}(k) \in \{r, r+1\}$ cannot occur because of the error sets $B^{(r+1),r}_{\mathrm{guess}}$ and $B^{(r+1),r+1}_{\mathrm{guess}}$. Hence, we only have to consider unused keys, i.e., where $k \notin \mathrm{dom}(\mathit{used})$. The only difference might occur if *nextused* $= r+1$ but this case is ruled out by $B^{(r+1)}_{\text{guess-unused}}$.
2. If party p decrypts c with pointer ptr where $\mathit{key}(p, ptr) = k$ for some $k \in \mathcal{K}\setminus\mathcal{K}_{\mathrm{known}}$ and $\mathit{used}(k) = \bot$ then it is important to verify that the behavior is the same. If *nextused* $\leq r$ both $\mathcal{F}^{(r+1)}_{\mathrm{senc}}$ and $\widehat{\mathcal{F}}^{(r)}_{\mathrm{senc}}$ treat the key as ideal and return $\bot$. If *nextused* $= r+1$ then $\mathcal{F}^{(r+1)}_{\mathrm{senc}}$ treats the key ideal and returns $\bot$ but $\widehat{\mathcal{F}}^{(r)}_{\mathrm{senc}}$ treats the key real and computes $\mathrm{dec}(k, c)$ which by $B^{(r+1)}_{\mathrm{int}}$ returns $\bot$. If *nextused* $> r+1$ then both systems treat the key real but we know that it is not the r-th key, so, they use the same key.

Since the distribution of $L(1^\eta, \overline{k})$ equals the one of $L(1^\eta, k^*)$ (L leaks exactly the length of a message and $\overline{k}$ and k^* have the same length) and by Lemma 5 (the r-th key is never encrypted non-ideally), we conclude that the behavior upon encryption and decryption is always the same, i.e., we have that

$$\Pr\left[C_{m-1} \to C_m\right] = \Pr\left[C^{k^*}_{m-1} \to C^{k^*}_m\right] \qquad \text{for all } m \in \{1, \ldots, n\} \setminus \{\overline{m}', \overline{m}\} \ .$$

As in the proof of (6), the runs differ at the transitions from $\overline{m}' - 1$ to $\overline{m}'$ and from $\overline{m} - 1$ to $\overline{m}$ because in $C_{\overline{m}'}$ the key $\overline{k}$ is generated while in $C^{k^*}_{\overline{m}'}$ the payload key k^* is generated. In $C_{\overline{m}}$ the key $\overline{k}$ is first used and no keys are generated while in $C^{k^*}_{\overline{m}}$ the encryption key $\overline{k}$ is generated and then used. Hence,

$$\begin{aligned}
\Pr\left[C^{k^*}_{\overline{m}'-1} \to C^{k^*}_{\overline{m}'}\right] &= \Pr\left[C_{\overline{m}'-1} \to C_{\overline{m}'}\right] \cdot \frac{\Pr[\mathrm{gen}(1^\eta) = k^*]}{\Pr[\mathrm{gen}(1^\eta) = \overline{k}]} \ , \\
\Pr\left[C^{k^*}_{\overline{m}-1} \to C^{k^*}_{\overline{m}}\right] &= \Pr\left[C_{\overline{m}-1} \to C_{\overline{m}}\right] \cdot \Pr[\mathrm{gen}(1^\eta) = \overline{k}] \ .
\end{aligned}$$

As for (6), we conclude

$$\begin{aligned}
&\Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] \leq \Pr[\rho] \ , \\
&\Pr[\{\alpha_{k^*}(\rho) \mid k^* \in \overline{\mathcal{K}}_\rho\}] \geq \Pr[\rho] - f(1^\eta, a) \cdot \Pr[\rho]
\end{aligned}$$

from which we obtain (17).

By Lemma 6, 10, 11 and 12, we find a negligible f' such that for all $r, \eta \in \mathbb{N}$ and $a \in \{0,1\}^*$ we have $f'(1^\eta, a) \geq \Pr[B] = \Pr[\overline{\mathrm{dom}(R_{r,\eta,a})}]$. Hence, for all $r, \eta \in \mathbb{N}$ and $a \in \{0,1\}^*$ the relation $R_{r,\eta,a}$ is a $(f(1^\eta, a), 0, f'(1^\eta, a))$-probabilistic trace relation between $\mathrm{runs}^a_\eta(\mathcal{C}^{(r+1)})$ and $\mathrm{runs}^a_\eta(\widehat{\mathcal{C}}^{(r)}_{\mathsf{auth}})$. Theorem 4 implies that (7) holds with $f_{\mathrm{ideal}} = f + f'$. Note that since f and f' are independent of r, so is f_{ideal}.