# Joint State Composition Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation*

Ralf Küsters [*] Max Tuengerthal [+] Daniel Rausch [*]

[*] University of Stuttgart, Germany

{ralf.kuesters,daniel.rausch}@sec.uni-stuttgart.de

[+] Siemens Mobility

max.tuengerthal@siemens.com

September 26, 2019

**Abstract**

In frameworks for universal composability, complex protocols can be built from sub-protocols in a modular way using composition theorems. However, as first pointed out and studied by Canetti and Rabin, this modular approach often leads to impractical implementations. For example, when using a functionality for digital signatures within a more complex protocol, parties have to generate new verification and signing keys for every session of the protocol. This motivates to generalize composition theorems to so-called joint state (composition) theorems, where different copies of a functionality may share some state, e.g., the same verification and signing keys.

In this paper, we present a joint state theorem which is more general than the original theorem of Canetti and Rabin, for which several problems and limitations are pointed out. We apply our theorem to obtain joint state realizations for three functionalities: public-key encryption, replayable public-key encryption, and digital signatures. Unlike most other formulations, our functionalities model that ciphertexts and signatures are computed locally, rather than being provided by the adversary. To obtain the joint state realizations, the functionalities have to be designed carefully. Other formulations proposed in the literature are shown to be unsuitable. Our work is based on the IITM model. Our definitions and results demonstrate the expressivity and simplicity of this model. For example, unlike Canetti's UC model, in the IITM model no explicit joint state operator needs to be defined and the joint state theorem follows immediately from the composition theorem in the IITM model.

**Keywords:** *Universal Composability, IITM Model, Joint State Composition, Ideal Functionalities, Public-Key Encryption, Digital Signatures*

*This work is an extended and updated version of the paper [22].

# Contents

# 1 Introduction

In frameworks for universal composability (see, e.g., [6, 7, 9, 15, 18–21, 24, 26]) the security of protocols is defined in terms of an ideal protocol (also called an ideal functionality). A real protocol securely realizes the ideal protocol if every attack on the real protocol can be translated to an "equivalent" attack on the ideal protocol, where equivalence is specified based on an environment trying to distinguish the real attack from the ideal one. That is, for every real adversary on the real protocol there must exist an ideal adversary (also called a simulator) on the ideal protocol such that no environment can distinguish whether it interacts with the real protocol and the real adversary or the ideal protocol and the ideal adversary. So the real protocol is as secure as the ideal protocol (which, by definition, is secure) in all environments. At the core of the universal composability approach are composition theorems which say that if a protocol uses one or more (independent) instances[1] of an ideal functionality, then all instances of the ideal functionality can be replaced by instances of the real protocol that realizes the ideal functionality. In this way, more and more complex protocols can be designed and analyzed in a modular way based on ideal functionalities, which later can be replaced by their realizations.

However, as first pointed out and studied by Canetti and Rabin [14] (see the related work), this modular approach often leads to impractical implementations since the composition theorems assume that different instances of a protocol have disjoint state. In particular, the random coins used in different instances have to be chosen independently. Consequently, when, for example, using a functionality for digital signatures within a more complex protocol, e.g., a key exchange protocol, parties have to generate new verification and signing keys for every instance of the protocol. This is completely impractical and motivates to generalize composition theorems to so-called joint state (composition) theorems, where different instances of a protocol may share some state, such as the same verification and signing keys.

The main goal of this paper is to obtain a general joint state theorem and to apply it to (novel) public-key encryption, replayable public-key encryption, and digital signature functionalities with local computation. In these functionalities, ciphertexts and signatures are computed locally, rather than being provided by the adversary, a feature often needed in applications. To obtain the joint state realizations, the functionalities have to be designed carefully. Other formulations proposed in the literature are shown to be unsuitable.

**Contribution of this paper.** In a nutshell, our contributions include *(i)* novel and rigorous formulations of ideal (replayable) public-key encryption and digital signature functionalities with local computation, along with their implementations, *(ii)* a joint state theorem which is more general than other formulations and corrects flaws in these formulations, and *(iii)* based on this theorem, joint state realizations and theorems for (replayable) public-key encryption and digital signatures.

Unfortunately, all other joint state theorems claimed in the literature for such functionalities with local computation can be shown to be flawed. An overall distinguishing feature of our work is the rigorous treatment, the simplicity of our definitions, and the generality of our results, which is due to the expressivity and simplicity of the model for universal composability that we use, the IITM model [21, 24]. For example, unlike Canetti's UC model [6, 7],[2] in the IITM model no explicit joint state operator needs to be defined and the joint state theorem follows immediately from the composition theorems of the IITM model. More precisely, our contributions are as follows.

*(i)* We formulate three functionalities: digital signatures, public-key encryption, and replayable public-key encryption. Our formulation of replayable public-key encryption is meant to model in a universal composability setting the notion of replayable IND-CCA2 security (IND-RCCA security) [12]. This relaxation of IND-CCA2 security permits anyone to generate new ciphertexts that decrypt to the same plaintext as a given ciphertext. As argued in [12], IND-RCCA security suffices for most existing applications of IND-CCA2 security. In our formulations of the above mentioned functionalities ciphertexts and signatures are determined by local

[1] We often also use the term "copy" in this work. These terms are used interchangeably.

[2] This paper was submitted in 2013, when the most recent version of the UC model was the one from 2013. This is the version that we refer to when we say "the most recent version" or "recent versions" in the following. A new version was published at the end of 2018 while this paper was still under review.

computations, and hence, as needed in many applications, a priori do not reveal signed messages or ciphertexts. In other formulations, e.g., those in [1, 8, 12, 14, 17], signatures and ciphertexts are determined by interaction with the adversary, with the disadvantage that the adversary learns all signed messages and all ciphertexts. Hence, such functionalities cannot be used, for example, in the context of secure message transmissions where a message is first signed and then encrypted, or in protocols with nested encryptions. Although there exist formulations of non-replayable public-key encryption and digital signature functionalities with local computation in the literature, these formulations have several deficiencies, in particular, as mentioned, concerning joint state realizations (see below).

We show that a public-key encryption scheme implements our (replayable) public-key encryption functionality if and only if it is IND-CCA2 secure (IND-RCCA secure), in case of static corruptions. We also prove equivalence between UF-CMA security of digital signatures schemes and our digital signature functionality, in case of adaptive corruptions.

*(ii)* In the spirit of Canetti and Rabin [14], we state a general joint state theorem. However, in contrast to Canetti's UC model as employed in [14] and the new versions of his model [6], within the IITM model we do not need to explicitly define a specific joint state operator. Also, our joint state theorem, unlike the one in the UC model, immediately follows from the composition theorem in the IITM model, no extra proof is needed. In addition to the seamless treatment of the joint state theorem within the IITM model, which exemplifies the simplicity and expressivity of the IITM model, our theorem is even more general than the ones in [6, 14] (see Section 3). We also note in Section 3 that, due to the kind of ITMs used in the UC model, the assumptions of the joint state theorems in the UC models can in many interesting cases not be satisfied and in the cases where they are satisfied, the theorem does not necessarily hold true.

We note that, similarly to the UC model, in the proposed GNUC model [20] dealing with joint state is quite cumbersome as well. In this model, in a run of a system machines have to form a call tree (every machine must have a unique caller), which is not the case in settings with joint state. Hence, unlike the IITM model, this model does not allow for dealing with joint state in a natural and smooth way. For example, the general joint state theorem does not immediately follow from the composition theorem in the GNUC model. It rather requires a non-trivial proof, which has to take into account details fixed in the GNUC model, such as corruption and so-called invited messages.

*(iii)* We apply our general joint state theorem to obtain joint state theorems for our (replayable) public-key encryption and digital signature functionalities. These joint state theorems are based on our ideal functionalities alone, and hence, work for all implementations of these functionalities. While the core of our joint state realizations are quite standard, their constructions and the proofs need care; as already mentioned, all other joint state theorems claimed in the literature for such functionalities with local computation are flawed.

**Related work.** As mentioned, Canetti and Rabin [14] were the first to explicitly study the problem of joint state, based on Canetti's original UC model [7]. They propose a general joint state theorem and apply it to a digital signature functionality with non-local computation (see also [1, 13]), i.e., the adversary is asked to provide a signature for every message. While the basic ideas in this work are interesting and useful, their general joint state theorem has several problems and limitations, as discussed in Section 3.

While most formulations of digital signatures and public-key encryption proposed in the literature use non-local computation, some formulations with local computations exist, which however, as already mentioned, are unsuitable for obtaining joint state realizations (see Section 6 for a detailed discussion).

For example, in [6] (version of December 2005),[3] Canetti proposes functionalities for public-key encryption and digital signatures with local computation. He sketches a functionality for replayable public-key encryption in a few lines. However, this formulation only makes sense in a setting with non-local computation, as proposed in [12]. As for joint state, Canetti only points to [14], with the limitations and problems inherited from this work. Moreover, as further discussed in Section 6, the joint state theorems claimed for the public-key encryption and digital signature functionalities in [6] are flawed. The same is true for the work by Canetti

[3] In the most recent version, the one from July 2013, Canetti does not consider functionalities for public-key encryption and digital signatures.

and Herzog in [11], where another public-key encryption functionality with local computation is proposed and a joint state theorem is claimed.

We note that, despite the problems with the joint state theorem and its application in the UC model pointed out in this work (see Sections 3.1 and 6 for detailed discussions), the basic ideas and contributions in that model are important and useful. However, we believe that it is crucial to equip that body of work with a more rigorous and elegant framework. This is one of the goals of this work.

In [10], Canetti et al. study universal composability with global setup. We note that they have to extend the UC model to allow the environment to access the functionality for the global setup. In the IITM model, this is not necessary (see the discussion in [24]). The global setup can be considered as joint state. But it is a joint state shared across *all* entities, unlike the joint state settings considered here, where the joint state is only shared within instances of functionalities. Therefore the results proved in [10] do not apply to the problem studied in this paper.

The present paper is an extended and updated version of [22]. In contrast to [22], where we use the original version of the IITM model [21], here we use the new version [24].

**Structure of the paper.** In the following section, we briefly recall the IITM model. The general joint state theorem is presented in Section 3, along with a discussion of the joint state theorem of Canetti and Rabin [14]. In Section 4, we present our formulations of ideal functionalities for digital signatures, public-key encryption, and replayable public-key encryption along with realizations of these functionalities. Joint state realizations of these functionalities are provided in Section 5. In Section 6, we discuss further related work and provide more details for the related work mentioned above. Some more details are provided in the appendix.

**Notation and basic terminology.** For a bit string $a \in \{0,1\}^*$ we denote by $|a|$ the length of $a$. Given bit strings $a_1, \ldots, a_n$, by $(a_1, \ldots, a_n)$ we denote the tuple consisting of these bit strings. We assume that tuples have a simple bit string representation and that converting a tuple to its bit string representation and vice versa is efficient. We do not distinguish between a tuple and its bit string representation.

Following [6, 24], a function $f\colon \mathbb{N} \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$ is called *negligible* if for all $c, d \in \mathbb{N}$ there exists $\eta_0 \in \mathbb{N}$ such that for all $\eta > \eta_0$ and all $a \in \bigcup_{\eta' \leq \eta^d} \{0,1\}^{\eta'}$: $f(\eta, a) < \eta^{-c}$.[4] A function $f\colon \mathbb{N} \times \{0,1\}^* \to [0,1]$ is called *overwhelming* if $1 - f$ is negligible.[5]

**Acknowledgments.** We thank Ran Canetti for many interesting discussions on the UC model and joint state. This work was in part funded by the *Deutsche Forschungsgemeinschaft (DFG)* through Grant KU 1434/9-1.

## 2 The IITM Model

In this section, we recall the IITM model [21, 24], a simple and expressive model for universal composability. More precisely, here we use the IITM model as presented in [24], which equips the original IITM model [21] with a more general notion of runtime. This allows us to formulate protocols and ideal functionalities in a more intuitive way, without technical artifacts concerning runtime. As discussed in [24], the (new) IITM model has several advantages compared to other models for universal composability. In particular, it resolves problems in Canetti's UC model and does not suffer from restrictions imposed in the GNUC model [20]. As already mentioned in the introduction and further discussed in Section 6, these problems and restrictions also affect the joint state theorems.

[4] We note that this definition of negligibility is equivalent to the following: $f$ is negligible if and only if for all positive polynomials $p(\eta)$ and $q(\eta)$ in $\eta \in \mathbb{N}$ (i.e., $p(\eta) > 0$ and $q(\eta) > 0$ for all $\eta \in \mathbb{N}$) there exists $\eta_0 \in \mathbb{N}$ such that for all $\eta > \eta_0$ and all $a \in \bigcup_{\eta' \leq q(\eta)} \{0,1\}^{\eta'}$: $f(\eta, a) < \frac{1}{p(\eta)}$. We further note that negligible functions have the following properties: (i) If $f$ and $g$ are negligible, then $f+g$ is negligible. (ii) If $f$ is negligible and $p(n)$ is a positive polynomial in $n \in \mathbb{N}$, then $g(\eta, a) := p(\eta + |a|) \cdot f(\eta, a)$ and $g'(\eta, a) := p(\eta) \cdot f(\eta, a)$, for all $\eta, a$, are negligible.

[5] By $[0,1]$ we denote the interval of all real numbers $x$ such that $0 \leq x \leq 1$.

As discussed in [24], the IITM model does not fix details such as addressing of machines by party/session IDs or corruption. Such details can be specified in a flexible and general way as part of the protocol specification. The IITM model also does not impose any specific structure, e.g., a hierarchical structure with protocols and subroutines, on systems. Altogether, this makes the model more expressive. It also makes the theorems proven in the IITM model, such as composition and joint state theorems, more general as they hold true for a large class of protocols and no matter how certain details are fixed.

Since the IITM model is in the spirit of Canetti's UC model, we note that conceptually the results presented in this paper also carry over to other models for universal composability.

## 2.1 The General Computational Model

In the IITM model, security notions and composition theorems are formalized based on a simple, expressive general computational model, in which IITMs (inexhaustible interactive Turing machines) and systems of IITMs are defined.

**Inexhaustible interactive Turing machines.** An *inexhaustible interactive Turing machine (IITM)* is a probabilistic Turing machine with named input and output tapes as well as an associated polynomial. The tape names determine how different machines are connected in a system of IITMs (see below). Tapes named start and decision serve a particular purpose when running a system of IITMs. It is required that only input tapes can be named start and only output tapes can be named decision. Tapes named start are used to provide a system with external input and to trigger an IITM if no other IITM was triggered. An IITM is triggered by another IITM if the latter sends a message to the former. An IITM with an input tape named start is called *master IITM*. On tapes named decision the final output of a system of IITMs will be written. An IITM runs in one of two modes, CheckAddress and Compute. The CheckAddress mode is used as a generic mechanism for addressing copies of IITMs in a system of IITMs, as explained below. In this mode, an IITM may perform, in every activation, a deterministic polynomial-time computation in the length of the security parameter plus the length of the current input plus the length of its current configuration, where the polynomial is the one associated with the IITM. The IITM is supposed to output "accept" or "reject" at the end of the computation in this mode, indicating whether the received message is processed further or ignored. The actual processing of the message, if accepted, is done in mode Compute. In this mode, a machine may only output at most one message on an output tape (and hence, only at most one other machine is triggered). The runtime in this mode is not a priori bounded. Later the runtime of systems and their subsystems will be defined in such a way that the overall runtime of a system of IITMs is polynomially bounded in the security parameter. We note that in both modes, an IITM cannot be exhausted (hence, the name): in every activation it can perform actions and cannot be forced to stop. This property, while not satisfied in all other models, is crucial to obtain a reasonable model for universal composability (see, e.g., [24] for more discussion).

**Systems of IITMs.** A *system* $\mathcal{S}$ of IITMs is of the form

$$\mathcal{S} = M_1 \mid \cdots \mid M_k \mid !M'_1 \mid \cdots \mid !M'_{k'}$$

where $M_i$, $i \in \{1, \ldots, k\}$, and $M'_j$, $j \in \{1, \ldots, k'\}$, are IITMs such that, for every tape name $c$, at most two of these IITMs have a tape named $c$ and, if two IITMs have a tape named $c$, this tape is an input tape in one of the machines and an output tape in the other. That is, two IITMs can be connected via tapes with the same name and opposite directions. These tapes are called *internal* and all other tapes are called *external*. The IITMs $M'_j$ are said to be in the scope of a bang operator. This operator indicates that in a run of a system an unbounded number of (fresh) copies of a machine may be generated. Conversely, machines which are not in the scope of a bang operator may not be copied. Systems in which multiple copies of a machine may be generated are often needed, e.g., in case of multi-party protocols or in case a system describes the concurrent execution of multiple instances of a protocol. The above conditions imply that in every system only at most one IITM may be a master IITM, i.e., may have an input tape named start; there may be several copies of such a machine in a run of a system though.

**Running a system.** In a run of a system $\mathcal{S}$ with security parameter $\eta$ and external input $a$ — such a system is denoted by $\mathcal{S}(1^\eta, a)$ —, at any time only one (copy of an) IITM is active and all other (copies of) IITMs wait for new input.[6] The active copy, say $M'$, which is a copy of a machine $M$ defined in $\mathcal{S}$, may write at most one message, say $m$, on one of its output tapes, say $c$. This message is then delivered to another (copy of an) IITM with an input tape named $c$, say $N$ is the machine specified in $\mathcal{S}$ with an input tape named $c$.[7] In the current configuration of the system, there may be several copies of $N$. In the order of creation, the copies of $N$ are run in mode CheckAddress with input $m$. Once one copy accepts $m$, this copy gets to process $m$, i.e., it runs in mode Compute with input $m$, and in particular, may produce output on one output tape, which is then sent to another copy and so on. If no copy of $N$ accepts $m$ and $N$ is in the scope of a bang, a fresh copy of $N$ is created and run in mode CheckAddress. If this copy accepts $m$, it gets to process $m$ in mode Compute. Otherwise, the new copy of $N$ is deleted, $m$ is dropped, and a master IITM is activated (with empty input). If $N$ is not in the scope of a bang (and—the only copy of—$N$ does not accept $m$), then too a master IITM is activated. The first IITM to be activated in a run is a master IITM. It gets the bit string $a$ as external input (on tape start). A master IITM is also activated if the currently active machine does not produce output (i.e., stops in its activation without writing to any output tape). A run stops if a master IITM, after being activated, does not produce output or output was written by some machine on an output tape named decision. The *overall output* of the run is defined to be the message that is output on decision. The probability that, in runs of $\mathcal{S}(1^\eta, a)$, the overall output is $m \in \{0,1\}^*$ is denoted by

$$\Pr\left[\mathcal{S}(1^\eta, a) = m\right] \quad .[8]$$

To illustrate runs of systems, consider, for example, the system $\mathcal{S} = M_1 \mid !M_2$ and assume that $M_1$ has an output tape named $c$, $M_2$ has an input tape named $c$, and $M_1$ is the master IITM. (There may be other tapes connecting $M_1$ and $M_2$.) Furthermore, assume that in the run of $\mathcal{S}$ executed so far, two copies of $M_2$, say $M_2'$ and $M_2''$, have been generated, with $M_2'$ generated before $M_2''$, and that $M_1$ just sent a message $m$ on tape $c$. This message is delivered to $M_2'$ (as the first copy of $M_2$). First, $M_2'$ runs in mode CheckAddress with input $m$; as mentioned, this is a deterministic polynomial-time computation which outputs "accept" or "reject". If $M_2'$ accepts $m$, then $M_2'$ gets to process $m$ in mode Compute and could, for example, send a message back to $M_1$. Otherwise, $m$ is given to $M_2''$ which then runs in mode CheckAddress with input $m$. If $M_2''$ accepts $m$, then $M_2''$ gets to process $m$ in mode Compute. Otherwise (if both $M_2'$ and $M_2''$ do not accept $m$), a new copy $M_2'''$ of $M_2$ with fresh randomness is generated and $M_2'''$ runs in mode CheckAddress with input $m$. If $M_2'''$ accepts $m$, then $M_2'''$ gets to process $m$. Otherwise, $M_2'''$ is removed again, the message $m$ is dropped, and the master IITM is activated (with empty input), in this case $M_1$, and so on.

**Equivalence/indistinguishability of systems.** Two systems that produce overall output 1 with almost the same probability are called equivalent or indistinguishable:

**Definition 1** ( [24])**.** Let $f\colon \mathbb{N} \times \{0,1\}^* \to \mathbb{R}_{\geq 0}$ be a function. Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called *$f$-equivalent* or *$f$-indistinguishable* ($\mathcal{P} \equiv_f \mathcal{Q}$) if and only if for every security parameter $\eta \in \mathbb{N}$ and external input $a \in \{0,1\}^*$:

$$\left|\Pr\left[\mathcal{P}(1^\eta, a) = 1\right] - \Pr\left[\mathcal{Q}(1^\eta, a) = 1\right]\right| \leq f(\eta, a) \quad .$$

Two systems $\mathcal{P}$ and $\mathcal{Q}$ are called *equivalent* or *indistinguishable* ($\mathcal{P} \equiv \mathcal{Q}$) if and only if there exists a negligible function $f$ such that $\mathcal{P} \equiv_f \mathcal{Q}$.

It is easy to see that for every two functions $f, f'$ as in Definition 1 the relation $\equiv_f$ is reflexive and that $\mathcal{P} \equiv_f \mathcal{Q}$ and $\mathcal{Q} \equiv_{f'} \mathcal{S}$ implies $\mathcal{P} \equiv_{f+f'} \mathcal{S}$. In particular, $\equiv$ is reflexive and transitive.

[6] We would like to emphasize the difference between a *description* of a machine and an *instance/copy* of a machine. The description of a machine $M$ specifies the behavior of a machine and is part of the specification of a system $\mathcal{S}$. In a run of $\mathcal{S}$, instances of $M$ are created. These instances have a specific state (or configuration), receive input on their input tapes, process the input according to their specifications (program code), thereby updating their state, and produce output. In what follows, for simplicity, we do not always distinguish between a description of a machine and its instances as the meaning is clear from the context.

[7] By the convention on the names of input tapes in systems of IITMs there can be at most one such machine.

[8] Formally, $\mathcal{S}(1^\eta, a)$ is a random variable that describes the overall output of runs of $\mathcal{S}(1^\eta, a)$, based on a standard probability space for runs of systems, see [24] for details.

**Composition of systems.** We say that a system $\mathcal{P}$ is *connectable* or *can be connected to* a system $\mathcal{Q}$ if $\mathcal{P}$ connects only to the external tapes of $\mathcal{Q}$, i.e., tapes with the same name in $\mathcal{P}$ and $\mathcal{Q}$ are external tapes of $\mathcal{P}$ and $\mathcal{Q}$, respectively, and they have opposite directions (an input tape in one system is an output tape in the other). By $\mathcal{P} \mid \mathcal{Q}$ we denote the composition of the systems $\mathcal{P}$ and $\mathcal{Q}$, defined in the obvious way. For example, if $\mathcal{P} = M_1 \mid !M_2$ and $\mathcal{Q} = !M_3 \mid !M_4 \mid !M_5$, then $\mathcal{P} \mid \mathcal{Q} = M_1 \mid !M_2 \mid !M_3 \mid !M_4 \mid !M_5$. When writing $\mathcal{P} \mid \mathcal{Q}$ we implicitly assume that the internal tapes of $\mathcal{P}$ and $\mathcal{Q}$ are renamed in such a way that the sets of internal tapes of $\mathcal{P}$ and $\mathcal{Q}$ are disjoint. This guarantees that $\mathcal{P}$ and $\mathcal{Q}$ communicate only over their external tapes.

## 2.2 Polynomial Time and Properties of Systems

So far, the runtime of IITMs in mode Compute has not been restricted in any way. To define notions of universal composability, it has to be enforced that systems run in polynomial time (except maybe with negligible probability). This will be done based on the following runtime notions.

A system $\mathcal{S}$ is called *strictly bounded* if there exists a polynomial $p$ such that, for every security parameter $\eta$ and external input $a$, the overall runtime of $\mathcal{S}$ in mode Compute (i.e., the overall number of transitions taken in this mode) is bounded by $p(\eta + |a|)$ in every run of $\mathcal{S}(1^\eta, a)$. If this holds only for an overwhelming set of runs, $\mathcal{S}$ is still called *almost bounded*. As shown in [24], every almost/strictly bounded system can be simulated (except maybe with a negligible error) by a probabilistic polynomial-time Turing machine.

A system $\mathcal{E}$ is called *universally bounded* if there exists a polynomial $p$ such that, for every security parameter $\eta$, external input $a$, and system $\mathcal{S}$ that can be connected to $\mathcal{E}$, the overall runtime of $\mathcal{E}$ in mode Compute is bounded by $p(\eta + |a|)$ in every run of $(\mathcal{E} \mid \mathcal{S})(1^\eta, a)$. (We note that environmental systems will be defined to be universally bounded, see below.)

A system $\mathcal{P}$ is called *environmentally (almost) bounded* if $\mathcal{E} \mid \mathcal{P}$ is almost bounded for every universally bounded system $\mathcal{E}$ that can be connected to $\mathcal{P}$. Similarly, $\mathcal{P}$ is called *environmentally strictly bounded* if $\mathcal{E} \mid \mathcal{P}$ is strictly bounded for every universally bounded system $\mathcal{E}$ that can be connected to $\mathcal{P}$.[9] (We note that protocol systems will be environmentally bounded. Therefore, it will be guaranteed that a protocol, together with an environment, runs in polynomial time, see below.)

## 2.3 Notions of Universal Composability

To define notions of universal composability, we first introduce the following terminology. For a system $\mathcal{S}$, the external tapes are grouped into *I/O* and *network tapes*. Three different types of systems are considered: protocol systems, adversarial systems, and environmental systems, modeling *(i)* real and ideal protocols/functionalities, *(ii)* adversaries and simulators, and *(iii)* environments, respectively. *Protocol systems*, *adversarial systems*, and *environmental systems* are systems which have an I/O and network interface, i.e., they may have I/O and network tapes. Environmental systems have to be universally bounded and protocol systems have to be environmentally bounded.[10] Protocol systems and adversarial systems may not have a tape named start or decision; only environmental systems may have such tapes, i.e., environmental systems may contain a master IITM and may determine the overall output of a run. Furthermore, for every IITM $M$ that occurs in a protocol system and is not in the scope of a bang, it is required that $M$ accepts every incoming message in mode CheckAddress.[11]

[9] As discussed in [24], since the runtime of universally bounded systems is polynomially bounded, the definition of environmentally almost/strictly bounded is equivalent to the following: $\mathcal{P}$ is environmentally almost/strictly bounded iff, for every universally bounded system $\mathcal{E}$, there exists a polynomial $p$ such that, for every $\eta$ and $a$, the overall runtime of $\mathcal{P}$ in mode Compute (i.e., the overall number of transitions taken by machines of $\mathcal{P}$ in mode Compute) is bounded by $p(\eta + |a|)$ in every run of $(\mathcal{E} \mid \mathcal{P})(1^\eta, a)$ (except for a negligible set of runs).

[10] We note that protocol systems, as defined in [24], per se are not required to be environmentally bounded. Instead, to obtain more general results, this is explicitly stated where needed. However, in most applications and throughout this paper protocol systems are always environmentally bounded (or even environmentally strictly bounded). Therefore, we simply require protocol systems to be environmentally bounded here.

[11] The motivation behind this condition is that if $M$ does not occur in the scope of a bang, then, in every run of the protocol system (in some context), there will be at most one copy of $M$. Hence, there is no reason to address different copies of $M$, and therefore, in mode CheckAddress, $M$ should accept every incoming message. This condition is needed in the proofs of the composition theorems for unbounded self-composition.

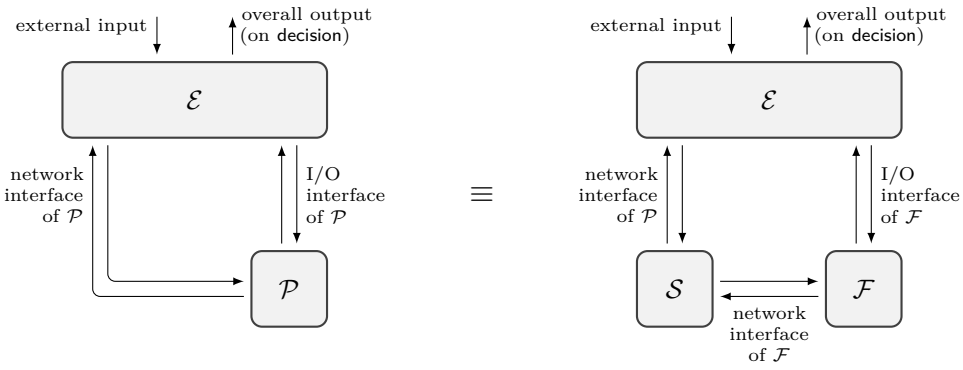


Figure 1: Strong simulatability ($\mathcal{P} \leq \mathcal{F}$). We note that $\mathcal{P}$ and $\mathcal{F}$ have the same I/O interface.

Given a system $\mathcal{S}$, the set of all environmental systems that can be connected to $\mathcal{S}$ (on the network or I/O interface) is denoted by $\text{Env}(\mathcal{S})$. For two protocol systems $\mathcal{P}$ and $\mathcal{F}$, $\text{Sim}^{\mathcal{P}}(\mathcal{F})$ denotes the set of all adversarial systems $\mathcal{A}$ such that $\mathcal{A}$ can be connected to $\mathcal{F}$, the set of external tapes of $\mathcal{A}$ is disjoint from the set of I/O tapes of $\mathcal{F}$ (i.e., $\mathcal{A}$ only connects to the network interface of $\mathcal{F}$), $\mathcal{A} \,|\, \mathcal{F}$ and $\mathcal{P}$ have the same external network and I/O interface, and $\mathcal{A} \,|\, \mathcal{F}$ is environmentally bounded.

We now recall the definition of strong simulatability; other, equivalent, security notions, such as UC and dummy UC, can be defined in a similar way [24]. The systems considered in this definition are depicted in Figure 1.

**Definition 2** ( [24])**.** Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems, the *real* and *ideal protocol*, respectively. Then, $\mathcal{P}$ *realizes* $\mathcal{F}$ ($\mathcal{P} \leq \mathcal{F}$) if and only if there exists a simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}}(\mathcal{F})$ (also called ideal adversary) such that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$ for every environment $\mathcal{E} \in \text{Env}(\mathcal{P})$.

As shown in [24], this relation is reflexive and transitive.

## 2.4 Composition Theorems

The first composition theorem handles concurrent composition of a fixed number of (possibly different) protocol systems. The second one guarantees secure composition of an unbounded number of copies of a protocol system.

**Theorem 1** ( [24])**.** *Let $k \geq 1$. Let $\mathcal{Q}, \mathcal{P}_1, \ldots, \mathcal{P}_k, \mathcal{F}_1, \ldots, \mathcal{F}_k$ be protocol systems such that they connect only via their I/O interfaces, $\mathcal{Q} \,|\, \mathcal{P}_1 \,|\, \cdots \,|\, \mathcal{P}_k$ is environmentally bounded, and $\mathcal{P}_i \leq \mathcal{F}_i$, for $i \in \{1, \ldots, k\}$. Then, $\mathcal{Q} \,|\, \mathcal{P}_1 \,|\, \cdots \,|\, \mathcal{P}_k \leq \mathcal{Q} \,|\, \mathcal{F}_1 \,|\, \cdots \,|\, \mathcal{F}_k$.*

Note that this theorem does not require that the protocols $\mathcal{P}_i/\mathcal{F}_i$ are subprotocols of $\mathcal{Q}$, i.e., that $\mathcal{Q}$ has matching external I/O tapes for all of these protocols. How these protocols connect to each other via their I/O interfaces is not restricted in any way, even the environment could connect directly to (the full or the partial) I/O interface of these protocols. Clearly, the theorem also holds true if the system $\mathcal{Q}$ is dropped.

For the following composition theorem, we introduce the notion of a session version of a protocol in order to be able to address copies of the protocol. Given an IITM $M$, the *session version* $\underline{M}$ of $M$ is an IITM which internally simulates $M$ and acts as a "wrapper" for $M$. More precisely, in mode CheckAddress, $\underline{M}$ accepts an incoming message $m'$ only if the following conditions are satisfied: (i) $\underline{M}$ has not accepted a message yet (in mode CheckAddress), $m'$ is of the form $(id, m)$, and $m$ is accepted by the simulated $M$ in mode CheckAddress. (In this case, later when activated in mode Compute, the ID $id$ will be stored by $\underline{M}$.) (ii) $\underline{M}$ has accepted a message before, $m'$ is of the form $(id', m)$, $id'$ coincides with the ID $id$ that $\underline{M}$ has stored before (in mode Compute), and $m$ is accepted by $M$ when simulated in mode CheckAddress. In mode Compute, if $\underline{M}$ is activated for the first time in this mode, i.e., the incoming message, say $m' = (id, m)$,

was accepted in mode CheckAddress for the first time, then first $id$ is stored and then $M$ is simulated with input $m$. Otherwise (if $\underline{M}$ was activated in mode Compute before), $M$ is directly simulated with input $m$. If the simulated $M$ produces output on some tape, then $\underline{M}$ prefixes this output with $id$ and then outputs the resulting message on the corresponding tape.

The ID $id$ typically is some session ID (SID) or some party ID (PID) or a combination of both. Clearly, it is not essential that messages are of the form $(id, m)$. Other forms are possible as well. In fact, everything checkable in polynomial time works.

To illustrate the notion of a session version of an IITM, assume that $M$ specifies some ideal functionality. Then $!\underline{M}$ denotes the multi-session version of $M$, i.e., a system in which an unbounded number of copies of $M$ can be created where every copy of $M$ can be addressed by a unique ID, where the ID could be a PID (then an instance of $\underline{M}$ might model one party running $M$) or an SID (then an instance of $\underline{M}$ models one session of $M$)

We sometimes require IDs to belong to a specific (polynomially decidable) domain $D$. In this case, we refer to *session version with domain $D$*. For such a session version, in mode CheckAddress only those SIDs are accepted that belong to $D$. With this, we could, for example, define a session version $\underline{M}$ of an IITM $M$ which only accepts SIDs of the form $(sid, pid)$, where $pid$ denotes a party and $sid$ identifies the session in which this party runs. Hence, in a run of the system $!\underline{M}$ (in some environment) all instances of $\underline{M}$ would have SIDs of this form. In this case, an instance $\underline{M}$ with ID $(sid, pid)$ models an instance of party $pid$ running $M$ in session $sid$.

In statements involving session versions, such as composition theorems, details of how the domains of SIDs are chosen are typically not important, as long as they are chosen consistently. We therefore omit such details in the statements.

Given a system $\mathcal{S}$, its *session version* $\underline{\mathcal{S}}$ is obtained by replacing all IITMs in $\mathcal{S}$ by their session version. For example, we obtain $\underline{\mathcal{S}} = \underline{M} \mid !\underline{M'}$ for $\mathcal{S} = M \mid !M'$.

Now, the following composition theorem says that if a protocol $\mathcal{P}$ realizes $\mathcal{F}$, then the multi-session version of $\mathcal{P}$ realizes the multi-session version of $\mathcal{F}$.

**Theorem 2** ( [24]). *Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems such that $!\underline{\mathcal{P}}$ is environmentally bounded and $\mathcal{P} \leq \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq !\underline{\mathcal{F}}$.*

We note that the extra proof obligation that $!\underline{\mathcal{P}}$ is environmentally bounded is typically easy to show. If $\mathcal{P}$ is environmentally strictly bounded (which should be the case in most applications), it even follows immediately that $!\underline{\mathcal{P}}$ is environmentally (strictly) bounded, as further discussed below.

Theorems 1 and 2 can be applied iteratively to construct more and more complex systems. For example, as an immediate consequence of Theorem 1 and 2 we obtain that if (an unbounded number of sessions of) an ideal protocol $\mathcal{F}$ is used as a component in a more complex system $\mathcal{Q}$, then it can be replaced by its realization $\mathcal{P}$:[12]

**Corollary 1.** *Let $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{F}$ be protocol systems such that $\mathcal{P}$ and $\mathcal{F}$ have the same I/O interface, $\mathcal{Q}$ only connects to the I/O interface of $!\underline{\mathcal{P}}$ (and, hence, $!\underline{\mathcal{F}}$), and $!\underline{\mathcal{P}}$ and $\mathcal{Q} \mid !\underline{\mathcal{P}}$ are environmentally bounded. If $\mathcal{P} \leq \mathcal{F}$, then $\mathcal{Q} \mid !\underline{\mathcal{P}} \leq \mathcal{Q} \mid !\underline{\mathcal{F}}$.*

When addressing a session version $\underline{M}$ of a machine $M$, the machine $M$ simulated within $M$ is not aware of its ID and cannot use it. For example, it cannot put the ID into a message that $M$ creates. However, sometimes this is desirable. Therefore another, more general, composition theorem is considered, where machines are aware of their IDs. While these IDs can, as already mentioned above, be interpreted in different ways, they will often be referred to as SIDs.

To this end, [24] first generalized the notion of a session version. They consider (polynomial-time computable) *session identifier (SID) functions* which, given a message and a tape name, output a SID (a bit string) or $\bot$. For example, the following function takes the prefix of a message as its SID: $\sigma_{\text{prefix}}(m, c) := s$ if $m = (s, m')$ for some $s, m'$ and $\sigma_{\text{prefix}}(m, c) := \bot$ otherwise, for all $m, c$. Clearly, many more examples are

[12] This corollary is in the spirit of Canetti's universal composition theorem [7].

conceivable. The reason that $\sigma$, besides a message, also takes a tape name as input is that the way SIDs are extracted from messages may depend on the tape a message is received on.

Given an SID function $\sigma$, an IITM $M$ is called a *$\sigma$-session machine* (or a *$\sigma$-session version*) if the following conditions are satisfied: *(i)* $M$ rejects (in mode CheckAddress) a message $m$ on tape $c$ if $\sigma(m, c) = \bot$. *(ii)* If $m_0$ is the first message that $M$ accepted (in mode CheckAddress), say on tape $c_0$, in a run, then, $M$ will reject all messages $m$ received on some tape $c$ (in mode CheckAddress) with $\sigma(m, c) \neq \sigma(m_0, c_0)$. *(iii)* Whenever $M$ outputs a messages $m$ on tape $c$, then $\sigma(m, c) = \sigma(m_0, c_0)$, with $m_0$ and $c_0$ as before.

A system $\mathcal{S}$ is a *$\sigma$-session version* if all IITMs defined in $\mathcal{S}$ are. It is easy to see that session versions are specific forms of $\sigma$-session versions: given an IITM $M$, we have that $\underline{M}$ is a $\sigma_{\text{prefix}}$-session version. The crucial difference is that while $\sigma$-session versions look like session version from the outside, inside they are aware of their SID.

Before the composition theorem can be stated, a notion of single-session realizability needs to be introduced.

An environmental system $\mathcal{E}$ is called *$\sigma$-single session* if it outputs messages only with the same SID according to $\sigma$. Hence, when interacting with a $\sigma$-session version, such an environmental system invokes at most one protocol session. Given a system $\mathcal{S}$ and an SID function $\sigma$, $\text{Env}_{\sigma\text{-single}}(\mathcal{S})$ denotes the set of all environments $\mathcal{E} \in \text{Env}(\mathcal{S})$ such that $\mathcal{E}$ is $\sigma$-single session, i.e., $\text{Env}_{\sigma\text{-single}}(\mathcal{S})$ is the set of all $\sigma$-single session environmental systems that can be connected to $\mathcal{S}$.

For two protocol systems $\mathcal{P}$ and $\mathcal{F}$ and an SID function $\sigma$, $\text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$ denotes the set of all adversarial systems $\mathcal{A}$ such that $\mathcal{A}$ can be connected to $\mathcal{F}$, the set of external tapes of $\mathcal{A}$ is disjoint from the set of I/O tapes of $\mathcal{F}$ (i.e., $\mathcal{A}$ connects to only the network interface of $\mathcal{F}$), $\mathcal{A} \,|\, \mathcal{F}$ has the same external tapes as $\mathcal{P}$, and $\mathcal{E} \,|\, \mathcal{A} \,|\, \mathcal{F}$ is almost bounded for every $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{A} \,|\, \mathcal{F})$. We note that $\text{Sim}^{\mathcal{P}}(\mathcal{F}) \subseteq \text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$; the only difference between these two sets is that the runtime condition on $\mathcal{A} \,|\, \mathcal{F}$ is relaxed in $\text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$.

Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems, which in the setting considered here would typically describe multiple sessions of a protocol. Moreover, we assume that $\mathcal{P}$ and $\mathcal{F}$ are $\sigma$-session versions. Now, it is defined what it means that a single session of $\mathcal{P}$ realizes a single session of $\mathcal{F}$. This is defined just as $\mathcal{P} \leq \mathcal{F}$ (Definition 2), with the difference that only $\sigma$-single session environments are considered, and hence, environments that invoke at most one session of $\mathcal{P}$ and $\mathcal{F}$.

**Definition 3** ( [24])**.** Let $\sigma$ be an SID function and let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems, the *real* and *ideal protocol*, respectively, such that $\mathcal{P}$ and $\mathcal{F}$ are $\sigma$-session versions. Then, $\mathcal{P}$ *single-session realizes* $\mathcal{F}$ *w.r.t.* $\sigma$ ($\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$) if and only if there exists $\mathcal{S} \in \text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$ such that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$ for every $\sigma$-single session environment $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{P})$.

Now, analogously to Theorem 2, the following theorem says that if $\mathcal{P}$ realizes $\mathcal{F}$ w.r.t. a single session, then $\mathcal{P}$ realizes $\mathcal{F}$ w.r.t. multiple sessions. As mentioned before, in the setting considered here $\mathcal{P}$ and $\mathcal{F}$ would typically model multi-session versions of a protocol/functionality.

**Theorem 3** ( [24])**.** *Let $\sigma$ be an SID function and let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems such that $\mathcal{P}$ and $\mathcal{F}$ are $\sigma$-session versions and $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$. Then, $\mathcal{P} \leq \mathcal{F}$.*

Clearly, this theorem can be combined with the other composition theorems to construct more and more complex systems. For example, similar to the above corollary, we obtain the following corollary:

**Corollary 2.** *Let $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{F}$ be protocol systems such that $\mathcal{P}$ and $\mathcal{F}$ are $\sigma$-session versions for some SID function $\sigma$, $\mathcal{P}$ and $\mathcal{F}$ have the same I/O interface, $\mathcal{Q}$ connects to only the I/O interface of $\mathcal{P}$ (and, hence, $\mathcal{F}$), and $\mathcal{Q} \,|\, \mathcal{P}$ is environmentally bounded. If $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$, then $\mathcal{Q} \,|\, \mathcal{P} \leq \mathcal{Q} \,|\, \mathcal{F}$.*

As discussed in [24], the composition of two environmentally bounded systems is not necessarily environmentally bounded. For instance, two systems, where each on its own is environmentally bounded, could play ping-pong, i.e., send message back and forth between each other. However, in applications the composition of environmentally almost/strictly bounded systems is basically always environmentally almost/strictly bounded. Moreover, in applications it is typically easy to see whether a system, including the composition of two environmentally almost/strictly bounded systems, is environmentally almost/strictly bounded. As also observed in [24], in applications protocol systems are typically *strictly* bounded, and for such systems we obtain useful general composability statements, which are briefly recalled next.

**Lemma 1** ( [24]). *Let $\mathcal{P}$ and $\mathcal{Q}$ be two environmentally strictly bounded protocol systems such that the sets of external tapes of $\mathcal{P}$ and $\mathcal{Q}$ are disjoint. Then, $\mathcal{P} \mid \mathcal{Q}$ is environmentally strictly bounded.*

This lemma can be generalized to the case where $\mathcal{P}$ and $\mathcal{Q}$ can communicate via tapes, provided that the information flow from $\mathcal{P}$ to $\mathcal{Q}$ is polynomially bounded in the security parameter, the length of the external input, and the overall length of messages $\mathcal{P}$ gets from the environment.

The following lemma says that the notion of environmentally strict boundedness is closed under unbounded self-composition.

**Lemma 2** ( [24]). *Let $\mathcal{S}$ be an environmentally strictly bounded protocol system. Then, $!\underline{\mathcal{S}}$ is environmentally strictly bounded.*

# 3 The Joint State Theorem

As already sketched in the introduction, joint state theorems are needed for the following reason. Composition theorems (for unbounded self-composition) state that it suffices to prove that a real protocol realizes an ideal functionality in a single session in order to conclude that multiple sessions of the real protocol realize multiple sessions of the ideal functionality. The problem is that this requires the states of the different sessions of the protocols/functionalities to be disjoint. In particular, the random coins used in different sessions have to be chosen independently. This, for example for digital signatures or public-key encryption, means that a party would have to choose new key pairs for *every* session, which is completely impractical.

Canetti and Rabin [14] proposed composition theorems with joint state, or joint state (composition) theorems for short, to solve this problem.

In this section, we first recall the general joint state theorem proposed by Canetti and Rabin in [14] and discuss several (partly severe) problems of this theorem. We then present a general joint state theorem in the IITM model. As we will see, this theorem does not suffer from the problems in the UC model and it can be stated in a more elegant and general way, and, unlike in the UC model, it follows immediately from the composition theorem as a simple special case.

## 3.1 The Joint State Theorem in the UC Model

To state the general joint state theorem proposed by Canetti and Rabin in the UC model, let $\mathbf{Q}$ be a protocol which uses multiple sessions with multiple parties of some ideal functionality $\mathbf{F}$, i.e., $\mathbf{Q}$ works in an $\mathbf{F}$-hybrid model. Let $\widehat{\mathbf{P}}$ be a realization of $\widehat{\mathbf{F}}$, where $\widehat{\mathbf{F}}$ is a single machine which simulates the multi-session multi-party version of $\mathbf{F}$. Now, $\mathbf{Q}^{[\widehat{\mathbf{P}}]}$ denotes the JUC composition of $\mathbf{Q}$ and $\widehat{\mathbf{P}}$, where calls from $\mathbf{Q}$ to $\mathbf{F}$ are translated to calls to $\widehat{\mathbf{P}}$ and where for each party there is only one copy of $\widehat{\mathbf{P}}$ and this copy handles all sessions of this party, i.e., $\widehat{\mathbf{P}}$ may make use of joint state. Now, Canetti and Rabin obtain the following theorem.

**Theorem 4** (Joint State Theorem in UC Model, informal). *If $\widehat{\mathbf{P}}$ realizes $\widehat{\mathbf{F}}$, then $\mathbf{Q}^{[\widehat{\mathbf{P}}]}$ realizes $\mathbf{Q}$ in the $\mathbf{F}$-hybrid model.*

The typical use case of this theorem is that $\widehat{\mathbf{P}}$ realizes $\widehat{\mathbf{F}}$ in the $\mathbf{F}$-hybrid model in such a way that $\widehat{\mathbf{P}}$ creates only one copy of $\mathbf{F}$ per party and that this copy handles all sessions of this party. The protocol $\widehat{\mathbf{P}}$ then plays the role of a kind of multiplexer which maps all sessions of one party to the corresponding copy of $\mathbf{F}$. In this sense, $\widehat{\mathbf{P}}$ is a joint state realization of the multi-session and multi-party version of $\mathbf{F}$. Now, the theorem says that if $\mathbf{Q}$ uses the multi-session and multi-party version of $\mathbf{F}$ (i.e., $\mathbf{Q}$ works in the $\mathbf{F}$-hybrid model where there is on fresh copy of $\mathbf{F}$ per party and session), then $\mathbf{Q}$ can instead use the joint state realization $\widehat{\mathbf{P}}$ where only one copy of $\mathbf{F}$ is used per party and this copy is used across all sessions of that party. For example, if $\mathbf{F}$ is an ideal functionality for digital signatures which allows one party to sign messages and allows all parties to verify signatures of that party, then the theorem says that the protocol $\mathbf{Q}$ which uses one "signing box" per party (through the joint state realization $\widehat{\mathbf{P}}$) realizes the protocol $\mathbf{Q}$ when it uses a new signing box per party and session.

As further discussed in Section 3.2, due to the restricted expressivity of the UC model and unlike the IITM model, formulating the joint state theorem in the UC model requires some new notions, such as the notion of JUC composition, and a non-trivial proof.

Moreover, unfortunately there are some partly severe technical problems with this theorem in the UC model as discussed next, which are mainly due to the way the runtime of (systems of) ITMs is defined.

We note that the JUC theorem from [14] has been shown for the initial version of the UC model from 2001 only. Thus, technically speaking, there currently is no JUC theorem for any of the more recent versions. However, as we argue below, the fundamental problems of the JUC theorem still exist even if one were to transfer the theorem to more recent versions of the UC model.

**Problems of the joint state theorem in the UC model.** In the UC model, the *overall* runtime of an ITM is bounded by a polynomial in the security parameter alone in the original UC model [7] or in the security parameter and the overall length of the input on the I/O interface in the new versions of the model [6], including the most recent one. Consequently, once the overall bound is hit, the ITMs are forced to stop. In particular, it is easy to force an ITM to stop by sending many (useless) messages (on the network interface). This, among others, results in the following problem in the UC model. In general, a single ITM, say $M$, cannot simulate a concurrent composition of a fixed finite number of ITMs, say $M_1, \ldots, M_n$, or an unbounded number of (instances of) ITMs: By sending many messages to $M$ intended for $M_1$, say, $M$ will eventually stop, and hence, cannot simulate the other machines anymore, even though, in the actual composition these machines could still take actions.

Now, this causes problems in the joint state theorem of the UC model: Although the ITM $\widehat{\mathbf{F}}$ in the joint state theorem is intended to simulate the multi-party, multi-session version of $\mathbf{F}$, for the reason explained above, it cannot do this in general; it can only simulate some approximated version. The same is true for $\widehat{\mathbf{P}}$. This, as further explained below, has several negative consequences:

A) For many interesting functionalities, including existing versions of digital signatures and public-key encryption, it is not always possible to find a $\widehat{\mathbf{P}}$ that realizes $\widehat{\mathbf{F}}$ (for a reasonable functionality $\mathbf{F}$), and hence, in these cases the precondition of the joint state theorem cannot be satisfied.

B) In some cases, the joint state theorem in the UC model itself fails.

ad A) We first illustrate the problem of realizing $\widehat{\mathbf{F}}$ in the original UC model, i.e., the one presented in [7], on which the work in [14] is based. We then explain the corresponding problem for the new versions of the UC model [6].

The ITM $\widehat{\mathbf{F}}$ is intended to simulate the multi-party, multi-session version of $\mathbf{F}$, e.g., a digital signature functionality. The realization $\widehat{\mathbf{P}}$ is intended to do the same, but it contains an ITM for every party. Now, consider an environment that sends many requests to one party, e.g., verification requests such that the answer to all of them is *ok*. Eventually, $\widehat{\mathbf{F}}$ will be forced to stop, as it runs out of resources. Consequently, requests to other parties cannot be answered anymore. However, such requests can still be answered in $\widehat{\mathbf{P}}$, because these requests are handled by other ITMs, which are not exhausted. Consequently, an environment can easily distinguish between the ideal ($\widehat{\mathbf{F}}$) and real world ($\widehat{\mathbf{P}}$). This argument works independently of the simulator. The situation just described is very common. Therefore, strictly speaking, for many functionalities of interest it is not possible to find a realization of $\widehat{\mathbf{F}}$ in the original UC model.

In the new versions of the UC model [6], the problem of realizing $\widehat{\mathbf{F}}$ is similar. However, ITMs cannot be exhausted (forced to stop) via communication on the I/O interface. Nevertheless, exhaustion is possible via the network interface. Assume that $\widehat{\mathbf{P}}$ tries to realize $\widehat{\mathbf{F}}$ in an $\mathbf{F}$-hybrid model, where for every party one instance of $\widehat{\mathbf{P}}$ and $\mathbf{F}$ is generated, if any.[13] The environment (via a dummy adversary) can access any copy of $\mathbf{F}$ in the $\mathbf{F}$-hybrid model directly via the network interface. In this way, the environment can send

[13] This, as already mentioned before, is the typical setting for joint state realizations. Our arguments also apply in many cases where $\widehat{\mathbf{P}}$ does not work in the $\mathbf{F}$-hybrid model, which is however quite uncommon. The whole point of modular protocol analysis and design is to use the ideal functionalities.

many messages to a copy of $\mathbf{F}$, and hence, exhaust this copy, i.e., force it to stop, after some time. Even when the copy has stopped, the environment can keep sending messages to this copy, which in the hybrid model does not have any effect. On the ideal side, the simulator, say $\mathbf{S}$, has to know when a copy of $\mathbf{F}$ would stop in the hybrid model, because it then must not forward messages addressed to this copy of $\mathbf{F}$ to $\widehat{\mathbf{F}}$. Otherwise, $\widehat{\mathbf{F}}$ would get exhausted as well and the environment could distinguish between the hybrid and the ideal world as above: It simply contacts another copy of $\mathbf{F}$ in the $\mathbf{F}$-hybrid world (via $\widehat{\mathbf{P}}$ and the I/O interface or directly via the network interface). This copy (since it is another ITM and not exhausted) would still be able to react, while $\widehat{\mathbf{F}}$ is not. However, in general $\mathbf{S}$ does not necessarily know if an instance in the hybrid model is exhausted, e.g., because the simulator does not know how many resources have been provided to the functionalities on the I/O interface, to which $\mathbf{S}$ does not have access, and how many resources the functionality has consumed. Hence, in this case $\mathbf{S}$ always has to forward messages, because the functionality might still have enough resources to react. But this then leads to the exhaustion of $\widehat{\mathbf{F}}$, with the consequence that the environment can distinguish between the hybrid and the ideal world as described above. It is easy to come up with functionalities where the problem just described occurs, including reasonable formulations of public-key encryption and digital signature functionalities. Typically, formulations of functionalities in the UC model are not precise about the runtime of functionalities, e.g., whether a functionality stops as soon as it gets a message of a wrong format or whether it ignores messages until it gets the expected message and only stops if it runs out of runtime. Ill-defined functionalities or different interpretations of how the runtime is defined can then lead to the mentioned problems. Even if there is a realization of $\widehat{\mathbf{F}}$ that would work, proving this can become quite tricky because of the described exhaustion problem and its consequences.

We note that even if one were to prove a different JUC theorem that, e.g., changes how $\widehat{\mathbf{F}}$ is defined, it would still be hard or even impossible to show that the precondition of the theorem is fulfilled for many interesting functionalities. This is due to the following general problem caused by the runtime notion employed by the UC model: In every type of joint-state realization, there is one instance $i$ in the real world that corresponds to multiple instances/sessions $s_1, \ldots, s_n$ in the ideal world. The runtime notion of the UC model allows the environment to exhaust the runtime of $i$ in the real world such that $i$ does not perform any actions anymore. The simulator $\mathbf{S}$ has to simulate the same behavior for the instances $s_1, \ldots, s_n$ in the ideal world. That is, $\mathbf{S}$ typically has to learn how much runtime each of the instances/sessions $s_1, \ldots, s_n$ currently has obtained so far, compute the runtime bound of $i$ from this, learn how much runtime is left for each $s_1, \ldots, s_n$, then send more runtime to those $s_j$ that would stop earlier than $i$, and stop those $s_j$ that would run longer than $i$. Thus, ideal functionalities in the UC model generally have to leak their runtime and provide some means to the simulator to stop sessions in order to enable joint-state realizations. This is typically not done by functionalities found in the literature and is not feasible in cases where the runtime depends on secret information.

ad B) Having discussed the problem of meeting the assumptions of the joint state theorem in the UC model, we now turn to flaws of the joint state theorem itself. For this, assume that $\widehat{\mathbf{P}}$ realizes $\widehat{\mathbf{F}}$ within the $\mathbf{F}$-hybrid model, with the (usual) intention that $\widehat{\mathbf{P}}$ creates only one copy per party of $\mathbf{F}$. Such a copy handles all sessions of $\mathbf{F}$ for that party. In contrast, $\widehat{\mathbf{F}}$ simulates a new copy of $\mathbf{F}$ per party and session. According to the joint state theorem in the UC model, we should have that $\mathbf{Q}^{[\widehat{\mathbf{P}}]}$ (real world) realizes $\mathbf{Q}$ (ideal world) in the $\mathbf{F}$-hybrid model. However, the following problems occur: An environment can directly access (via a dummy adversary) a copy of $\mathbf{F}$ in the real world. By sending many messages to this copy, this copy will be exhausted. This copy of $\mathbf{F}$, let us call it $\mathbf{F}_{[pid]}$, which together with $\widehat{\mathbf{P}}$ handles all sessions of a party $pid$, corresponds to several copies $\mathbf{F}_{[pid,\,sid]}$ of $\mathbf{F}$, for SIDs $sid$, in the ideal world. Hence, once $\mathbf{F}_{[pid]}$ in the real world is exhausted, the simulator also has to exhaust all its corresponding copies $\mathbf{F}_{[pid,\,sid]}$ in the ideal world for every $sid$, because otherwise an environment could easily distinguish the two worlds. (While $\mathbf{F}_{[pid]}$ cannot respond, some of the copies $\mathbf{F}_{[pid,\,sid]}$ still could otherwise.) Consequently, for the simulation to work, $\mathbf{F}$ will have to provide to the simulator a way to be terminated, a feature typically not contained in formulations of functionalities in the UC model. Hence, for such functionalities the joint state theorem would typically fail. However, this could be fixed by assuming this feature for functionalities (even though this might be quite artificial.) A more serious problem is that the simulator might not know whether $\mathbf{F}_{[pid]}$ in the real model

is exhausted (the simulator does not necessarily see how many resources $\mathbf{F}_{[pid]}$ gets from the I/O interface and how much resources $\mathbf{F}_{[pid]}$ has used), and hence, the simulator does not know when to terminate the corresponding copies in the ideal model. So, in these cases again the joint state theorem fails. In fact, just as in the case of realizing $\widehat{\mathbf{F}}$, it is not hard to come up with functionalities where the joint state theorem fails, including reasonable formulations of public-key encryption and digital signature functionalities. So, the joint state theorem cannot simply be applied to arbitrary functionalities. One has to reprove this theorem on a case by case basis or characterize classes of functionalities for which the theorem holds true.

We finally note that in the original UC model [7] there is yet another, but smaller problem with the joint state theorem. Since in the original UC model the number of copies of $\mathbf{F}$ that $\widehat{\mathbf{F}}$ can simulate is bounded by a polynomial in the security parameter, this number typically also has to be bounded in the realization $\widehat{\mathbf{P}}$. However, now the environment can instruct $\mathbf{Q}$ to generate many copies of $\mathbf{F}$ for one party. In the real world, after some time no new copies of $\mathbf{F}$ for this party can be generated because $\widehat{\mathbf{P}}$ is bounded. However, an unbounded number of copies can be generated in the ideal world, which allows the environment to distinguish between the real and ideal world. The above argument uses that the runtime of $\mathbf{Q}$ is big enough such that the environment can generate, through $\mathbf{Q}$, more copies than $\widehat{\mathbf{P}}$ can produce. So, this problem can easily be fixed by assuming that the runtime of $\mathbf{Q}$ is bounded appropriately. Conversely, given $\mathbf{Q}$, the runtime of $\widehat{\mathbf{P}}$ should be made big enough. This, however, has not been mentioned in the joint state theorem in [14].

As already mentioned in the introduction, despite of the various problems with the joint state theorem in the UC model, within that model useful and interesting results have been obtained. However, it is crucial to equip that body of work with a coherent as well as more rigorous and elegant framework. We believe that the IITM model provides such a framework.

## 3.2 The Joint State Theorem in the IITM Model

In order to present the joint state theorem in the IITM model, assume that $\mathcal{F}$ is a protocol system (modeling an ideal functionality). For our joint state theorem any protocol system can be used. In applications, $\mathcal{F}$ will typically model an ideal functionality that can be used by multiple parties in one session. For example, $\mathcal{F}$ could be some $\sigma_{\text{prefix}}$-session version which expects messages of the form $(pid, m)$, where $pid$ is a party ID (PID). A specific instance of such a functionality would be a functionality of the form $!\underline{\mathcal{F}'}$, where $\mathcal{F}'$ is a protocol system which describes an ideal functionality that can be used by one party in one session. Runs of $!\underline{\mathcal{F}'}$ can thus contain multiple instances of $\mathcal{F}'$ where every instance can be addressed by some ID, which in this case would be interpreted as a PID. In particular, messages to $!\underline{\mathcal{F}'}$ would be of the form $(pid, m)$ and such a message would be sent to the instance of $\mathcal{F}'$ corresponding to $pid$ and this instance would be given the message $m$.

Given some ideal functionality $\mathcal{F}$, the system $!\underline{\mathcal{F}}$ models a multi-session version of $\mathcal{F}$: A run of $!\underline{\mathcal{F}}$ can contain multiple sessions of $\mathcal{F}$. In order to send a message $m$ to session $sid$, one would send the message $(sid, m)$ to $!\underline{\mathcal{F}}$. If $\mathcal{F}$ is a multi-party, single-session formulation of an ideal functionality, as explained above, in order to send a message $m$ to party $pid$ in session $sid$ one would send the message $(sid, (pid, m))$ to $!\underline{\mathcal{F}}$.

In the formulation of our joint state theorem we use $!\underline{\mathcal{F}}$ to denote a multi-session version of the functionality $\mathcal{F}$. However, the specific form of the multi-session version does not matter. We could replace $!\underline{\mathcal{F}}$ by any protocol system. We use $!\underline{\mathcal{F}}$ because this system is closer to the intended application of the theorem.

Now, our joint state theorem can be stated as follows (see also Figure 2 for an illustration of the runs of the systems considered in this theorem).

**Theorem 5.** *Let $\mathcal{Q}, \mathcal{P}_{\text{js}}^{\mathcal{F}}, \mathcal{F}$ be (arbitrary) protocol systems such that $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ and $!\underline{\mathcal{F}}$ have the same I/O interface and $\mathcal{Q}$ connects only to the I/O interface of $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ (and, hence, $!\underline{\mathcal{F}}$), $\mathcal{Q} \,|\, \mathcal{P}_{\text{js}}^{\mathcal{F}}$ and $\mathcal{Q} \,|\, !\underline{\mathcal{F}}$ are environmentally bounded, and $\mathcal{P}_{\text{js}}^{\mathcal{F}} \leq\, !\underline{\mathcal{F}}$. Then, $\mathcal{Q} \,|\, \mathcal{P}_{\text{js}}^{\mathcal{F}} \leq \mathcal{Q} \,|\, !\underline{\mathcal{F}}$.*

*Proof.* By Theorem 1, $\mathcal{P}_{\text{js}}^{\mathcal{F}} \leq\, !\underline{\mathcal{F}}$ immediately implies that $\mathcal{Q} \,|\, \mathcal{P}_{\text{js}}^{\mathcal{F}} \leq \mathcal{Q} \,|\, !\underline{\mathcal{F}}$. □

The fact that Theorem 5 immediately follows from Theorem 1 shows that, in the IITM model, there is no need for an explicit (general) joint state theorem.

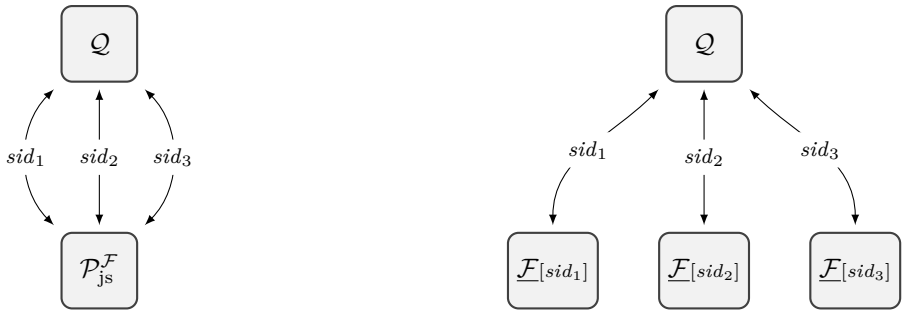


Figure 2: A run of $\mathcal{Q} \mid \mathcal{P}_{\text{js}}^{\mathcal{F}}$ (left) and $\mathcal{Q} \mid !\underline{\mathcal{F}}$ (right), respectively, with three sessions (with SIDs) $sid_1, sid_2, sid_3$. The runs are with respect to some environment that is not displayed. By $!\underline{\mathcal{F}}[sid_i]$ we denote the copy of $\underline{\mathcal{F}}$ that is addressed by $sid_i$. The arrows denote the connections between the systems via I/O tapes and addressing with SIDs. In addition, all systems may be connected to the environment via I/O and network tapes; these connections are not displayed.

The reason that such a theorem is needed in the UC model lies in the restricted expressivity of this model: First, one has to define a single ITM $\widehat{\mathbf{F}}$ which simulates the multi-party, multi-session version of $\mathcal{F}$. One cannot simply write $!\underline{\mathcal{F}}$ because multi-party, multi-session versions only exist as part of a hybrid model. In particular, one cannot write $\mathcal{P}_{\text{js}}^{\mathcal{F}} \leq !\underline{\mathcal{F}}$ directly, but has to say that $\widehat{\mathbf{P}}$ realizes $\widehat{\mathbf{F}}$. Second, the JUC operator has to be defined explicitly since it cannot be directly stated that only one instance of $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ is invoked by $\mathcal{Q}$; in the IITM model we can simply write $\mathcal{Q} \mid \mathcal{P}_{\text{js}}^{\mathcal{F}}$. Also, a composition theorem corresponding to Theorem 1, which is used to show that $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ can be replaced by $!\underline{\mathcal{F}}$, is not directly available in the UC model, only a composition theorem corresponding to Corollaries 1 and 2. Finally, due to the addressing mechanism employed in the UC model, redirection of messages have to be made explicit.

We note that despite the trivial proof of Theorem 5 in the IITM model (given the composition theorem), the statement that Theorem 5 makes is stronger than that of the joint state theorem in the UC model [6, 14]. Inherited from our composition theorems, and unlike the theorem in the UC model, Theorem 5 does not require that $\mathcal{Q}$ completely shields the sub-protocol from the environment, and hence, from super-protocols on higher levels. This can lead to simpler systems and more efficient implementations.

As already mentioned in the introduction and further explained in [24], also in the recently proposed GNUC model [20] it is necessary to explicitly state a joint state theorem. The main problem in that model is that it imposes a tree structure on protocols, which for joint state (and global state) is too restricted and requires a quite artificial workaround in that model.

**Applying the joint state theorem.** Theorem 5, just like the joint state theorem in the UC model, does not by itself yield practical joint state realizations, as it does not answer the question of how a practical realization $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ can be found. A desirable instantiation of $\mathcal{P}_{\text{js}}^{\mathcal{F}}$ would be of the form $!\mathcal{P}_{\text{js}} \mid \mathcal{F}$ where $!\mathcal{P}_{\text{js}}$ is a very simple protocol in which for every party only one copy of $\mathcal{P}_{\text{js}}$ is generated and this copy handles, as a multiplexer, all sessions of this party via the single instance of the ideal multi-party, single-session functionality $\mathcal{F}$. Hence, the goal is to find a protocol system $!\mathcal{P}_{\text{js}}$ (with one copy per party) such that:

$$!\mathcal{P}_{\text{js}} \mid \mathcal{F} \;\leq\; !\underline{\mathcal{F}} \;.^{14} \tag{1}$$

The protocol $!\mathcal{P}_{\text{js}} \mid \mathcal{F}$ will be called a *(practical) joint state realization* of $!\underline{\mathcal{F}}$ in what follows.

Now, assume that $\mathcal{P} \leq \mathcal{F}$. Provided that $\mathcal{F}$ is a multi-party, single-session functionality, note that $\mathcal{P}$ too is a multi-party protocol which realizes a single session of $\mathcal{F}$. By (1), the composition theorems, and the transitivity of $\leq$ we immediately obtain that $!\mathcal{P}_{\text{js}} \mid \mathcal{P} \leq !\underline{\mathcal{F}}$. That is, we obtain a realization of the multi-session

[14] Strictly speaking, one has to rename the I/O tapes of $\mathcal{F}$ on the right-hand side (or I/O tapes of $\mathcal{P}_{\text{js}}$ on the left-hand side), to ensure that both sides have the same external I/O interface.

version of $\mathcal{F}$ where only one session of $\mathcal{P}$ is used (in combination with the multiplexer $\mathcal{P}_{\text{js}}$) to realize all sessions of $\mathcal{F}$.

Moreover, if $\mathcal{F} = !\underline{\mathcal{F}'}$ is the multi-party, single-session version of the single-party, single-session functionality $\mathcal{F}'$ and $\mathcal{P}'$ realizes $\mathcal{F}'$, i.e., $\mathcal{P}' \leq \mathcal{F}'$, then $!\mathcal{P}_{\text{js}} \mid !\underline{\mathcal{P}'} \leq !\mathcal{P}_{\text{js}} \mid !\underline{\mathcal{F}'} \leq !\underline{\mathcal{F}} = !\underline{\underline{\mathcal{F}'}}$, where $\underline{\mathcal{P}'}$ denotes the party version of $\mathcal{P}'$, $\underline{\mathcal{F}'}$ the party version of $\mathcal{F}'$, and $\underline{\underline{\mathcal{F}'}}$ the session and party version of $\mathcal{F}'$. That is, to realize the multi-session and multi-party version of $\underline{\underline{\mathcal{F}'}}$, we obtain a joint state realization where only one copy of $\mathcal{P}'$ is used per party. This copy handles all sessions of that party.

The seamless treatment of joint state in the IITM model allows for iterative applications of the joint state theorem. Consider a protocol $\mathcal{Q}$ that uses the multi-session version $!\underline{\mathcal{F}}$ of a (multi-party) ideal functionality $\mathcal{F}$. That is, we consider the system $\mathcal{Q} \mid !\underline{\mathcal{F}}$. Furthermore, assume that multiple sessions of $\mathcal{Q}$ are used within a more complex protocol. Hence, such a protocol uses the system $!\underline{(\mathcal{Q} \mid !\underline{\mathcal{F}})} = !\underline{\mathcal{Q}} \mid !\underline{\underline{\mathcal{F}}}$. In this system, in every session of $\mathcal{Q}$ several sub-sessions of $\mathcal{F}$ can be used. Now iterated application of the composition theorems/joint state theorem and (1) yields: $!\underline{\mathcal{Q}} \mid !\underline{\underline{\mathcal{F}}} = !\underline{(\mathcal{Q} \mid !\underline{\mathcal{F}})} \geq !\underline{(\mathcal{Q} \mid (!\mathcal{P}_{\text{js}} \mid \mathcal{F}))} = !\underline{\mathcal{Q}} \mid !\underline{\mathcal{P}_{\text{js}}} \mid !\underline{\mathcal{F}} \geq !\underline{\mathcal{Q}} \mid !\underline{\mathcal{P}_{\text{js}}} \mid !\mathcal{P}_{\text{js}} \mid \mathcal{F}$. This means that $!\underline{\mathcal{P}_{\text{js}}} \mid !\mathcal{P}_{\text{js}} \mid \mathcal{F}$ is a joint state realization of $!\underline{\underline{\mathcal{F}}}$. Note that in this realization only a single instance of $\mathcal{F}$ is used to realize all sessions of $\mathcal{F}$ in the system $!\underline{\underline{\mathcal{F}}}$. Messages sent to $!\underline{\underline{\mathcal{F}}}$ (and hence, $!\underline{\mathcal{P}_{\text{js}}} \mid !\mathcal{P}_{\text{js}} \mid \mathcal{F}$) are of the form $(sid_1, sid_2, pid)$ where $sid_1$ denotes the SID of a session of $\mathcal{Q}$, $sid_2$ denotes the session of $\mathcal{F}$ within the session $sid_1$, and $pid$ denotes the party running in session $(sid_1, sid_2)$. While in $!\underline{\underline{\mathcal{F}}}$ there is a new copy of $\mathcal{F}$ for each SID $(sid_1, sid_2)$, in the joint state realization all such sessions would be handled by a single copy of $\mathcal{F}$. If $\mathcal{F} = !\underline{\mathcal{F}'}$, then all sessions $(sid_1, sid_2)$ for party $pid$ would be handled by the copy $\mathcal{F}'$ of $pid$. If, for example, $\mathcal{F}'$ is an ideal (single-party, single-session) public-key encryption functionality, then this means that there is only one decryption/encryption box for every party which is used across all sessions of $\mathcal{Q}$.

# 4 Ideal Functionalities

We now present ideal functionalities for digital signatures, public-key encryption, and replayable public-key encryption; along with realizations.

## 4.1 Notation for the Definition of IITMs

We start with notational conventions that we use in the following to define IITMs.

### 4.1.1 Pseudocode

To define IITMs (and algorithms in general), we use standard pseudocode with the obvious semantics.

By $x := y$ we denote deterministic assignment of a variable or constant $y$ to a variable $x$. By $x \leftarrow A$ we denote probabilistic assignment to a variable $x$ according to the distribution of the output of an algorithm $A$. By $x \xleftarrow{\$} S$ we denote that $x$ is chosen uniformly at random from a finite set $S$.

All values that are manipulated are bit strings or special symbols such as the symbol $\perp$. We only use very basic data structures. For example, we often use tuples and sets of bit strings. As already mentioned at the end of the introduction, for tuples we assume an efficient encoding as bit strings. Furthermore, we assume an efficient implementation of sets (e.g., by lists or tuples) that allows us *(i)* to add a bit string to a set, *(ii)* to remove a bit string from a set, *(iii)* to test if a bit string is an element of a set, and *(iv)* to iterate over all elements of a set. We denote the empty set by $\emptyset$.

### 4.1.2 Specification of IITMs

Most of our definitions of IITMs are divided into six parts (where some are optional): *Parameters*, *Tapes*, *State*, *CheckAddress*, *Initialization*, and *Compute*.

**Parameters.** In this part, we list all parameters of the IITM. That is, when defining a system that contains this IITM, these parameters have to be instantiated. This part is omitted if the IITM has no parameters.

For example, our ideal functionalities are typically parameterized by a number $n > 0$ that defines the I/O interface (more precisely, the number of I/O tape pairs, see below).

**Tapes.** This part lists all input and output tapes. Unless otherwise stated, I/O tapes are named $\mathsf{io}_x^y$ and network tapes are named $\mathsf{net}_x^y$ for some decorations $x, y$. The IITMs we define in this paper have a corresponding output tape for every input tape. The intuition is that, upon receiving a message on some input tape, the response is sent on the corresponding output tape. Furthermore, we typically give a name (this name is independent of the tape names) to every such pair of input and output tapes: We write "from/to $z$: $(c, c')$" to denote that the pair of tapes $(c, c')$ is named $z$. Then, we refer to the input tape $c$ by "from $z$" and to the output tape $c'$ by "to $z$". We use the generic names $\mathsf{IO}$ and $\mathsf{NET}$ to refer to general I/O and network tapes to which an environment or adversary/simulator, respectively, connect to. If the tapes connect to a known machine/system, we typically use the name of this machine/system.

For example, the ideal signature functionality $\mathcal{F}_{\text{sig}}$ (see Section 4.2.1) has the I/O input tapes $\mathsf{io}_i^{\text{in}}$ (for all $i \in \{1, \ldots, n\}$ where the number $n$ is a parameter of $\mathcal{F}_{\text{sig}}$), the network input tape $\mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{in}}$, and the corresponding output tapes $\mathsf{io}_i^{\text{out}}$ and $\mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{out}}$. We give the name $\mathsf{IO}_i$ to the pair $(\mathsf{io}_i^{\text{in}}, \mathsf{io}_i^{\text{out}})$ and the name $\mathsf{NET}$ to $(\mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{in}}, \mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{out}})$. So, "from $\mathsf{IO}_i$" refers to the tape $\mathsf{io}_i^{\text{in}}$, "to $\mathsf{NET}$" refers to $\mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{out}}$, etc.

**State.** Here, we list all state variables of the machine. These are variables that define the state of this copy of the IITM and are saved on its work tapes (i.e., they are local to the copy of the IITM and cannot be accessed by other copies). These state variables are set to some initial value when a copy of this machine is created. Typically, the initial value is $\perp$ (undefined) for bit strings and tuples of bit strings and the empty set $\emptyset$ for sets. In mode Compute, the machine may modify the values of these variables. We always use sans-serif font for state variables.

For example, all ideal functionalities that we define in this paper have a state variable $\mathsf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ which holds the corruption status of the ideal functionality.

**CheckAddress.** In this part, we define the mode CheckAddress of the machine.

**Initialization.** This part is optional. If it exists and (this copy of) the machine is activated for the first time in mode Compute, then the machine executes the code in this part. When the code finishes, the machine then processes the incoming message as defined in the part *Compute*, see below.

Initialization is used for example to tell the adversary (or simulator) that a new copy of this machine has been created and to allow her to corrupt this copy of the machine right from the start.

**Compute.** The description in mode Compute consists of a sequence of blocks where every block is of the form "**recv** $m_t$ **on** $c$ **s.t.** $\langle condition \rangle$**:** $\langle code \rangle$" where $m_t$ is an input template (see below), $c$ is an input tape (see above), $\langle condition \rangle$ is a condition on the input, and $\langle code \rangle$ is the code of this block that is executed if the input template matches and the condition is satisfied (see below).

An *input template* is recursively defined as follows: It is either an unbound variable, a constant bit string, a state variable (see above), or a tuple of input templates. We say that a bit string $m$ *matches* an input template $m_t$ if there exists a mapping $\sigma$ from the unbound variables in $m_t$ to bit strings such that $m$ equals $m_t'$ where $m_t'$ is obtained from $m_t$ by replacing every unbound variable $x$ in $m_t$ by the bit string $\sigma(x)$ and every state variable $x$ in $m_t$ by the value of the state variable (according to the state of the machine). We say that $\sigma$ is the *matcher* of $m$ and $m_t$. To distinguish unbound variables from constant bit strings and state variables, we use sans-serif font for constant bit strings and state variables and *cursive font* for unbound variables. For example, the input template $(\mathsf{Enc}, x)$ is matched by every tuple that consists of the constant bit string Enc and an arbitrary bit string.

Upon activation, the blocks are checked one after the other. The (copy of the) machine executes the code of the first block that matches the input (see below). If no block matches the input, the machine stops for this activation without producing output. In the next activation, the machine will again go through the sequence of blocks, starting with the first one, and so on.

A block, as above, *matches some input*, say message $m$ on input tape $c'$, if $c = c'$, $m$ matches $m_t$ (as defined above), and $\langle condition \rangle$ is satisfied. The condition may use state variables of the machine and the unbound variables contained in $m_t$ (these are instantiated by the matcher $\sigma$ of $m$ and $m_t$). Similarly, when executing the code, the unbound variables contained in $m_t$ are instantiated by the matcher $\sigma$ of $m$ and $m_t$.

Every execution of $\langle code \rangle$ ends with a send command: **send** $m$ **to** $c$, where $m$ is a bit string and $c$ is an output tape. This means that the machine outputs the message $m$ on tape $c$ and stops for this activation. In the next activation the machine will not proceed at the point where it stopped, but again go through the sequence of blocks, starting with the first one, as explained above. However, if the send command is followed directly by a receive command, such as **send** $m$ **on** $c$; **recv** $m_t$ **on** $c'$ **s.t.** $\langle condition \rangle$ (where $m_t$ is an input template, $c'$ an input tape, and $\langle condition \rangle$ a condition, as above), then the machine does the following: It outputs $m$ on tape $c$ and stops for this activation. In the next activation, it will check whether it received a message on input tape $c'$ and check whether this message matches $m_t$ and the condition is satisfied (as above). If it does, the computation continues at this point in the code. Otherwise, the machine stops for this activation without producing output. In the next activation, it will again check whether it received a message on input tape $c'$ and whether this message matches $m_t$ and the condition is satisfied and behaves as before, and so on, until it receives an expected message.

For named pairs of input and output tapes, as described above in the *Tapes* part, we use the following notation: Let $z$ be the name of the pair $(c, c')$ of an input tape $c$ and an output tape $c'$. Then, we write "**recv** $m_t$ **from** $z$ **s.t.** $\langle condition \rangle$" for "**recv** $m_t$ **on** $c$ **s.t.** $\langle condition \rangle$" and "**send** $m$ **to** $z$" for "**send** $m$ **on** $c'$".

#### 4.1.3 Running External Code

Sometimes, an IITM $M$ obtains the description of an algorithm $A$ as input on some tape and has to execute it (e.g., all ideal functionalities defined in this paper receive algorithms from the adversary/simulator). We write $y \leftarrow A^{(p)}(x)$, where $p$ is a polynomial, to say that $M$ simulates algorithm $A$ on input $x$ for $p(\eta + |x|)$ steps, where $\eta$ is the security parameter and $|x|$ the length of $x$. The random coins that might be used by $A$ are chosen by $M$ uniformly at random. The variable $y$ is set to the output of $A$ if $A$ terminates after at most $p(\eta + |x|)$ steps. Otherwise, $y$ is set to the error symbol $\perp$. If we want to enforce that $M$ simulates $A$ in a deterministic way, we write $y := A^{(p)}(x)$. In the simulation of $A$, $M$ sets the random coins of $A$ to zero.

Typically, we are interested in environmentally bounded systems. If such a system contains an IITM $M$ that executes external code $A$ (e.g., $A$ is provided by the adversary or simulator), then $M$ is only allowed to perform a polynomial number of steps for executing the algorithm $A$ (except with negligible probability). So, $M$ has to be parameterized by a polynomial $p$ and simulates $A$ as described above. We note that at least the degree of the polynomial that bounds the runtime of the algorithm 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.

### 4.2 Digital Signatures

In this section, we present our ideal functionality for digital signatures with local computation as explained in the introduction and show that a digital signature scheme realizes this functionality if and only if it is UF-CMA secure; see Section 6.1 for a comparison of our digital signature functionality with other functionalities in the literature.

#### 4.2.1 An Ideal Functionality $\mathcal{F}_{\text{sig}}$ for Digital Signatures

The basic idea of an ideal functionality for digital signatures is that verification only succeeds if the message has actually been signed using the functionality. This ideally prevents forgery of signatures, see, e.g., [6, 8].

**Parameters:** – $n > 0$ {*number of I/O tape pairs*
– $p$ {*polynomial that bounds the runtime of the algorithms provided by the adversary*
**Tapes:** from/to $\mathsf{IO}_i$ ($i \in \{1, \ldots, n\}$): $(\mathsf{io}_i^{\text{in}}, \mathsf{io}_i^{\text{out}})$; from/to NET: $(\mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{in}}, \mathsf{net}_{\mathcal{F}_{\text{sig}}}^{\text{out}})$
**State:** – $\mathsf{sig}, \mathsf{ver}, \mathsf{pk}, \mathsf{sk} \in \{0,1\}^* \cup \{\bot\}$ {*algorithms and key pair (provided by the adversary); initially* $\bot$
– $\mathsf{H} \subseteq \{0,1\}^*$ {*set of recorded messages; initially* $\emptyset$
– $\mathsf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ {*corruption status; initially* false
**CheckAddress:** Accept every input on every tape.
**Initialization:** Upon receiving the first message in mode Compute do:
**send** Init **to** NET; **recv** $(corrupt, sig, ver, pk, sk)$ **from** NET **s.t.** $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ { *receive algorithms and key pair from adversary, allow corruption*
$\mathsf{corrupted} := corrupt$; $\mathsf{sig} := sig$; $\mathsf{ver} := ver$; $\mathsf{pk} := pk$; $\mathsf{sk} := sk$
Then, continue processing the first request as defined below.
**Compute:**
**recv** PubKey? **from** $\mathsf{IO}_i$**: send** $(\mathsf{PubKey}, \mathsf{pk})$ **to** $\mathsf{IO}_i$ {*return public key*
**recv** $(\mathsf{Sign}, x)$ **from** $\mathsf{IO}_i$**:**
$\sigma \leftarrow \mathsf{sig}^{(p)}(\mathsf{sk}, x)$; $b := \mathsf{ver}^{(p)}(\mathsf{pk}, x, \sigma)$ {*sign* $x$ *and check that verification succeeds)*
**if** $\sigma = \bot \vee (b \neq \mathsf{true} \wedge \mathsf{corrupted} = \mathsf{false})$**: send** $(\mathsf{Signature}, \bot)$ **to** $\mathsf{IO}_i$ {*error: signing or test verification failed*
**add** $x$ **to** H; **send** $(\mathsf{Signature}, \sigma)$ **to** $\mathsf{IO}_i$ {*record* $x$ *for verification and return signature*
**recv** $(\mathsf{Verify}, pk, x, \sigma)$ **from** $\mathsf{IO}_i$**:**
$b := \mathsf{ver}^{(p)}(pk, x, \sigma)$ {*verify signature*
**if** $\mathsf{corrupted} = \mathsf{false} \wedge pk = \mathsf{pk} \wedge b = \mathsf{true} \wedge x \notin \mathsf{H}$**: send** $(\mathsf{VerResult}, \bot)$ **to** $\mathsf{IO}_i$ {*prevent forgery, return error*
**send** $(\mathsf{VerResult}, b)$ **to** $\mathsf{IO}_i$ {*return verification result*
**recv** CorrStatus? **from** $\mathsf{IO}_i$**: send** $(\mathsf{CorrStatus}, \mathsf{corrupted})$ **to** $\mathsf{IO}_i$ {*corruption status request*
**recv** Corrupt **from** NET**:** $\mathsf{corrupted} := \mathsf{true}$; **send** Corrupted **to** NET {*adaptive corruption*

Figure 3: The ideal signature functionality $\mathcal{F}_{\text{sig}}$. See Section 4.1 for notational conventions.

Our ideal signature functionality $\mathcal{F}_{\text{sig}}(n, p)$ is an IITM which is parametrized by a number $n > 0$ and a polynomial $p$. We often omit $n$ and $p$ and just write $\mathcal{F}_{\text{sig}}$ instead of $\mathcal{F}_{\text{sig}}(n, p)$. The number $n$ defines the I/O interface: for every $i \in \{1, \ldots, n\}$, $\mathcal{F}_{\text{sig}}$ has an I/O input tape and an I/O output tape. These I/O tapes allow (machines of) a protocol that uses $\mathcal{F}_{\text{sig}}$ to send requests to $\mathcal{F}_{\text{sig}}$ (and to receive the responses). For example, these tapes can be used by a protocol system that consists of $n$ machines such that the $i$-th machine connects to the $i$-th I/O input and output tape of $\mathcal{F}_{\text{sig}}$. We note that these tapes are only for addressing purposes, to allow $n$ different machines to connect to $\mathcal{F}_{\text{sig}}$; $\mathcal{F}_{\text{sig}}$ does not interpret input on different I/O tapes differently. If a request is sent on the $i$-th I/O input tape, $\mathcal{F}_{\text{sig}}$ outputs the response on the $i$-th I/O output tape.[15] Furthermore, $\mathcal{F}_{\text{sig}}$ has a network input tape and a network output tape to communicate with the adversary (or simulator). In mode CheckAddress, $\mathcal{F}_{\text{sig}}$ accepts all input on all tapes. As usual for machines that run external code (see Section 4.1.3), the polynomial $p$ bounds the runtime of the signing and verification algorithms provided by the adversary. Since every potential signing and verification algorithm has polynomial runtime, $p$ can always be chosen in such a way that the algorithms run as expected.

The functionality $\mathcal{F}_{\text{sig}}$ is defined in pseudocode in Figure 3. Upon the first request (initialization), $\mathcal{F}_{\text{sig}}$ first asks the adversary for a signature and verification algorithm, a public/private key pair, and whether it is corrupted (this allows corruption upon initialization but later corruption is allowed too, see below). We note that, when $\mathcal{F}_{\text{sig}}$ executes these algorithms, $\mathcal{F}_{\text{sig}}$ executes them as described in Section 4.1.3 where the polynomial $p$ is used to bound their runtime and the execution of the verification algorithm is forced to be deterministic. After the initialization, the first request is executed just as all later requests. We now describe the operations that $\mathcal{F}_{\text{sig}}$ provides in more detail. See also the remarks below for the typical usage of this functionality.

[15] The $n$ machines connecting to $\mathcal{F}_{\text{sig}}$ might model $n$ roles in a protocol, such as initiator and responder. However, instead of several I/O tapes one could also consider one pair of I/O tapes and use identifiers for roles, similarly to SIDs. Hence, $\mathcal{F}_{\text{sig}}$ could expect messages of the form $(r, m)$ and would return messages of the form $(r, m)$ where $r$ is an identifier for a role. In this way, one can model an arbitrary number of roles that can use $\mathcal{F}_{\text{sig}}$.

*Public key request* PubKey?*:* Upon this request on an I/O input tape, $\mathcal{F}_{\text{sig}}$ returns (on the corresponding I/O output tape) the recorded public key (provided by the adversary upon initialization). This request allows the "owner" of the public/private key pair to obtain its public key (e.g., to distribute it) and can also be used to model certain setup assumptions such as a public-key infrastructure (see the remarks below).

*Signature generation request* $(\mathsf{Sign}, x)$*:* Upon a signature generation request for a message $x$ on an I/O input tape, $\mathcal{F}_{\text{sig}}$ computes a signature for $x$ using the recorded signature generation algorithm and private key (both provided by the adversary upon initialization). Then, $\mathcal{F}_{\text{sig}}$ checks that the signature verifies (using the recorded verification algorithm and public key). If this check fails[16] and $\mathcal{F}_{\text{sig}}$ is uncorrupted (note that upon corruption, $\mathcal{F}_{\text{sig}}$ does not guarantee anything, not even that the public and private key belong together), $\mathcal{F}_{\text{sig}}$ returns an error message. Otherwise, $\mathcal{F}_{\text{sig}}$ records the message $x$ (to prevent forgery, see below) and returns the signature.

*Verification request* $(\mathsf{Verify}, pk, x, \sigma)$*:* Upon a signature verification request on an I/O input tape, $\mathcal{F}_{\text{sig}}$ verifies the signature $\sigma$ for $x$ using the provided public key $pk$ and the recorded verification algorithm (provided by the adversary). If the verification succeeds but $\mathcal{F}_{\text{sig}}$ is not corrupted, $pk$ equals the recorded public key (provided by the adversary), and $x$ has not been recorded (upon signature generation), then $\mathcal{F}_{\text{sig}}$ returns an error message. This ideally prevents forgery (if $\mathcal{F}_{\text{sig}}$ is uncorrupted and the correct public key is used) because it guarantees that signatures only verify if the message has previously been signed using $\mathcal{F}_{\text{sig}}$. Otherwise, $\mathcal{F}_{\text{sig}}$ returns the verification result.

*Corruption status request* CorrStatus?*:* Upon a corruption status request on an I/O input tape, $\mathcal{F}_{\text{sig}}$ returns its corruption status, i.e., true if it is corrupted and false otherwise.

As always in universal composability settings, the distinguishing environment should have the possibility to know which functionalities are corrupted because, otherwise, a simulator could always corrupt a functionality and then no security guarantees would be provided by the functionality. As a result, in the case of $\mathcal{F}_{\text{sig}}$, even insecure digital signature schemes would realize $\mathcal{F}_{\text{sig}}$.

*Corrupt request* Corrupt*:* Upon a corruption request on the network input tape (i.e., from the adversary), $\mathcal{F}_{\text{sig}}$ records that it is corrupted and returns an acknowledgment message (on the network output tape). This models adaptive corruption. We could have defined $\mathcal{F}_{\text{sig}}$ to output its entire state (in particularly all recorded messages) to the adversary upon corruption. However, this would only make the simulator stronger and it is not needed to realize $\mathcal{F}_{\text{sig}}$, as we will see below.

**Remarks.** As mentioned in the introduction, since signatures are determined by local computations, the signatures and the signed messages are a priori not revealed to the adversary. This, for example, is needed to reason about protocols where signatures or signed messages should remain secret.

The functionality $\mathcal{F}_{\text{sig}}$ is formulated for a single public/private key pair. The "owner" of this key pair is not made explicit in $\mathcal{F}_{\text{sig}}$ because it is irrelevant for the tasks provided by $\mathcal{F}_{\text{sig}}$. Instead, the environment has to use $\mathcal{F}_{\text{sig}}$ appropriately, i.e., only the party that "owns" this key pair should be allowed to send Sign requests. Of course, every party should be allowed to send Verify requests. If parties other than the "owner" are allowed to send PubKey? requests to $\mathcal{F}_{\text{sig}}$, then this models that the public key is distributed among the parties, e.g., by some kind of a public-key infrastructure.

To make the "owner" explicit and to obtain multiple instances of $\mathcal{F}_{\text{sig}}$, one can consider the system $!\underline{\mathcal{F}_{\text{sig}}}$, where $\underline{\mathcal{F}_{\text{sig}}}$ is the multi-party version of $\mathcal{F}_{\text{sig}}$. Recall that for every PID $pid$ (in a run of $!\underline{\mathcal{F}_{\text{sig}}}$ with some environment) there can be a copy of $\underline{\mathcal{F}_{\text{sig}}}$. Let us denote this copy by $\mathcal{F}_{\text{sig}[pid]}$. The "owner" of $\mathcal{F}_{\text{sig}[pid]}$ is

[16] Note that every reasonable digital signature scheme satisfies that this check never fails. However, as we do not put any restrictions on the algorithms provided by the adversary, $\mathcal{F}_{\text{sig}}$ does not know whether they have this property. This test guarantees that every verification request to $\mathcal{F}_{\text{sig}}$ succeeds for signatures that have been created by $\mathcal{F}_{\text{sig}}$ (if the correct message and public key are provided upon verification).

the party with PID $pid$. Every message sent to/received from this copy is prefixed with $pid$. For example, if a party wants to verify a message, it would send a message of the form $(pid, (\mathsf{Verify}, pk, x, \sigma))$ to $\mathcal{F}_{\text{sig}[pid]}$. Only the owner of $\mathcal{F}_{\text{sig}[pid]}$ should have unrestricted access to all commands provided by $\mathcal{F}_{\text{sig}[pid]}$. Other parties should, for example, not be able to issue signing requests to $\mathcal{F}_{\text{sig}[pid]}$. As mentioned, this should be guaranteed by the protocols that use $\mathcal{F}_{\text{sig}[pid]}$.

Alternatively, one can restrict access to $\mathcal{F}_{\text{sig}}$ in the following way. One can add a wrapper $M_{access}$ that controls access to $\underline{\mathcal{F}_{\text{sig}}}$. The machine $M_{access}$ connects to all I/O tapes of $\underline{\mathcal{F}_{\text{sig}}}$ and has the same number of I/O tapes for connecting to the environment. An instance of $M_{access}$ expects inputs that are prefixed with two PIDs $(pid_{sender}, pid_{owner})$, where $pid_{sender}$ is the ID of the sender of the input and $pid_{owner}$ is the ID of the owner of the instance $\mathcal{F}_{\text{sig}[pid_{owner}]}$. Now, $M_{access}$ forwards verification requests to $\mathcal{F}_{\text{sig}[pid_{owner}]}$ for any combination of $pid_{sender}$ and $pid_{owner}$, however, signing requests are blocked unless $pid_{sender} = pid_{owner}$. All responses from $\mathcal{F}_{\text{sig}}$ are returned via $M_{access}$ to the environment. Thus, a higher level protocol can only sign messages in the name of their own PID, but verify messages for any PID. Note that this assumes that higher-level protocols are defined in such a way that parties cannot lie about their PIDs at the I/O interface.

The multi-session, multi-party version of $\mathcal{F}_{\text{sig}}$ can be described by the system $!\underline{\underline{\mathcal{F}_{\text{sig}}}}$. In this system, to address different copies of $\mathcal{F}_{\text{sig}}$ all messages are prefixed by a SID and a PID.

It is easy to see that $\mathcal{F}_{\text{sig}}$ is environmentally strictly bounded. Hence, by Lemma 2, both the multi-party version $!\underline{\mathcal{F}_{\text{sig}}}$ and the multi-session, multi-party version $!\underline{\underline{\mathcal{F}_{\text{sig}}}}$ are environmentally strictly bounded.

#### 4.2.2 Realizing $\mathcal{F}_{\text{sig}}$ by UF-CMA Secure Digital Signature Schemes

In this section, we show that a (digital) signature scheme (more precisely, the protocol system induced by it) realizes the ideal signature functionality $\mathcal{F}_{\text{sig}}$ if and only if the signature scheme is UF-CMA secure (unforgeability under chosen-message attacks). UF-CMA security is a standard security notion for signature schemes, see, e.g., [16]. We recall the definition of signature schemes and UF-CMA security in Appendix A.1.

Every signature scheme $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ induces a realization $\mathcal{P}_{\text{sig}}(n, \Sigma)$ of $\mathcal{F}_{\text{sig}}(n, p)$ (where $p$ depends on $\Sigma$) in a straightforward way. This realization is defined as follows (see also Figure 9 in the appendix): Upon initialization (i.e., when receiving the first message), $\mathcal{P}_{\text{sig}}$ asks the adversary whether it is corrupted. If the adversary decides to corrupt $\mathcal{P}_{\text{sig}}$ upon initialization, she provides a public/private key pair. Otherwise, $\mathcal{P}_{\text{sig}}$ generates a fresh key pair itself (using $\mathsf{gen}$). The key pair, say $(pk, sk)$, is recorded in $\mathcal{P}_{\text{sig}}$. The adversary can also corrupt $\mathcal{P}_{\text{sig}}$ adaptively by sending the message $\mathsf{Corrupt}$ to $\mathcal{P}_{\text{sig}}$ upon which $\mathcal{P}_{\text{sig}}$ returns the recorded key pair $(pk, sk)$. Upon a signature generation requests of the form $(\mathsf{Sign}, x)$ from some party (i.e., from the environment on an I/O input tape), $\mathcal{P}_{\text{sig}}$ computes a signature $\sigma \leftarrow \mathsf{sig}(sk, x)$ and returns $\sigma$. Upon a signature verification requests of the form $(\mathsf{Verify}, pk', x, \sigma)$ from some party, $\mathcal{P}_{\text{sig}}$ verifies the signature, $b := \mathsf{ver}(pk', x, \sigma)$, and returns $b$. Upon a public key request ($\mathsf{PubKey}?$) from some party, $\mathcal{P}_{\text{sig}}$ returns the recorded public key $pk$. Upon a corruption status request from some party, $\mathcal{P}_{\text{sig}}$ returns $\mathsf{true}$ if it has been corrupted by the adversary (upon initialization or by receiving the message $\mathsf{Corrupt}$) and $\mathsf{false}$ otherwise. It is easy to see that $\mathcal{P}_{\text{sig}}$ is environmentally strictly bounded.

We obtain the following theorem. A proof is provided in Appendix B.1. The proof is similar to other proofs for realizations of digital signatures [1, 6, 8].

**Theorem 6.** *Let $n > 0$, $\Sigma$ be a signature scheme, and $p$ be a polynomial that bounds the runtime of the algorithms in $\Sigma$ (in the length of their inputs). Then, $\Sigma$ is UF-CMA secure if and only if $\mathcal{P}_{\text{sig}}(n, \Sigma) \leq \mathcal{F}_{\text{sig}}(n, p)$.*

### 4.3 Public-Key Encryption

In this section, we present our ideal functionality for public-key encryption with local computation as explained in the introduction. Our functionality is parametrized by what we call a *leakage algorithm* which allows to define the amount of information that may be leaked by the encryption. We first define and discuss leakage algorithms. Then, we present our ideal public-key encryption functionality and show that a public-key encryption scheme realizes this functionality (given an appropriate leakage algorithm) if and only if it is

IND-CCA2 secure; see Section 6.2 for a comparison of our public-key encryption functionality with other functionalities in the literature.

### 4.3.1 Leakage Algorithms

We now introduce leakage algorithms that are used by our ideal functionality for public-key encryption. In this functionality, instead of the actual plaintext, its *leakage* is encrypted. The leakage is computed by a leakage algorithm and captures the amount of information that may be leaked about the plaintexts even in the ideal setting.

**Definition 4.** Let $D = \{D(\eta)\}_{\eta\in\mathbb{N}}$ with $D(\eta) \subseteq \{0,1\}^*$ for all $\eta \in \mathbb{N}$ be a polynomial-time decidable domain of plaintexts.[17] A *leakage algorithm $L$ with domain $D$* is a probabilistic, polynomial-time algorithm that takes as input $1^\eta$ for some security parameter $\eta \in \mathbb{N}$ and a plaintext $x \in D(\eta)$ and returns a bit string $\overline{x} \in D(\eta)$, the leakage of $x$.

The plaintext domain associated with a leakage algorithm $L$ is denoted by $D_L$.

**Example 1.** Typical examples of leakage algorithms are *(i)* $L(1^\eta, x) := 0^{|x|}$ (for all $\eta, x$) and *(ii)* the algorithm that returns a random bit string of length $|x|$. They both leak exactly the length of a plaintext. The domain of these leakage algorithms is the domain of all bit strings.

We sometimes require leakage algorithms to have some of the following properties.

**Definition 5.** We call a leakage algorithm $L$ *length preserving* if $|\overline{x}| = |x|$ for every $\eta \in \mathbb{N}$, $x \in D_L(\eta)$, and leakage $\overline{x}$ produced by $L(1^\eta, x)$.

We say that a leakage algorithm leaks at most the length of a plaintext if the leakage of a plaintext does not reveal any information about the actual bits of the plaintext. Formally, this is defined as follows:

**Definition 6.** A leakage algorithm $L$ *leaks at most the length of a plaintext* if there exists a probabilistic, polynomial-time algorithm $T$ such that, for every $\eta \in \mathbb{N}$ and $x \in D_L(\eta)$, the probability distributions of $T(1^\eta, 1^{|x|})$ and $L(1^\eta, x)$ are equal (i.e., $\Pr[T(1^\eta, 1^{|x|}) = \overline{x}] = \Pr[L(1^\eta, x) = \overline{x}]$ for every $\overline{x} \in \{0,1\}^*$, where the probability is over the random coins of $T$ and $L$, respectively).

**Definition 7.** We say that a leakage algorithm $L$ *leaks exactly the length of a plaintext* if it is length preserving and leaks at most the length of a plaintext.

We say that a leakage algorithm has high entropy if collisions of the leakage occur only with negligible probability.

**Definition 8.** A leakage algorithm $L$ has *high entropy* if the probability of collisions, i.e.,

$$\sup_{x,x'\in D_L(\eta)} \Pr\left[\overline{x} \leftarrow L(1^\eta, x), \overline{x}' \leftarrow L(1^\eta, x') : \overline{x} = \overline{x}'\right]$$

(the probability is over the random coins of $L$) is negligible (as a function in $\eta$).

For example, both leakage algorithms from Example 1 are length preserving and leak at most the length of a plaintext (for any plaintext domain), i.e., they leak exactly the length of a plaintext. Moreover, the second leakage algorithm, which returns a random bit string of the same length as the plaintext, has high entropy if its plaintext domain only contains "long" plaintexts, e.g., only bit strings of length $\geq \eta$.[18] We note that deterministic leakage algorithms (e.g., the first leakage algorithm from Example 1, which returns a constant bit string of the same length as the plaintext) do not have high entropy if they are associated with any non-empty domain of plaintexts.

[17] That is, there exists a deterministic algorithm $A$ and a polynomial $p$ such that, for all $\eta \in \mathbb{N}$ and $x \in \{0,1\}^*$, $A(1^\eta, x) = 1$ iff $x \in D(\eta)$ and the runtime of $A(1^\eta, x)$ is bounded from above by $p(\eta + |x|)$.

[18] For this leakage algorithm, the probability of collisions of leakages is $2^{-l}$ for plaintexts that have the same length $l$ and 0 if they are of different length. Hence, for a plaintext domain $\{D(\eta)\}_{\eta\in\mathbb{N}}$ with $|x| \geq \eta$ for all $\eta \in \mathbb{N}$ and $x \in D(\eta)$, the probability in Definition 8 is at most $2^{-\eta}$, which is negligible.

### 4.3.2 An Ideal Functionality $\mathcal{F}_{\text{pke}}$ for Public-Key Encryption

Our ideal functionality $\mathcal{F}_{\text{pke}}$ for public-key encryption with local computation is in the spirit of the one proposed by Canetti in [6] (version of December 2005) in that, other than providing an encryption and decryption algorithm as well as a public/private key pair (Canetti does not distinguish between a public key and the encryption algorithm), the simulator is not involved in the execution of the functionality. In particular, all ciphertexts and decryptions are performed locally within the functionality. However, our formulation differs in essential ways from the one by Canetti, e.g., Canetti's formulation is not suitable for joint state realizations (see Section 6.2).

We now present our ideal public-key encryption functionality $\mathcal{F}_{\text{pke}}(n, p, L)$. In many technical matters the formulation is similar to $\mathcal{F}_{\text{sig}}$. The IITM $\mathcal{F}_{\text{pke}}(n, p, L)$ is parametrized by a number $n > 0$, a polynomial $p$, and a leakage algorithm $L$. We often omit some or all parameters if they are clear from the context and, for example, just write $\mathcal{F}_{\text{pke}}$ instead of $\mathcal{F}_{\text{pke}}(n, p, L)$. Just like for $\mathcal{F}_{\text{sig}}$, $n$ determines the I/O interface of $\mathcal{F}_{\text{pke}}$ and $p$ bounds the runtime of the encryption and decryption algorithms provided by the adversary. Since every potential encryption and decryption algorithm has polynomial runtime, $p$ can always be chosen in such a way that the algorithms run as expected. Furthermore, just like $\mathcal{F}_{\text{sig}}$, $\mathcal{F}_{\text{pke}}$ has a network input and output tape to communicate with the adversary (or simulator).

The functionality $\mathcal{F}_{\text{pke}}$ is defined in pseudocode in Figure 4. We now describe the operations that $\mathcal{F}_{\text{pke}}$ provides in more detail. Upon the first request (initialization), $\mathcal{F}_{\text{pke}}$ first asks the adversary for an encryption and decryption algorithm, a public/private key pair, and whether it is corrupted (this models static corruption). We note that, when $\mathcal{F}_{\text{pke}}$ executes these algorithms, $\mathcal{F}_{\text{pke}}$ executes them as described in Section 4.1.3 where the polynomial $p$ is used to bound their runtime and the execution of the decryption algorithm is forced to be deterministic. After the initialization, the first request is executed just as all later requests.

*Public key request* $\mathsf{PubKey}?$*:* Just as $\mathcal{F}_{\text{sig}}$, upon this request on an I/O input tape, $\mathcal{F}_{\text{pke}}$ returns the recorded public key (on the corresponding I/O output tape). This request allows the "owner" of the public/private key pair to obtain its public key (e.g., to distribute it) and can also be used to model certain setup assumptions such as a public-key infrastructure (see the remarks below).

*Encryption request* $(\mathsf{Enc}, pk, x)$*:* Upon an encryption request for a plaintext $x \in D_L(\eta)$ (recall that $D_L = \{D_L(\eta)\}_{\eta \in \mathbb{N}}$ is the plaintext domain associated with the leakage algorithm $L$) under a public key $pk$ on an I/O input tape, $\mathcal{F}_{\text{pke}}$ does the following. If $\mathcal{F}_{\text{pke}}$ is corrupted or $pk$ is not the recorded public key (that has been provided by the adversary upon initialization), $\mathcal{F}_{\text{pke}}$ encrypts $x$ under $pk$ (using the encryption algorithm provided by the adversary upon initialization) and returns the ciphertext. Otherwise, $\mathcal{F}_{\text{pke}}$ generates the ciphertext by encrypting the leakage $\overline{x} \leftarrow L(1^\eta, x)$ of $x$. Then, $\mathcal{F}_{\text{pke}}$ checks that the decryption of the ciphertext yields the leakage $\overline{x}$ again. If this check fails, $\mathcal{F}_{\text{pke}}$ returns an error message. Otherwise, $\mathcal{F}_{\text{pke}}$ records the message $x$ for that ciphertext (for later decryption) and returns the ciphertext.

We note that, every reasonable encryption scheme satisfies that the decryption of the encryption yields the plaintext again. However, as we do not put any restrictions on the algorithms provided by the adversary, $\mathcal{F}_{\text{pke}}$ does not know whether they have this property. In the remarks below we explain why the decryption test performed by $\mathcal{F}_{\text{pke}}$ is useful and sometimes needed. In particularly, it is needed for our joint state realization, see Section 5.2.

*Decryption request* $(\mathsf{Dec}, y)$*:* Upon a decryption request for a ciphertext $y$ on an I/O input tape, $\mathcal{F}_{\text{pke}}$ does the following. If $\mathcal{F}_{\text{pke}}$ is corrupted or there is no recorded message for $y$, $\mathcal{F}_{\text{pke}}$ decrypts $y$ using the recorded private key and decryption algorithm (both provided by the adversary upon initialization) and returns the resulting plaintext. Otherwise, the plaintext that is recorded for $y$ is returned (an error message is returned if there is more than one recorded plaintext for $y$ because unique decryption is not possible in this case).

**Parameters:** – $n > 0$ {*number of I/O tape pairs*
– $p$ {*polynomial that bounds the runtime of the algorithms provided by the adversary*
– $L$ {*leakage algorithm with associated plaintext domain* $D_L = \{D_L(\eta)\}_{\eta\in\mathbb{N}}$
**Tapes:** from/to $\mathsf{IO}_i$ ($i \in \{1,\ldots,n\}$): $(\mathsf{io}_i^{\text{in}}, \mathsf{io}_i^{\text{out}})$; from/to NET: $(\mathsf{net}^{\text{in}}_{\mathcal{F}_{\text{pke}}}, \mathsf{net}^{\text{out}}_{\mathcal{F}_{\text{pke}}})$
**State:** – $\mathsf{enc}, \mathsf{dec}, \mathsf{pk}, \mathsf{sk} \in \{0,1\}^* \cup \{\perp\}$ {*algorithms and key pair (provided by the adversary); initially* $\perp$
– $\mathsf{H} \subseteq \{0,1\}^* \times \{0,1\}^*$ {*set of recorded plaintext/ciphertext pairs; initially* $\emptyset$
– $\mathsf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ {*corruption status; initially* false
**CheckAddress:** Accept every input on every tape.
**Initialization:** Upon receiving the first message in mode Compute do: { *receive algorithms and key pair from adversary, allow corruption*
**send** Init **to** NET; **recv** $(corrupt, enc, dec, pk, sk)$ **from** NET **s.t.** $corrupt \in \{\mathsf{false}, \mathsf{true}\}$
$\mathsf{corrupted} := corrupt$; $\mathsf{enc} := enc$; $\mathsf{dec} := dec$; $\mathsf{pk} := pk$; $\mathsf{sk} := sk$
Then, continue processing the first request as defined below.
**Compute:**
**recv** PubKey? **from** $\mathsf{IO}_i$**:** **send** $(\mathsf{PubKey}, \mathsf{pk})$ **to** $\mathsf{IO}_i$ {*return public key*
**recv** $(\mathsf{Enc}, pk, x)$ **from** $\mathsf{IO}_i$ **s.t.** $x \in D_L(\eta)$**:**
**if** $\mathsf{corrupted} = \mathsf{false} \wedge pk = \mathsf{pk}$**:** {*i.e., uncorrupted and correct public key*
$\overline{x} \leftarrow L(1^\eta, x)$; $y \leftarrow \mathsf{enc}^{(p)}(\mathsf{pk}, \overline{x})$; $\overline{x}' := \mathsf{dec}^{(p)}(\mathsf{sk}, y)$ {*encrypt leakage of x, test if decryption yields leakage again*
**if** $y = \perp \vee \overline{x}' \neq \overline{x}$**:** **send** $(\mathsf{Ciphertext}, \perp)$ **to** $\mathsf{IO}_i$ {*error: encryption or decryption test failed*
**add** $(x, y)$ **to** H; **send** $(\mathsf{Ciphertext}, y)$ **to** $\mathsf{IO}_i$ {*record* $(x, y)$ *for decryption, return y*
**else:** {*i.e., corrupted or wrong public key*
$y \leftarrow \mathsf{enc}^{(p)}(pk, x)$; **send** $(\mathsf{Ciphertext}, y)$ **to** $\mathsf{IO}_i$ {*encrypt x, return y*
**recv** $(\mathsf{Dec}, y)$ **from** $\mathsf{IO}_i$**:**
**if** $\mathsf{corrupted} = \mathsf{false} \wedge \exists x\colon (x, y) \in \mathsf{H}$**:**
**if** $\exists x, x'\colon x \neq x' \wedge (x, y), (x', y) \in \mathsf{H}$**:** **send** $(\mathsf{Plaintext}, \perp)$ **to** $\mathsf{IO}_i$ {*error: unique decryption not possible*
**let** $x$ **s.t.** $(x, y) \in \mathsf{H}$ {*in this case, such an x always exists and is unique*
**send** $(\mathsf{Plaintext}, x)$ **to** $\mathsf{IO}_i$ {*return x*
**else:**
$x := \mathsf{dec}^{(p)}(\mathsf{sk}, y)$; **send** $(\mathsf{Plaintext}, x)$ **to** $\mathsf{IO}_i$ {*decrypt y, return x*
**recv** CorrStatus? **from** $\mathsf{IO}_i$**:** **send** $(\mathsf{CorrStatus}, \mathsf{corrupted})$ **to** $\mathsf{IO}_i$ {*corruption status request*

Figure 4: The ideal public-key encryption functionality $\mathcal{F}_{\text{pke}}$. See Section 4.1 for notational conventions.

*Corruption status request* CorrStatus?*:* Just as $\mathcal{F}_{\text{sig}}$, upon a corruption status request on an I/O input tape, $\mathcal{F}_{\text{pke}}$ returns true if it is corrupted and false otherwise; see the description of $\mathcal{F}_{\text{sig}}$ for a discussion on corruption status requests.

**Remarks.** The same remarks for $\mathcal{F}_{\text{sig}}$ (see Section 4.2.1) apply also to $\mathcal{F}_{\text{pke}}$: It is left to the environment to use $\mathcal{F}_{\text{pke}}$ appropriately, i.e., only the "owner" of the public/private key pair should use $\mathcal{F}_{\text{pke}}$ to decrypt messages. Alternatively, one can use a wrapper similar to $M_{access}$ for $\mathcal{F}_{\text{sig}}$ to control encryption and decryption requests. As mentioned for $\mathcal{F}_{\text{sig}}$, a multi-party version of $\mathcal{F}_{\text{pke}}$ where every party (with PID) *pid* owns one copy of $\mathcal{F}_{\text{pke}}$ can be modeled by the system $!\underline{\mathcal{F}_{\text{pke}}}$ and a multi-session, multi-party version of $\mathcal{F}_{\text{pke}}$ can be modeled by $!\underline{\underline{\mathcal{F}_{\text{pke}}}}$.

If $\mathcal{F}_{\text{pke}}(L)$ is used with a leakage algorithm $L$ with high entropy, then an uncorrupted $\mathcal{F}_{\text{pke}}$ guarantees that ciphertexts stored in H cannot be guessed. For example, if one ciphertext, say $y$, is given to the adversary only encrypted (nested encryption), then the adversary is not able to guess $y$. The reason that $\mathcal{F}_{\text{pke}}(L)$ has this property, provided that $L$ has high entropy, is as follows: the ciphertext has to contain as much information as the leakage $L(1^\eta, x)$, because of the decryption test performed in $\mathcal{F}_{\text{pke}}(L)$ (decryption of a ciphertext must yield the original plaintext). Since the leakage has high entropy, $L(1^\eta, x)$ is sufficiently random and can be guessed only with negligible probability.

It can be shown that a realization of $\mathcal{F}_{\text{pke}}$ is impossible if it is adaptively corruptible [25]. Therefore, our formulation of $\mathcal{F}_{\text{pke}}$, unlike $\mathcal{F}_{\text{sig}}$, only allows for corruption upon initialization.

It is easy to see that $\mathcal{F}_{\text{pke}}$ is environmentally strictly bounded. Hence, by Lemma 2, both the multi-party version $!\underline{\mathcal{F}_{\text{pke}}}$ and the multi-session, multi-party version $!\underline{\underline{\mathcal{F}_{\text{pke}}}}$ are environmentally strictly bounded.

#### 4.3.3 Realizing $\mathcal{F}_{\text{pke}}$ by IND-CCA2 Secure Public-Key Encryption Schemes

In this section, we show that a public-key encryption scheme realizes the ideal public-key encryption functionality $\mathcal{F}_{\text{pke}}$ (given an appropriate leakage algorithm) if and only if the encryption scheme is IND-CCA2 secure (indistinguishability under chosen-ciphertext attacks). IND-CCA2 security is a standard security notion for public-key encryption schemes, see, e.g., [3, 4]. We recall the definition of public-key encryption schemes and IND-CCA2 security in Appendix A.2. Similar to leakage algorithms, we assume that every public-key encryption scheme $\Sigma$ is associated with a polynomial-time decidable domain of plaintexts $D_\Sigma = \{D_\Sigma(\eta)\}_{\eta\in\mathbb{N}}$ for some $D_\Sigma(\eta) \subseteq \{0,1\}^*$ for every security parameter $\eta \in \mathbb{N}$.

Every public-key encryption scheme $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ induces in a straightforward way a realization $\mathcal{P}_{\text{pke}}(n, \Sigma)$ of $\mathcal{F}_{\text{pke}}$. The realization $\mathcal{P}_{\text{pke}}(n, \Sigma)$ is defined in Figure 10 (in the appendix). Informally, it is described as follows: Upon initialization (i.e., when the first message is received), $\mathcal{P}_{\text{pke}}$ asks the adversary whether it is corrupted. If the adversary decides to corrupt $\mathcal{P}_{\text{pke}}$ upon initialization, he provides a public/private key pair. Otherwise, $\mathcal{P}_{\text{pke}}$ generates a fresh key pair itself (using $\mathsf{gen}$). The key pair, say $(pk, sk)$, is recorded in $\mathcal{P}_{\text{pke}}$. As already mentioned above, $\mathcal{F}_{\text{pke}}$ is not realizable under adaptive corruption due to the commitment problem [25]. Therefore, the adversary can only corrupt $\mathcal{P}_{\text{pke}}$ upon initialization. Upon an encryption requests of the form $(\mathsf{Enc}, pk', x)$ with $x \in D_\Sigma(\eta)$ from some party (i.e., from the environment on an I/O input tape), $\mathcal{P}_{\text{pke}}$ computes the ciphertext $y \leftarrow \mathsf{enc}(pk', x)$ and returns $y$. Upon a decryption requests of the form $(\mathsf{Dec}, y)$ from some party, $\mathcal{P}_{\text{pke}}$ computes the plaintext $x := \mathsf{dec}(sk, x)$ (where $sk$ is the recorded private key) and returns $x$. Upon a public key request ($\mathsf{PubKey}?$) from some party, $\mathcal{P}_{\text{pke}}$ returns the recorded public key $pk$. Upon a corruption status request from some party, $\mathcal{P}_{\text{pke}}$ returns $\mathsf{true}$ if it has been corrupted by the adversary upon initialization and $\mathsf{false}$ otherwise. It is easy to see that $\mathcal{P}_{\text{pke}}$ is environmentally strictly bounded.

The following theorem shows that $\mathcal{F}_{\text{pke}}(L)$ exactly captures the standard security notion IND-CCA2, if the leakage algorithm leaks exactly the length of a plaintext (Definition 7). A proof of the following theorem is provided in Appendix B.2.

**Theorem 7.** *Let $n > 0$, $\Sigma$ be a public-key encryption scheme, $p$ be a polynomial that bounds the runtime of the algorithms in $\Sigma$ (in the length of their inputs), and $L$ be a leakage algorithm such that $D_\Sigma = D_L$ (i.e., $\Sigma$ and $L$ have the same plaintext domain) and $L$ leaks exactly the length of a plaintext (e.g., $L$ is one of the algorithms from Example 1). Then, $\Sigma$ is IND-CCA2 secure if and only if $\mathcal{P}_{\text{pke}}(n, \Sigma) \leq \mathcal{F}_{\text{pke}}(n, p, L)$.*

*The direction from left to right holds for any length preserving leakage algorithm $L$ and the direction from right to left holds for any leakage algorithm $L$ that leaks at most the length of a plaintext.*

We note that Bellare et al. [4] define two security notions for public-key encryption schemes (namely IND-CCA-BP and IND-CCA-BE) that are shown to be strictly weaker than IND-CCA2 security (which is called IND-CCA-SE in the taxonomy of [4]). Theorem 7 now shows that these weaker notions do not suffice to realize $\mathcal{F}_{\text{pke}}$ (if $L$ leaks at most the length of a plaintext).

### 4.4 Replayable Public-Key Encryption

In this section, we present our replayable public-key encryption functionality with local computation, as explained in the introduction, and show that a public-key encryption scheme realizes this functionality (given an appropriate leakage algorithm) if and only if it is IND-RCCA secure. We refer to Section 6.3 for a comparison of our replayable public-key encryption functionality with other functionalities in the literature.

#### 4.4.1 An Ideal Functionality $\mathcal{F}_{\text{rpke}}$ for Replayable Public-Key Encryption

Our ideal functionality $\mathcal{F}_{\text{rpke}}$ with local computation for replayable public-key encryption is defined as follows.

The functionality $\mathcal{F}_{\text{rpke}}(n, p, L)$ (or $\mathcal{F}_{\text{rpke}}$ for short) is, just as $\mathcal{F}_{\text{pke}}$, parametrized by a number $n > 0$ which defines the I/O interface, a polynomial $p$ that bounds the runtime of the algorithms provided by

**Parameters, Tapes, State, CheckAddress, Initialization:** Just like $\mathcal{F}_{\text{pke}}$, see Figure 4.

**Compute:**

**recv** PubKey? **from** $\mathsf{IO}_i$**: send** $(\mathsf{PubKey}, \mathsf{pk})$ **to** $\mathsf{IO}_i$ {*return public key*

**recv** $(\mathsf{Enc}, pk, x)$ **from** $\mathsf{IO}_i$ **s.t.** $x \in D(\eta)$**:**
- **if** $\mathsf{corrupted} = \mathsf{false} \wedge pk = \mathsf{pk}$**:** {*i.e., uncorrupted and correct public key*
  - $\overline{x} \leftarrow L(1^\eta, x)$; $y \leftarrow \mathsf{enc}^{(p)}(\mathsf{pk}, \overline{x})$; $\overline{x}' := \mathsf{dec}^{(p)}(\mathsf{sk}, y)$ {*encrypt leakage of $x$, test if decryption yields leakage again)*
  - **if** $y = \bot \vee \overline{x}' \neq \overline{x}$**: send** $(\mathsf{Ciphertext}, \bot)$ **to** $\mathsf{IO}_i$ {*error: encryption or decryption test failed*
  - **add** $(x, \overline{x})$ **to** H; **send** $(\mathsf{Ciphertext}, y)$ **to** $\mathsf{IO}_i$ {*record $(x, y)$ for decryption, return $y$*
- **else:** {*i.e., corrupted or wrong public key*
  - $y \leftarrow \mathsf{enc}^{(p)}(pk, x)$; **send** $(\mathsf{Ciphertext}, y)$ **to** $\mathsf{IO}_i$ {*encrypt $x$, return $y$*

**recv** $(\mathsf{Dec}, y)$ **from** $\mathsf{IO}_i$**:**
- $\overline{x} := \mathsf{dec}^{(p)}(\mathsf{sk}, y)$ {*decrypt $y$*
- **if** $\mathsf{corrupted} = \mathsf{false} \wedge \exists x\colon (x, \overline{x}) \in \mathsf{H}$**:**
  - **if** $\exists x, x'\colon x \neq x' \wedge (x, \overline{x}), (x', \overline{x}) \in \mathsf{H}$**: send** $(\mathsf{Plaintext}, \bot)$ **to** $\mathsf{IO}_i$ {*error: unique decryption not possible*
  - **let** $x$ **s.t.** $(x, \overline{x}) \in \mathsf{H}$ {*in this case, such an $x$ always exists and is unique*
  - **send** $(\mathsf{Plaintext}, x)$ **to** $\mathsf{IO}_i$ {*return $x$*
- **else:**
  - **send** $(\mathsf{Plaintext}, \overline{x})$ **to** $\mathsf{IO}_i$ {*return $\overline{x}$*

**recv** CorrStatus? **from** $\mathsf{IO}_i$**: send** $(\mathsf{CorrStatus}, \mathsf{corrupted})$ **to** $\mathsf{IO}_i$ {*corruption status request*

Figure 5: The ideal functionality $\mathcal{F}_{\text{rpke}}$ for replayable public-key encryption. See Section 4.1 for notational conventions.

the adversary (or simulator), and a leakage algorithm $L$. A definition of $\mathcal{F}_{\text{rpke}}$ in pseudocode is given in Figure 5. The only difference between $\mathcal{F}_{\text{rpke}}$ and $\mathcal{F}_{\text{pke}}$ is that upon encryption of a plaintext $x$, the pair $(x, \overline{x})$ (where $\overline{x} \leftarrow L(1^\eta, x)$ is the leakage of $x$) is stored instead of $(x, y)$ (where $y$ is the ciphertext) and that upon decryption it is not looked for the ciphertext $y$ but for the decryption $\mathsf{dec}(\mathsf{sk}, y)$ of the ciphertext. Hence, it might be possible for an adversary to produce a ciphertext $y' \neq y$ such that the decryption $x$ of $y'$ is the same as the one of $y$, without knowing $x$. This models replayable encryption.

We note that the decryption test upon encryption (to test that the decryption yields the leakage again) is not needed for the joint state theorem for $\mathcal{F}_{\text{rpke}}$ (see below), so, it could be omitted. However, it is sometimes useful, e.g., when reasoning about protocols with nested encryption, as discussed for $\mathcal{F}_{\text{pke}}$.

It is easy to see that $\mathcal{F}_{\text{rpke}}$ is environmentally strictly bounded. Hence, just as for $\mathcal{F}_{\text{pke}}$, by Lemma 2, both the multi-party version $!\underline{\mathcal{F}_{\text{rpke}}}$ and the multi-session, multi-party version $!\underline{\underline{\mathcal{F}_{\text{rpke}}}}$ are environmentally strictly bounded.

### 4.4.2 Realizing $\mathcal{F}_{\text{rpke}}$ by IND-RCCA Secure Public-Key Encryption Schemes

We now show that a public-key encryption scheme realizes the ideal replayable public-key encryption functionality $\mathcal{F}_{\text{rpke}}$ (given an appropriate leakage algorithm) if and only if the encryption scheme is IND-RCCA (replayable IND-CCA2) secure. IND-RCCA security, which has been introduced by Canetti et al. [12], is a relaxed form of IND-CCA2 security where modifications of the ciphertext that yield the same plaintext are permitted. In particular, IND-CCA2 security implies IND-RCCA security [12]. As explained by Canetti et al., IND-RCCA security suffices in many applications where IND-CCA2 security is used. We recall the definition of public-key encryption schemes and IND-RCCA security in Appendix A.2. As mentioned above, similar to leakage algorithms, we assume that every public-key encryption scheme $\Sigma$ is associated with a polynomial-time decidable domain of plaintexts $D_\Sigma$.

The realization $\mathcal{P}_{\text{pke}}(n, \Sigma)$ (see Section 4.3.3) of $\mathcal{F}_{\text{rpke}}$ is the same as for $\mathcal{F}_{\text{pke}}$; only the requirements on $\Sigma$ are milder, namely IND-RCCA security instead of IND-CCA2 security.

The following theorem shows that $\mathcal{F}_{\text{rpke}}(L)$ exactly captures IND-RCCA security if the leakage algorithm $L$ leaks exactly the length of a plaintext (Definition 7) and has high entropy (Definition 8). For example, this condition on $L$ is satisfied if $L$ is the leakage algorithm that returns a random bit string of the length of

the plaintext and the domain of plaintexts only contains "long" plaintexts, e.g., only plaintexts of length $\geq \eta$ (where $\eta$ is the security parameter), see Section 4.3.1. A proof of the following theorem is provided in Appendix B.3.

**Theorem 8.** *Let $n > 0$, $\Sigma$ be a public-key encryption scheme, $p$ be a polynomial that bounds the runtime of the algorithms in $\Sigma$ (in the length of their inputs), and $L$ be a leakage algorithm such that $D_\Sigma = D_L$ (i.e., $\Sigma$ and $L$ have the same plaintext domain) and $L$ leaks exactly the length of a plaintext and has high entropy. Then, $\Sigma$ is IND-RCCA secure if and only if $\mathcal{P}_{\text{pke}}(n, \Sigma) \leq \mathcal{F}_{\text{rpke}}(n, p, L)$.*

*The direction from left to right holds for any length preserving leakage algorithm $L$ that has high entropy and the direction from right to left holds for any leakage algorithm $L$ that leaks at most the length of a plaintext.*

We note that if a length preserving leakage algorithm has high entropy, then its domain of plaintexts contains only "long" plaintexts (e.g., only plaintexts of length $\geq \eta$ for security parameter $\eta$). So, our result is consistent with the result by Canetti et al. [12], where large plaintext domains are assumed. We further remark that Canetti et al. showed that IND-RCCA security is not sufficient to realize $\mathcal{F}_{\text{rpke}}$ if plaintext domains have only polynomial size.

# 5 Joint State Realizations

In this section, we present joint state realizations of the ideal functionalities presented in the previous section, i.e., for digital signatures, public-key encryption, and replayable public-key encryption. We refer to Section 6 for a comparison of our joint state theorems with others proposed in the literature. The explanations given in Section 6 will also motivate and justify the definitions of our functionalities and the way our joint state theorems are stated.

## 5.1 A Joint State Realization for Digital Signatures

We now present a joint state realization $\mathcal{P}^{\text{js}}_{\text{sig}}$ of $\mathcal{F}_{\text{sig}}$. This realization uses a single copy of $\mathcal{F}_{\text{sig}}$ per party to realize multiple sessions of $\mathcal{F}_{\text{sig}}$ per party. The joint state theorem for digital signatures basically says that

$$!\mathcal{P}^{\text{js}}_{\text{sig}} \mid !\underline{\mathcal{F}'_{\text{sig}}} \;\leq\; !\underline{\underline{\mathcal{F}_{\text{sig}}}} \;,$$

where $\mathcal{F}'_{\text{sig}}$ is obtained from $\mathcal{F}_{\text{sig}}$ by renaming all input and output tapes. As described in Section 3, on the right-hand side we have the multi-session multi-party version of $\mathcal{F}_{\text{sig}}$: $!\underline{\mathcal{F}_{\text{sig}}}$ is the multi-party version of $\mathcal{F}_{\text{sig}}$, where in a run of this system we can have one copy of $\underline{\mathcal{F}_{\text{sig}}}$ per party, and $!\underline{\underline{\mathcal{F}_{\text{sig}}}}$ is the multi-session version of the multi-party version of $\mathcal{F}_{\text{sig}}$, where in a run of this system we can have multiple sessions of $\mathcal{F}_{\text{sig}}$ per party. So, altogether there can be one copy of $\mathcal{F}_{\text{sig}}$, denoted by $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$, per session (with SID) *sid* and per party (with PID) *pid* in a run of $!\underline{\underline{\mathcal{F}_{\text{sig}}}}$ with some environment. On the left-hand side, we consider only the multi-party version $!\underline{\mathcal{F}'_{\text{sig}}}$ of $\mathcal{F}'_{\text{sig}}$ and the "multiplexer" $!\mathcal{P}^{\text{js}}_{\text{sig}}$ and in a run of $!\mathcal{P}^{\text{js}}_{\text{sig}} \mid !\underline{\mathcal{F}'_{\text{sig}}}$ with some environment there can be at most one copy of $\mathcal{P}^{\text{js}}_{\text{sig}}$, denoted by $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$, for every party *pid* and this copy handles all sessions of this party through one copy of $\mathcal{F}'_{\text{sig}}$, namely the copy for party *pid*, which we denote by $\underline{\mathcal{F}'_{\text{sig}}}[pid]$. Hence, the multi-session multi-party version of $\mathcal{F}_{\text{sig}}$ is realized by a system (the joint state realization) with only a multi-party version of $\mathcal{F}'_{\text{sig}}$ where a copy of $\mathcal{F}'_{\text{sig}}$ for one party handles all sessions of that party. This is illustrated in Figure 6.

The basic idea of $\mathcal{P}^{\text{js}}_{\text{sig}}$ is simple and follows the one by Canetti and Rabin [14] (see also Figure 6): SIDs are added by $\mathcal{P}^{\text{js}}_{\text{sig}}$ to the messages to be signed so that signatures cannot be mixed between different sessions. More specifically, if a party *pid* in session *sid* sends a request to $!\mathcal{P}^{\text{js}}_{\text{sig}} \mid !\underline{\mathcal{F}'_{\text{sig}}}$ to sign a message $x$, i.e., a message of the form $(sid, (pid, (\mathsf{Sign}, x)))$ is send to $!\mathcal{P}^{\text{js}}_{\text{sig}} \mid !\underline{\mathcal{F}'_{\text{sig}}}$, and hence, $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$, then $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ replaces the

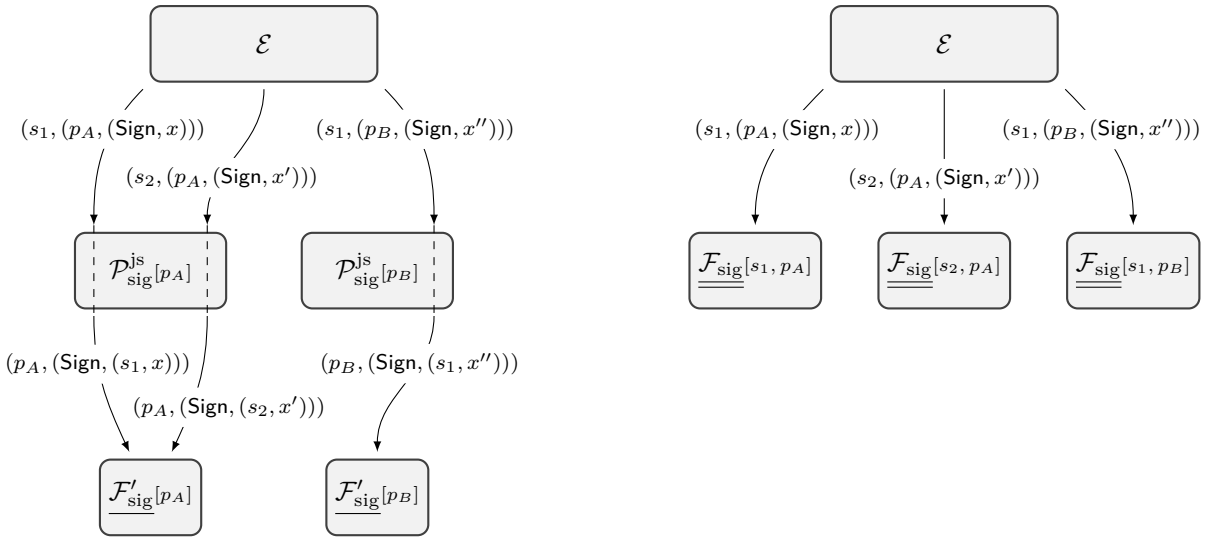


Figure 6: A run of $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}} \mid !\underline{\mathcal{F}'_{\mathrm{sig}}}$ (left) and $!\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}$ (right), respectively, where an environment $\mathcal{E}$ sends three signature generation requests: 1. to sign a message $x$ in session (with SID) $s_1$ with the private key of party (with PID) $p_A$, 2. to sign $x'$ in session $s_2$ with the private key of party $p_A$, and 3. to sign $x''$ in session $s_1$ with the private key of party $p_B$.

message $x$ by $(sid, x)$ and forwards the request to the copy of $\mathcal{F}'_{\mathrm{sig}}$ for this party, i.e., to $\underline{\mathcal{F}'_{\mathrm{sig}}}[pid]$.[19] Similarly, when a party $pid$ in session $sid$ sends a verification request for a signature $\sigma$, a message $x$, and a public key $pk$, then $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}[pid]$ forwards the requests to $\underline{\mathcal{F}'_{\mathrm{sig}}}[pid]$ but replaces $x$ by $(sid, x)$. However, this simple idea only works given an appropriate formulation of the digital signature functionality (see Section 6.1) and if some technical details are taken care of, see below.

We define $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}(n, D_{\mathrm{sid}})$ to be an IITM that is parametrized i) by a number $n > 0$ which, like in the case of $\mathcal{F}_{\mathrm{sig}}$, defines the I/O interface of $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ and ii) by a polynomial-time decidable domain of SIDs $D_{\mathrm{sid}} = \{D_{\mathrm{sid}}(\eta)\}_{\eta\in\mathbb{N}}$. Requests with an SID not in $D_{\mathrm{sid}}(\eta)$ (where $\eta$ is the security parameter) are ignored by $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$; see below why this is needed. We often omit $n$ and/or $D_{\mathrm{sid}}$ and just write, e.g., $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ instead of $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}(n, D_{\mathrm{sid}})$. The machine $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ additionally has an I/O interface to connect to $!\underline{\mathcal{F}'_{\mathrm{sig}}(n)}$ such that $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}(n) \mid !\underline{\mathcal{F}'_{\mathrm{sig}}(n)}$ and $!\underline{\underline{\mathcal{F}_{\mathrm{sig}}(n)}}$ have the same external I/O interface (which they must have because $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}(n) \mid !\underline{\mathcal{F}'_{\mathrm{sig}}(n)}$ is meant to realize $!\underline{\underline{\mathcal{F}_{\mathrm{sig}}(n)}}$. Following the above basic idea, $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ is defined in pseudocode in Figure 7. It is easy to see that $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}} \mid !\underline{\mathcal{F}'_{\mathrm{sig}}}$ is environmentally strictly bounded.

We emphasize a technical detail of $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ which is necessary for the joint state theorem to hold. If the environment sends the first request to $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}[pid]$, for some PID $pid$, with some SID $sid$, then $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}[pid]$ forwards it to $\underline{\mathcal{F}'_{\mathrm{sig}}}[pid]$ which in turn sends an initialization request to the adversary (on the network tape) and waits for a response from the adversary (because this is the first request sent to it). Now, while waiting for this response, the environment might send another request to $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}[pid]$, with some other SID $sid' \neq sid$. If this happens, $\underline{\mathcal{F}'_{\mathrm{sig}}}[pid]$ is still blocked because it is waiting for a response from the adversary. In the ideal world (i.e., in an interaction of the environment with $!\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}$ and a simulator) there would now be two copies, namely $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid, pid]$ and $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid', pid]$ waiting for a response to the initialization request from the simulator. The environment could provide a response to $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid', pid]$ which could then continue its work, while $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid, pid]$ is

[19] We note that the actual encoding of $(sid, x)$ as a bit string is not important. In fact, we could parametrize $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ by any appropriate pairing function $\tau\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ and replace $(sid, x)$ by $\tau(sid, x)$.

**Parameters:** – $n > 0$ {*number of I/O tape pairs*
– $D_{\text{sid}} = \{D_{\text{sid}}(\eta)\}_{\eta\in\mathbb{N}}$ {*polynomial-time decidable domain of SIDs*

**Tapes:** For all $i \in \{1,\ldots,n\}$: from/to $\mathsf{IO}_i$: $(\mathsf{io}_i^{\text{in}}, \mathsf{io}_i^{\text{out}})$; from/to $\mathsf{IO}'_i$ (to connect to $!\underline{\mathcal{F}'_{\text{sig}}}$): $(\mathsf{io}_i^{\text{out}\prime}, \mathsf{io}_i^{\text{in}\prime})$

**State:** – $\mathsf{pid} \in \{0,1\}^* \cup \{\bot\}$ {*PID; initially* $\bot$
– $\mathsf{B} \subseteq D_{\text{sid}}(\eta)$ {*set of blocked SIDs; initially* $\emptyset$
– $\mathsf{lastSID} \in D_{\text{sid}}(\eta) \cup \{\bot\}$ {*last received SID,* $\mathsf{lastSID} \neq \bot$ *iff waiting for response from* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$*; initially* $\bot$

**CheckAddress:** Accept input of the form $(sid, (pid, m))$ where $sid \in D_{\text{sid}}(\eta)$ from $\mathsf{IO}_i$ or input of the form $(pid, m)$ from $\mathsf{IO}'_i$ if $pid = \mathsf{pid}$ or $\mathsf{pid} = \bot$. Reject all other input.

**Initialization:** Upon receiving the first message, which, by definition of mode CheckAddress, is of the form $(sid, (pid, m))$ or $(pid, m)$, in mode Compute, set $\mathsf{pid} := pid$ (i.e., record the PID $pid$, which is used for addressing multiple instances of $\mathcal{P}^{\text{js}}_{\text{sig}}$). Then, continue processing the first message as defined below.

**Compute:**

**recv** $(sid, (\mathsf{pid}, m))$ **from** $\mathsf{IO}_i$**:**
**if** $\mathsf{lastSID} \neq \bot \vee sid \in \mathsf{B}$**:** {*waiting for response from* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$ *or SID is blocked*
**if** $\mathsf{lastSID} \neq sid$**:**
**add** $sid$ **to** $\mathsf{B}$
end this activation with empty output
**else:**
$\mathsf{lastSID} := sid$ {*record SID*
**if** $m = (\mathsf{Sign}, x)$ for some $x$**: send** $(\mathsf{pid}, (\mathsf{Sign}, (sid, x)))$ **to** $\mathsf{IO}'_i$ {*sign* $(sid, x)$ *using* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$
**else if** $m = (\mathsf{Verify}, pk, x, \sigma)$ for some $pk, x, \sigma$**: send** $(\mathsf{pid}, (\mathsf{Verify}, pk, (sid, x), \sigma))$ **to** $\mathsf{IO}'_i$ {*verify* $\sigma$ *using* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$
**else if** $m = \mathsf{PubKey}? \vee m = \mathsf{CorrStatus}?$**: send** $(\mathsf{pid}, m)$ **to** $\mathsf{IO}'_i$ {*forward request to* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$
**else:** $\mathsf{lastSID} := \bot$; end this activation with empty output {*ignore the request because it is not valid*

**recv** $(\mathsf{pid}, m)$ **from** $\mathsf{IO}'_i$ **s.t.** $\mathsf{lastSID} \neq \bot$**:** {*by definition,* $\mathsf{lastSID} \neq \bot$ *if* $\mathcal{P}^{\text{js}}_{\text{sig}}[\mathsf{pid}]$ *receives a message from* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$
$sid := \mathsf{lastSID}$; $\mathsf{lastSID} := \bot$; **send** $(sid, (\mathsf{pid}, m))$ **to** $\mathsf{IO}_i$ {*forward response from* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$ *with recorded SID*

Figure 7: The joint state realization $\mathcal{P}^{\text{js}}_{\text{sig}}$ for digital signatures. See Section 4.1 for notational conventions.

still blocked. Also, the environment could provide different responses to $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$ and $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid', pid]$. This is not possible in the real world where there is only one copy $\underline{\mathcal{F}'_{\text{sig}}}[pid]$ which is used to realize both $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$ and $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid', pid]$. To make the joint state realization indistinguishable from the ideal world in this case, we define $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ to record $sid'$ as *blocked* and to ignore this last request, i.e., to end this activation without producing output. All later requests with blocked SIDs are ignored too. Accordingly, the simulator will be defined to never complete initialization for $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid', pid]$. This guarantees that the environment cannot exploit such race conditions to distinguish between the joint state realization and the ideal world. It basically forces the environment to first finish the initialization before it can use $\underline{\mathcal{F}'_{\text{sig}}}[pid]$. Note that this problem of race conditions is limited to the initialization phase of $\underline{\mathcal{F}'_{\text{sig}}}[pid]$. During normal operation, i.e., after initialization, this instance answers all requests of $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ immediately without involving the environment. Thus, each request from the environment to $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ is also answered immediately, which prevents the environment from activating $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ in several different sessions simultaneously.[20] This would be a natural assumption as potential race conditions caused by non-immediate responses to initialization requests do not correspond to any actual attacks in reality.

Next, we state and prove the joint state theorem for digital signatures. In this theorem, we have to

[20]The problem could also be solved by restricting the environment. In particular, we could use the extended *IITM model with responsive environments* [5] which allows for requiring that the environment directly replies to certain requests without otherwise interfering with a protocol/functionality. In our specific case, this can be used to force the environment to directly reply to initialization requests from instances of $\underline{\mathcal{F}'_{\text{sig}}}$, making the blocking of SIDs unnecessary. The concept of responsive environment has been developed after the original submission of this work, and therefore is not considered here, although it would have been useful for simplifying the modeling.

restrict the length of SIDs to be polynomially bounded in the security parameter. This is needed to prove the theorem because the algorithms that are provided by the simulator and executed by $\mathcal{F}_{\text{sig}}$ get different inputs. In the joint state realization, they obtain input of the form $(sid, x)$ and in the ideal world, they just obtain input of the form $x$ and have to add the SID (see the proof for details). Therefore, we require that the domain of SIDs $D_{\text{sid}} = \{D_{\text{sid}}(\eta)\}_{\eta\in\mathbb{N}}$ is polynomially bounded: The domain $D_{\text{sid}} = \{D_{\text{sid}}(\eta)\}_{\eta\in\mathbb{N}}$ is called *polynomially bounded* if there exists a polynomial $q$ such that $|sid| \leq q(\eta)$ for all $\eta \in \mathbb{N}$ and $sid \in D_{\text{sid}}(\eta)$.

We note that the following theorem can be applied iteratively as described in Section 3 in order to reason about more and more complex systems. We also emphasize that the proof of this theorem uses Theorem 3 (composition theorem). By Theorem 3, it suffices to reason about only one party in order to obtain a result for multiple parties.

**Theorem 9.** *Let $n > 0$ and $D_{\text{sid}}$ be a polynomially bounded domain of SIDs. Then, for every polynomial $p$ there exists a polynomial $p'$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{sig}}(n, D_{\text{sid}}) \mid \underline{!\mathcal{F}'_{\text{sig}}(n, p)} \;\leq\; \underline{\underline{!\mathcal{F}_{\text{sig}}(n, p')}}$$

*where $\underline{!\mathcal{F}'_{\text{sig}}(n,p)}$ is the multi-party version of $\mathcal{F}_{\text{sig}}$ where all input and output tapes are renamed as described above and $\underline{\underline{!\mathcal{F}_{\text{sig}}(n,p')}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{sig}}$ where the domain of SIDs is $D_{\text{sid}}$.*[21]

*Proof.* Let $n > 0$ and $p$ be a polynomial as required by the theorem; we will show the existence of an appropriate polynomial $p'$ below. By Theorem 3, it suffices to reason about environments that use only a single PID: The protocol systems $\mathcal{P} := !\mathcal{P}^{\text{js}}_{\text{sig}}(n, D_{\text{sid}}) \mid \underline{!\mathcal{F}'_{\text{sig}}(n,p)}$ and $\mathcal{F} := \underline{\underline{!\mathcal{F}_{\text{sig}}(n,p')}}$ are $\sigma$-session versions (as defined in Section 2.4) for the following SID function $\sigma$: $\sigma(m, c) := pid$ if (i) $m = (sid, (pid, m'))$ for some $sid, pid, m'$ and $c$ is an external tape of $\mathcal{F}$ (or an external I/O tape of $\mathcal{P}$ because $\mathcal{F}$ and $\mathcal{P}$ have the same I/O interface) or (ii) $m = (pid, m')$ for some $pid, m'$ and $c$ is an external tape of $\underline{!\mathcal{F}'_{\text{sig}}}$ (i.e., an internal tape of $\mathcal{P}$, that connects $\mathcal{P}^{\text{js}}_{\text{sig}}$ with $\underline{!\mathcal{F}'_{\text{sig}}}$, or an external network tape of $\mathcal{P}$). Otherwise, $\sigma(m, c) := \bot$. So, to prove $\mathcal{P} \leq \mathcal{F}$, by Theorem 3, it suffices to show that $\mathcal{P}$ is environmentally bounded (which is easy to see, as mentioned above) and that $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$, i.e., that there exists a simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$ such that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ for every environment $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{P})$ that only uses a single PID $pid$ (of course, $\mathcal{E}$ may use multiple SIDs).

This "single-PID" simulator $\mathcal{S}$ that we define below will, when the environment sends algorithms $sig$, $ver$ and keys $pk$, $sk$ (to $\underline{\mathcal{F}'_{\text{sig}}[pid]}$), provide the algorithms $sig^{(sid)}$, $sig^{(sid)}$ and the keys $pk$, $sk$ to the instance $\underline{\underline{\mathcal{F}_{\text{sig}}[sid, pid]}}$ for every SID $sid$.

Let $\eta \in \mathbb{N}$ be a security parameter, $sid \in D_{\text{sid}}(\eta)$ be an SID, and $sig$ and $ver$ be descriptions of algorithms. We now define the algorithms $sig^{(sid)}$ and $ver^{(sid)}$:

- $sig^{(sid)}(sk, x)$ computes $\sigma \leftarrow sig(sk, (sid, x))$ and counts the steps needed. If at most $p(\eta+|sk|+|(sid, x)|)$ steps are needed, then it returns $\sigma$. Otherwise, it enters an infinite loop.
- $ver^{(sid)}(pk, x, \sigma)$ computes $b \leftarrow ver(pk, (sid, x), \sigma)$ and counts the steps needed. If at most $p(\eta + |pk| + |(sid, x)| + |\sigma|)$ steps are needed, then it returns $b$. Otherwise, it enters an infinite loop.

Since $|sid| \leq q(\eta)$ for some polynomial $q$ (because $D_{\text{sid}}$ is polynomially bounded), we find a polynomial $p'$ that only depends on $p$ and $q$ such that:

(a) For all $sk, x \in \{0,1\}^*$, the computation of $sig(sk, (sid, x))$ exceeds $p(\eta + |sk| + |(sid, x)|)$ steps if and only if the computation of $sig^{(sid)}(sk, x)$ exceeds $p'(\eta + |sk| + |x|)$ steps.

(b) For all $pk, x, \sigma \in \{0,1\}^*$, the computation of $ver(pk, (sid, x), \sigma)$ exceeds $p(\eta + |pk| + |(sid, x)| + |\sigma|)$ steps if and only if the computation of $ver^{(sid)}(pk, x, \sigma)$ exceeds $p'(\eta + |pk| + |x| + |\sigma|)$ steps.

[21] Recall the definition of session versions with domain from Section 2.4.

We now define the "single-PID" simulator $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$. Recall that this simulator has to work for environments that only use a single PID. The task of $\mathcal{S}$ is merely to forward initialization requests, to provide algorithms and keys, and to perform corruptions. For the simulation, $\mathcal{S}$ maintains a set $\mathsf{I}$ of SIDs (initially $\emptyset$), a flag $\mathsf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ (initially $\mathsf{false}$), and variables $\mathsf{sig}, \mathsf{ver}, \mathsf{pk}, \mathsf{sk}$ (initially undefined).

- When the simulator $\mathcal{S}$ receives the first initialization request from $\mathcal{F}$, i.e., the message $(sid, (pid, \mathsf{Init}))$ from $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid, pid]$ for some $sid, pid$, then $\mathcal{S}$ sends $(pid, \mathsf{Init})$ to $\mathcal{E}$.

- If another initialization requests arrives from $\mathcal{F}$, say with SID $sid'$ (i.e., from $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid', pid]$), but $\mathcal{S}$ has not yet received a response to the first initialization request from $\mathcal{E}$, then $\mathcal{S}$ records $sid'$ as *blocked* and ends this activation with empty output ($\mathcal{S}$ will never complete initialization for $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid', pid]$, i.e., this instance is "blocked", which corresponds to the fact that $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}[pid]$ would record $sid'$ as blocked in this case).

- When $\mathcal{S}$ receives a response to the initialization request from $\mathcal{E}$, i.e., a message of the form $(pid, (corrupt, sig, ver, pk, sk))$ with $corrupt \in \{\mathsf{false}, \mathsf{true}\}$, then $\mathcal{S}$ adds $sid$ to the set of initialized SIDs $\mathsf{I}$, sets $\mathsf{sig} := sig$; $\mathsf{ver} := ver$; $\mathsf{pk} := pk$; $\mathsf{sk} := sk$, and, if $corrupt = \mathsf{true}$, sets $\mathsf{corrupted} := \mathsf{true}$ (if $\mathsf{corrupted}$ is already $\mathsf{true}$, it remains $\mathsf{true}$ no matter what value $corrupt$ has). Then, $\mathcal{S}$ sends $(sid, (pid, (\mathsf{corrupted}, \mathsf{sig}^{(sid)}, \mathsf{ver}^{(sid)}, \mathsf{pk}, \mathsf{sk})))$ to $\mathcal{F}$ where $sid$ is the SID contained in the first initialization request $\mathcal{S}$ received from $\mathcal{F}$. That is, $\mathcal{S}$ completes initialization for $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid, pid]$.

- When another initialization request arrives from $\mathcal{F}$, say with SID $sid'$ and PID $pid$, and $\mathcal{S}$ has already received an initialization response from $\mathcal{E}$ (i.e., $\mathsf{I} \neq \emptyset$ and $\mathsf{sig}, \mathsf{ver}, \mathsf{pk}, \mathsf{sk}$ are defined), then $\mathcal{S}$ sends $(sid', (pid, (\mathsf{corrupted}, \mathsf{sig}^{(sid)}, \mathsf{ver}^{(sid)}, \mathsf{pk}, \mathsf{sk})))$ to $\mathcal{F}$ and adds $sid'$ to $\mathsf{I}$. That is, $\mathcal{S}$ completes initialization for $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}[sid', pid]$ without sending a request to $\mathcal{E}$.

- When $\mathcal{S}$ receives a corrupt request from $\mathcal{E}$, i.e., the message $(pid, \mathsf{Corrupt})$ for some $pid$, then $\mathcal{S}$ distinguishes the following cases:

  (i) If $\mathcal{S}$ has sent an initialization request to $\mathcal{E}$ but not yet received a response (i.e., $\mathsf{I} = \emptyset$), then $\mathcal{S}$ ignores the corrupt request (i.e., it ends this activation without producing output).

  (ii) If $\mathcal{S}$ has already received an initialization response from $\mathcal{E}$ (i.e., $\mathsf{I} \neq \emptyset$), then $\mathcal{S}$ sets $\mathsf{corrupted} := \mathsf{true}$ and, for every $sid \in \mathsf{I}$, sends $(sid, (pid, \mathsf{Corrupt}))$ to $\mathcal{F}$ and waits for receiving $(sid, (pid, \mathsf{Corrupted}))$ from $\mathcal{F}$ (which, by definition of $\mathcal{F}_{\mathrm{sig}}$, is the immediate response of $\mathcal{F}$ to corrupt requests). That is, $\mathcal{S}$ corrupts all existing instances of $\underline{\underline{\mathcal{F}_{\mathrm{sig}}}}$. Then, $\mathcal{S}$ returns $(pid, \mathsf{Corrupted})$ to $\mathcal{E}$.

  (iii) If $\mathcal{S}$ has not received any initialization request from $\mathcal{F}$ so far, then $\mathcal{S}$ sets $\mathsf{corrupted} := \mathsf{true}$ and returns $(pid, \mathsf{Corrupted})$ to $\mathcal{E}$.

- Upon any other input, $\mathcal{S}$ ends this activation without producing output.

It is easy to see that $\mathcal{S} \,|\, \mathcal{F}$ is environmentally strictly bounded, and hence, $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$.

Let $\mathcal{E} \in \mathrm{Env}_{\sigma\text{-single}}(\mathcal{P})$, i.e., $\mathcal{E}$ uses only a single PID. Furthermore, let $\eta \in \mathbb{N}$ be a security parameter and $a \in \{0,1\}^*$ be some external input. We now prove that $\Pr[(\mathcal{E} \,|\, \mathcal{P})(1^\eta, a) = 1] = \Pr[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F})(1^\eta, a) = 1]$ by showing that there exists a bijective mapping that maps every run $\rho$ of $(\mathcal{E} \,|\, \mathcal{P})(1^\eta, a)$ to a run $\rho'$ of $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F})(1^\eta, a)$ such that both runs have the same probability and overall output. From this, we immediately obtain $\Pr[(\mathcal{E} \,|\, \mathcal{P})(1^\eta, a) = 1] = \Pr[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F})(1^\eta, a) = 1]$. In fact, defining the bijection is simple. Let $\rho$ be a run of $(\mathcal{E} \,|\, \mathcal{P})(1^\eta, a)$. We define $\rho'$ as follows: First, we note that $\mathcal{P}^{\mathrm{js}}_{\mathrm{sig}}$ is deterministic and, since $\mathcal{E}$ only uses a single PID, there exists at most one instance of $\underline{\mathcal{F}'_{\mathrm{sig}}}$ in $\rho$. Let $\alpha_{\mathcal{E}}$ be the random coins used by $\mathcal{E}$ and $\alpha_{\mathcal{F}'_{\mathrm{sig}}}$ be the random coins used by the instance of $\underline{\mathcal{F}'_{\mathrm{sig}}}$ in $\rho$.[22] Furthermore, $\mathcal{S}$ is deterministic, that is, a run of

[22] We refer to [24] for a formal definition of random coins for systems of IITMs and runs of systems of IITMs.

$(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F})(1^\eta, a)$ is fixed by defining the random coins of $\mathcal{E}$ and $\mathcal{F}$ (note that $\mathcal{F}$ only uses random coins in the simulation of the signature algorithm). We define $\rho'$ by defining the random coins of $\mathcal{E}$ to be $\alpha_\mathcal{E}$ (i.e., $\mathcal{E}$ in $\rho'$ uses the same randomness as in $\rho$) and the random coins of $\mathcal{F}$ to be such that $\mathcal{F}$ uses the same random coins to sign messages as the instance of $\underline{\mathcal{F}'_{\text{sig}}}$ in $\rho$ uses to sign the messages. That is, the first message that is signed in $\rho$ is signed using the same random coins as those used to sign the first message in $\rho'$. This also holds for the second message and so on. By induction on the length of runs $\rho$, it is easy to see that the view of $\mathcal{E}$ is the same in both runs $\rho$ and $\rho'$ using the following arguments:

1. $\mathcal{E}$ does not observe any difference regarding blocked SIDs: It holds that $sid \in \mathsf{B}$ in $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ (in $\rho$) where $pid$ is the PID $\mathcal{E}$ uses iff $\mathcal{S}$ (in $\rho'$) recorded $sid$ as *blocked*. Therefore, if $\mathcal{E}$ sends a request with a blocked SID $sid$, then, in $\rho$, $\mathcal{P}^{\text{js}}_{\text{sig}}[pid]$ will end its activation with empty output and, in $\rho'$, $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$ will end its activation with empty output because it never completed initialization. Hence, in both runs the master IITM (in $\mathcal{E}$) is activated with empty input.

2. $\mathcal{E}$ does not observe any difference regarding Corrupted? requests: By definition of $\mathcal{S}$, $\underline{\mathcal{F}'_{\text{sig}}}[pid]$ (in $\rho$) is corrupted iff $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$ (in $\rho'$) is corrupted for all $sid$ such that $sid$ is not blocked and this instance exists.

3. The signing algorithm is executed on the same messages using the same random coins: Let $x$ be a message that is signed in $\rho$ with some SID $sid$. Let $sig$ be the signing algorithm and $sk$ be the secret key (both provided previously by $\mathcal{E}$). Then, the signature $\sigma$ is computed in $\rho$ by simulating $sig(sk, (sid, x))$ at most $p(\eta + |sk| + |(sid, x)|)$ steps and in $\rho'$ by simulating $sig^{(sid)}(sk, x)$ at most $p'(\eta + |sk| + |x|)$ steps. In both runs the same random coins are used. Hence, by definition of $sig^{(sid)}$ and (a), the same signature is created in both runs.

4. Similarly to signing, the verification algorithm is executed on the same messages in both runs. Hence, by definition of $ver^{(sid)}$ and (b), the algorithm returns the same verification result in both runs. Furthermore, the check to prevent forgery produces the same result in both runs because, for all $sid, x$, $(sid, x) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{sig}}}[pid]$ (in $\rho$) iff $x \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{sig}}}}[sid, pid]$ (in $\rho'$) where $pid$ is the PID $\mathcal{E}$ uses.

From this, we obtain that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$. Hence, $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$. By Theorem 3, we conclude $\mathcal{P} \leq \mathcal{F}$. □

Using the composition theorems, we can immediately replace the ideal functionality in the joint state realization by its realization as stated in Theorem 6, resulting in an actual joint state realization (without any ideal functionality):

**Corollary 3.** *Let $n > 0$, $D_{\text{sid}}$ be a polynomially bounded domain of SIDs, and $\Sigma$ be an UF-CMA secure signature scheme. Then, there exists a polynomial $p$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{sig}}(n, D_{\text{sid}}) \,|\, \underline{!\mathcal{P}'_{\text{sig}}(n, \Sigma)} \;\leq\; \underline{\underline{!\mathcal{F}_{\text{sig}}(n, p)}}$$

*where $\underline{!\mathcal{P}'_{\text{sig}}(n, \Sigma)}$ is the multi-party version of $\mathcal{P}_{\text{sig}}$ where all input and output tapes are renamed just as for $\mathcal{F}'_{\text{sig}}$ and, as above, $\underline{\underline{!\mathcal{F}_{\text{sig}}(n, p)}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{sig}}$ where the domain of SIDs is $D_{\text{sid}}$.*

*Proof.* By Theorem 6 (because $\Sigma$ is UF-CMA secure), it holds that $\mathcal{P}'_{\text{sig}} \leq \mathcal{F}'_{\text{sig}}(p')$ for any polynomial $p'$ that bounds the runtime of the algorithms in $\Sigma$. From this, by the composition theorems (Theorems 1 and 2), we obtain that $!\mathcal{P}^{\text{js}}_{\text{sig}} \,|\, \underline{!\mathcal{P}'_{\text{sig}}} \leq !\mathcal{P}^{\text{js}}_{\text{sig}} \,|\, \underline{!\mathcal{F}'_{\text{sig}}(p')}$. By Theorem 9 and transitivity of $\leq$, we conclude that $!\mathcal{P}^{\text{js}}_{\text{sig}} \,|\, \underline{!\mathcal{P}'_{\text{sig}}} \leq \underline{\underline{!\mathcal{F}_{\text{sig}}(p)}}$ for some polynomial $p$. □

**Parameters, Tapes, State, CheckAddress, Initialization:** Just as for $\mathcal{P}^{\text{js}}_{\text{sig}}$, see Figure 7.
**Compute:**

**recv** $(sid, (\mathsf{pid}, m))$ **from** $\mathsf{IO}_i$:
  **if** $\mathsf{lastSID} \neq \bot \vee sid \in \mathsf{B}$: {*waiting for response from* $\underline{\mathcal{F}'_{\text{pke}}}[\mathsf{pid}]$ *or SID is blocked*
    **add** *sid* **to** B; end this activation with empty output
  **else:**
    $\mathsf{lastSID} := sid$ {*record SID*
    **if** $m = (\mathsf{Enc}, pk, x)$ for some $pk, x$: **send** $(\mathsf{pid}, (\mathsf{Enc}, pk, (sid, x)))$ **to** $\mathsf{IO}'_i$ {*encrypt* $(sid, x)$ *using* $\underline{\mathcal{F}'_{\text{pke}}}[\mathsf{pid}]$
    **else if** $\exists y\colon m = (\mathsf{Dec}, y) \vee m = \mathsf{PubKey?} \vee m = \mathsf{CorrStatus?}$: **send** $(\mathsf{pid}, m)$ **to** $\mathsf{IO}'_i$ {*forward request to* $\underline{\mathcal{F}'_{\text{pke}}}[\mathsf{pid}]$
    **else:** $\mathsf{lastSID} := \bot$; end this activation with empty output {*ignore the request because it is not valid*

**recv** $(\mathsf{pid}, m)$ **from** $\mathsf{IO}'_i$ **s.t.** $\mathsf{lastSID} \neq \bot$: {*receive response from* $\underline{\mathcal{F}'_{\text{sig}}}[\mathsf{pid}]$*; by definition,* $\mathsf{lastSID} \neq \bot$
  $sid := \mathsf{lastSID}$; $\mathsf{lastSID} := \bot$
  **if** $m = (\mathsf{Plaintext}, (\mathsf{lastSID}, x))$ for some $x$: {$m$ *is response to a* Dec *request and plaintext is prefixed by correct SID*
    **send** $(sid, (\mathsf{pid}, (\mathsf{Plaintext}, x)))$ **to** $\mathsf{IO}_i$ {*strip off SID, return plaintext*
  **else if** $m = (\mathsf{Plaintext}, x)$ for some $x$: {$m$ *is response to a* Dec *request but plaintext has unexpected format*
    **send** $(sid, (\mathsf{pid}, (\mathsf{Plaintext}, \bot)))$ **to** $\mathsf{IO}_i$ {*return decryption error*
  **else:** {$m$ *is response to a* PubKey?*,* Enc*, or* CorrStatus? *request*
    **send** $(sid, (\mathsf{pid}, m))$ **to** $\mathsf{IO}_i$ {*forward response*

Figure 8: The joint state realization $\mathcal{P}^{\text{js}}_{\text{pke}}$ for public-key encryption. See Section 4.1 for notational conventions.

## 5.2 A Joint State Realization for Public-Key Encryption

The joint state realization $\mathcal{P}^{\text{js}}_{\text{pke}}$ presented in this section is similar to the joint state realization $\mathcal{P}^{\text{js}}_{\text{sig}}$ for digital signatures. It uses one copy of $\mathcal{F}_{\text{pke}}$ per party for all sessions of this party and realizes the multi-session, multi-party version of $\mathcal{F}_{\text{pke}}$ where one copy of $\mathcal{F}_{\text{pke}}$ is used per party *per session*. Hence, similarly to the case of digital signatures the joint state theorem for public-key encryption states that

$$!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}'_{\text{pke}}} \leq !\underline{\underline{\mathcal{F}_{\text{pke}}}}$$

where $\mathcal{F}'_{\text{pke}}$ is obtained from $\mathcal{F}_{\text{pke}}$ by renaming all input and output tapes. The basic idea for $\mathcal{P}^{\text{js}}_{\text{pke}}$, which again is similar to the case of digital signatures and first appeared in [11], but without any details or proofs (see Section 6.2 for further discussion), is as follows: The SID *sid* is prefixed to the plaintext $x$ prior to encryption, i.e., instead of encrypting $x$ in session *sid* under a separate key for this session, $(sid, x)$ is encrypted (under the same key for every session). Upon decryption of a ciphertext $y$ in session *sid* it is checked whether $y$ decrypts to $(sid, x)$ with the correct SID *sid*.[23] While the main idea is simple, it only works given an appropriate formulation of the public-key encryption functionality (see also Section 6.2).

Following this basic idea, $\mathcal{P}^{\text{js}}_{\text{pke}}$ is defined in pseudocode in Figure 8. Just as $\mathcal{P}^{\text{js}}_{\text{sig}}$, $\mathcal{P}^{\text{js}}_{\text{pke}}$ is parametrized by a number $n$ that defines the I/O interface and a domain of SIDs $D_{\text{sid}}$. We write $\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}})$ to denote $\mathcal{P}^{\text{js}}_{\text{pke}}$ with parameters $n$ and $D_{\text{sid}}$ but often omit one or both parameters. Analogously to the case of digital signatures, $\mathcal{P}^{\text{js}}_{\text{pke}}(n)$ connects to the I/O interface of $!\underline{\mathcal{F}'_{\text{pke}}}(n)$.

We can now state and prove the joint state theorem for public-key encryption. As mentioned above, the joint state realization $\mathcal{P}^{\text{js}}_{\text{pke}}$ is based on the multi-party version $!\underline{\mathcal{F}'_{\text{pke}}}$ of the ideal functionality $\mathcal{F}'_{\text{pke}}$. Recall that $\mathcal{F}'_{\text{pke}}$ is obtained from $\mathcal{F}_{\text{pke}}$ by renaming all external tapes. More importantly, $\mathcal{F}'_{\text{pke}}$ will use the leakage algorithm $L'$ that in addition to the leakage algorithm $L$ in the ideal world ($!\underline{\underline{\mathcal{F}_{\text{pke}}}}$) also leaks the SID of the session in which the message was encrypted. This, in conjunction with the decryption test performed in $\mathcal{F}_{\text{pke}}$ (to guarantee that the decryption of the encryption of a leakage yields the leakage again), guarantees that ciphertexts generated in different sessions are different. This is crucial for the joint state theorem to hold (see

[23] We note that the actual encoding of $(sid, x)$ as a bit string is not important. In fact, we could parametrize $\mathcal{P}^{\text{js}}_{\text{pke}}$ by any appropriate pairing function $\tau\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ and replace $(sid, x)$ by $\tau(sid, x)$.

below). Just as in the case of digital signatures, the domain of SIDs has to be restricted. We further remark that the theorem can be applied iteratively, as described in Section 3.

**Theorem 10.** *Let $n > 0$, $D_{\text{sid}}$ be a polynomially bounded domain of SIDs, and $L$ be a leakage algorithm. Then, for every polynomial $p$ there exists a polynomial $p'$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid \underline{!\mathcal{F}'_{\text{pke}}(n, p, L')} \;\leq\; \underline{\underline{!\mathcal{F}_{\text{pke}}(n, p', L)}}$$

*where*

1. *the leakage algorithm $L'$ is defined as follows: $L'(1^\eta, (sid, x)) := (sid, L(1^\eta, x))$ for all $\eta \in \mathbb{N}$, $sid \in D_{\text{sid}}(\eta)$, and $x \in D_L(\eta)$ (the domain of $L'$ is $D_{L'} = \{D_{L'}(\eta)\}_{\eta\in\mathbb{N}}$ with $D_{L'}(\eta) := \{(sid, x) \mid sid \in D_{\text{sid}}(\eta), x \in D_L(\eta)\}$ for all $\eta \in \mathbb{N}$),*
2. *$\underline{!\mathcal{F}'_{\text{pke}}(n, p, L')}$ is the multi-party version of $\mathcal{F}_{\text{pke}}$ where all input and output tapes are renamed, as described above, and*
3. *$\underline{\underline{!\mathcal{F}_{\text{pke}}(n, p', L)}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{pke}}$ where the domain of SIDs is $D_{\text{sid}}$.*[24]

*Proof.* The proof is similar to the proof of Theorem 9. The basic idea is that the usage of the SID in every plaintext, in conjunction with the definition of the leakage algorithm $L'$ (i.e., the SID is part of the leakage) and the decryption test performed in $\mathcal{F}_{\text{pke}}$ (i.e., ciphertexts are guaranteed to contain all the information contained in the leakage, in particularly the SID), guarantees that ciphertexts generated in different sessions are different and that ideal decryption (i.e., decryption that returns a recorded plaintext) in some session only succeeds if the ciphertext has been generated in this session.

As in the proof of Theorem 9, to show that $\mathcal{P} := !\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid \underline{!\mathcal{F}'_{\text{pke}}(n, p, L')}$ realizes $\mathcal{F} := \underline{\underline{!\mathcal{F}_{\text{pke}}(n, p', L)}}$, by Theorem 3, it suffices to show that there exists a simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$ such that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ for every environment $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{P})$ that only uses a single PID $pid$ (of course, $\mathcal{E}$ may use multiple SIDs), where $\sigma$ is the SID function defined in the proof of Theorem 9.

The "single-PID" simulator $\mathcal{S}$ is defined analogously to the one in the proof of Theorem 9, except for the following:

- $\mathcal{S}$ ignores corrupt requests from the environment (i.e., messages of the form $(pid, \mathsf{Corrupt})$) because $\mathcal{F}_{\text{pke}}$ is only corruptible upon initialization.
- We replace the algorithms $sig$ and $ver$, obtained from the environment, by the algorithms $enc$ and $dec$. Also, instead of $sig^{(sid)}$ and $sig^{(sid)}$, $\mathcal{S}$ provides the algorithms $enc^{(sid)}$ and $dec^{(sid)}$, defined below, to $\mathcal{F}$.

Let $\eta \in \mathbb{N}$ be a security parameter, $sid \in D_{\text{sid}}(\eta)$ be an SID, and $enc$ and $dec$ be descriptions of algorithms. We now define the algorithms $enc^{(sid)}$ and $dec^{(sid)}$:

- $enc^{(sid)}(pk, x)$ computes $y \leftarrow enc(pk, (sid, x))$ and counts the steps needed. If at most $p(\eta + |pk| + |(sid, x)|)$ steps are needed, then it returns $y$. Otherwise, it enters an infinite loop.
- $dec^{(sid)}(sk, y)$ computes $x' \leftarrow dec(sk, y)$ and counts the steps needed. If more than $p(\eta + |sk| + |y|)$ steps are needed, it enters an infinite loop. Otherwise, if $x' = (sid, x)$ for some $x$, then it returns $x$, otherwise, it returns the error symbol $\perp$.

Since $|sid| \leq q(\eta)$ for some polynomial $q$ (because $D_{\text{sid}}$ is polynomially bounded), we find a polynomial $p'$ that only depends on $p$ and $q$ (i.e., on $D_{\text{sid}}$) such that:

(a) For all $pk, x \in \{0,1\}^*$, the computation of $enc(pk, (sid, x))$ exceeds $p(\eta + |pk| + |(sid, x)|)$ steps if and only if the computation of $enc^{(sid)}(pk, x)$ exceeds $p'(\eta + |pk| + |x|)$ steps.

[24] Recall the definition of session versions with domain from Section 2.4.

(b) For all $sk, y \in \{0,1\}^*$, the computation of $dec(sk, y)$ exceeds $p(\eta + |sk| + |y|)$ steps if and only if the computation of $dec^{(sid)}(sk, y)$ exceeds $p'(\eta + |sk| + |y|)$ steps.

Let $\mathcal{E} \in \mathrm{Env}_{\sigma\text{-single}}(\mathcal{P})$, i.e., $\mathcal{E}$ uses only a single PID. Furthermore, let $\eta \in \mathbb{N}$ be a security parameter and $a \in \{0,1\}^*$ be some external input. As in the proof of Theorem 9, to prove that $\Pr[(\mathcal{E} \mid \mathcal{P})(1^\eta, a) = 1] = \Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F})(1^\eta, a) = 1]$, we show that there exists a bijective mapping that maps every run $\rho$ of $(\mathcal{E} \mid \mathcal{P})(1^\eta, a)$ to a run $\rho'$ of $(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F})(1^\eta, a)$ such that both runs have the same probability and overall output. Again, defining such a bijection is simple. Let $\rho$ be a run of $(\mathcal{E} \mid \mathcal{P})(1^\eta, a)$. Now, $\rho'$ is defined by defining the random coins of $\mathcal{E}$ and $\mathcal{F}$ (note that $\mathcal{S}$ is deterministic and $\mathcal{F}$ only uses random coins in the simulation of the leakage and encryption algorithms) such that $\mathcal{E}$ uses the same random coins as in $\rho$ and $\mathcal{F}$ uses the same random coins to encrypt messages as $\underline{\mathcal{F}'_{\mathrm{pke}}}$ uses in $\rho$. That is, the ciphertext for the first message that is encrypted in $\rho$ is computed using the same random coins as those used to compute the ciphertext for the first message in $\rho'$. This also holds for the second message and so on. By induction on the length of runs $\rho$, it is easy to see that the view of $\mathcal{E}$ is the same in both runs $\rho$ and $\rho'$ using the following arguments:

1. As in the proof of Theorem 9, $\mathcal{E}$ does not observe any difference regarding blocked SIDs or Corrupted? requests.

2. Upon encryption, in both runs, the leakage and encryption algorithms are executed on the same messages using the same random coins, and hence, the same ciphertext is returned: Let $x$ be a plaintext that is encrypted in $\rho$ under the public key $pk'$ with some SID $sid$ and PID $pid$. Furthermore, let $enc$ be the encryption algorithm and $pk$ be the public key provided previously by $\mathcal{E}$. We distinguish two cases:

   (i) *Ideal encryption*, i.e., $pk = pk'$ and $\underline{\mathcal{F}'_{\mathrm{pke}}[pid]}$ (in $\rho$) is not corrupted: In this case, the ciphertext $y$ returned to $\mathcal{E}$ is computed in $\rho$ by computing the leakage $\overline{x} \leftarrow L'(1^\eta, (sid, x))$ and simulating $enc(pk, \overline{x})$ at most $p(\eta + |pk| + |\overline{x}|)$ steps.
   We note that in this case, by definition of $\mathcal{S}$, $pk'$ is the recorded public key in $\underline{\underline{\mathcal{F}_{\mathrm{pke}}[sid, pid]}}$ (in $\rho'$) and $\underline{\underline{\mathcal{F}_{\mathrm{pke}}[sid, pid]}}$ is not corrupted. Hence, in $\rho'$, the ciphertext that is returned to $\mathcal{E}$ is computed by computing the leakage $\overline{x}' \leftarrow L(1^\eta, x)$ and simulating $enc^{(sid)}(pk, \overline{x}')$ at most $p'(\eta + |pk| + |\overline{x}'|)$ steps. The same random coins as in $\rho$ are used to compute the leakage. Hence, by definition of $L'$, $\overline{x} = (sid, \overline{x}')$. Furthermore, the same random coins as in $\rho$ are used to simulate the encryption algorithm. Hence, by definition of $enc^{(sid)}$ and by (a), the same ciphertext $y$ is returned.

   (ii) *Non-ideal encryption*, i.e., $pk \neq pk'$ or $\underline{\mathcal{F}'_{\mathrm{pke}}}$ in $\rho$ is corrupted: First, we note that in this case, by definition of $\mathcal{S}$, $pk'$ is not the recorded public key in $\underline{\underline{\mathcal{F}_{\mathrm{pke}}[sid, pid]}}$ (in $\rho'$) or $\underline{\underline{\mathcal{F}_{\mathrm{pke}}[sid, pid]}}$ is corrupted. Now, the ciphertext $y$ is computed in $\rho$ by simulating $enc(pk, (sid, x))$ at most $p(\eta + |pk| + |(sid, x)|)$ steps and in $\rho'$ by simulating $enc^{(sid)}(pk, x)$ at most $p'(\eta + |pk| + |x|)$ steps. Since in both runs the same random coins are used to simulate the algorithm, by definition of $enc^{(sid)}$ and by (a), the same ciphertext is created in both runs.

3. Upon decryption, in both runs, the same algorithms are executed on the same bit strings, and hence, the same plaintext is returned: Let $y$ be a ciphertext that is decrypted in $\rho$ with some SID $sid$ and PID $pid$. Furthermore, let $enc$, $dec$ be the algorithms and $pk, sk$ be the key pair provided previously by $\mathcal{E}$. We now distinguish the following cases:

   (i) *Ideal decryption*, i.e., $\underline{\mathcal{F}'_{\mathrm{pke}}[pid]}$ (in $\rho$) is not corrupted and there exists $x$ such that $(x, y) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\mathrm{pke}}[pid]}$: First, we note that in this case $\underline{\underline{\mathcal{F}_{\mathrm{pke}}[sid, pid]}}$ (in $\rho'$) is not corrupted. We now distinguish two cases:

   - *Ciphertexts collide*, i.e., there exist $x_0, x_1$ such that $x_0 \neq x_1$ and $(x_0, y), (x_1, y) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\mathrm{pke}}[pid]}$: In this case, decryption fails in $\rho$. We now show that decryption also fails in $\rho'$.

For this purpose, we first show that $x_0$ and $x_1$ "belong" to the same session. More precisely, we show that there exist $sid', x_0', x_1'$ such that $x_0 = (sid', x_0')$ and $x_1 = (sid', x_1')$. By definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$, $x_0 = (sid_0, x_0')$ and $x_1 = (sid_1, x_1')$ for some $sid_0, x_0', sid_1, x_1'$. Since $(x_0, y), (x_1, y) \in \mathsf{H}$ (in $\underline{\mathcal{F}'_{\text{pke}}}[pid]$ in $\rho$), $x_0$ and $x_1$ have been encrypted to $y$ such that $y$ has been computed once as $\overline{x}_0 \leftarrow L'(1^\eta, x_0); y \leftarrow enc(pk, \overline{x}_0)$ and another time as $\overline{x}_1 \leftarrow L'(1^\eta, x_1); y \leftarrow enc(pk, \overline{x}_1)$. Furthermore, the decryption check succeeded in both cases, i.e., $\overline{x}_0 = dec(sk, y) = \overline{x}_1$ (decryption is deterministic). By definition of $L'$ (because it leaks the SID), we have that $\overline{x}_0 = (sid_0, \overline{x}_0')$ and $\overline{x}_1 = (sid_1, \overline{x}_1')$ for some $\overline{x}_0', \overline{x}_1'$. Hence, $sid_0 = sid_1$.
Now, if $sid = sid'$ (i.e., $x_0$ and $x_1$ "belong" to session $sid$), then, because ideal encryption is performed identically in $\rho$ and $\rho'$ (as shown above), we have that $(x_0', y), (x_1', y) \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$. Since $x_0' \neq x_1'$ (because $x_0 \neq x_1$), we conclude that decryption also fails in $\rho'$.
Otherwise, i.e., $sid \neq sid'$, decryption also fails in $\rho'$ because $dec(sk, y) = (sid', \overline{x})$ for some $\overline{x}$. Hence, by definition of $dec^{(sid)}$, we obtain that $dec^{(sid)}(sk, y) = \bot$.

- *Ciphertexts do not collide*, i.e., there exist $x$ such that $(x, y) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{pke}}}[pid]$ and $x$ is unique with this property.
  If $x = (sid, x')$ for some $x'$, then the plaintext $x'$ is returned (to $\mathcal{E}$) in $\rho$. Since ideal encryption is performed identically in $\rho$ and $\rho'$ (as shown above), $(x', y) \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$, and there is no other recorded plaintext for $y$. Hence, $x'$ is returned in $\rho'$ too.
  Otherwise, i.e., $x = (sid', x')$ for some $sid', x'$ such that $sid \neq sid'$, by definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$, decryption fails in $\rho$ (i.e., $\bot$ is returned to $\mathcal{E}$). Since ideal encryption is performed identically in $\rho$ and $\rho'$ (as shown above), $(x'', y) \notin \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$ for any $x''$ ($(x', y)$ is only recorded in $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid', pid]$). Hence, $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$ computes the plaintext as $dec^{(sid)}(sk, y)$. This yields $\bot$ because $dec(sk, y) = (sid', \overline{x})$ for some $\overline{x}$ (since $\underline{\mathcal{F}'_{\text{pke}}}[pid]$ performed a decryption test upon encryption) and $sid \neq sid'$. Hence, decryption fails in $\rho'$ too.

(ii) *Non-ideal decryption*, i.e., $\underline{\mathcal{F}'_{\text{pke}}}[pid]$ (in $\rho$) is corrupted or there does not exist $x$ such that $(x, y) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{pke}}}[pid]$: In this case, the plaintext $x$ is computed by $\underline{\mathcal{F}'_{\text{pke}}}[pid]$ by simulating $dec(sk, y)$ at most $p(\eta + |sk| + |y|)$ steps ($x = \bot$ if number of steps is exceeded). Then, $x$ is returned to $\mathcal{P}^{\text{js}}_{\text{pke}}$ and $\mathcal{P}^{\text{js}}_{\text{pke}}$ returns $x'$ as the plaintext if $x = (sid, x')$ for some $x'$. Otherwise, $\mathcal{P}^{\text{js}}_{\text{pke}}$ returns $\bot$ (decryption error).
We note that in this case, by definition of $\mathcal{S}$, $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$ (in $\rho'$) is corrupted or, because ideal encryption is performed identically in $\rho$ and $\rho'$ (as shown above), there does not exist $x$ such that $(x, y) \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{pke}}}}[sid, pid]$ (in fact, $y$ is not recorded in any instance of $\underline{\underline{\mathcal{F}_{\text{pke}}}}$). Hence, in $\rho'$, the plaintext is computed by simulating $dec^{(sid)}(sk, y)$ at most $p'(\eta + |sk| + |y|)$ steps. By definition of $dec^{(sid)}$ and by (b), the same plaintext $x'$ (possibly $x' = \bot$) is returned.

From this, we obtain that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$. Hence, $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$. By Theorem 3, we conclude that $\mathcal{P} \leq \mathcal{F}$. □

We note that the above theorem implies that we obtain joint state realizations for all realizations of $\mathcal{F}_{\text{pke}}$. In particular, by Theorem 7 (and the composition theorems), we obtain that the joint state realization $!\mathcal{P}^{\text{js}}_{\text{pke}} \,|\, !\underline{\mathcal{P}_{\text{pke}}(\Sigma)}$ of an IND-CCA2 secure public-key encryption scheme $\Sigma$ realizes the multi-party, multi-session version of $\mathcal{F}_{\text{pke}}$. In this corollary, we assume that the leakage algorithm $L'$ (recall that $L'(1^\eta, (sid, x)) = (sid, L(1^\eta, x))$) is length preserving (i.e., $|L'(1^\eta, (sid, x))| = |(sid, L(1^\eta, x))| = |(sid, x)|$). Note that $L'$ is length preserving if $L$ is length preserving (e.g., $L$ is one of the leakage algorithms from Example 1) and pairing $(\cdot, \cdot)$ is *length preserving*. We say that pairing $(\cdot, \cdot)$ is *length preserving* if $|(sid, x)| = |(sid, x')|$ for all $\eta \in \mathbb{N}$, $sid \in D_{\text{sid}}(\eta)$, and $x, x' \in D_L(\eta)$ such that $|x| = |x'|$. This is a natural assumption.

**Corollary 4.** *Let $n > 0$, $D_{\text{sid}}$ be a polynomially bounded domain of SIDs, $\Sigma$ be an IND-CCA2 secure public-key encryption scheme, and $L$ be leakage algorithm such that the leakage algorithm $L'$ (as defined in*

*Theorem 10) is length preserving and $D_\Sigma = D_{L'}$ (i.e., $\Sigma$ and $L'$ have the same plaintext domain). Then, there exists a polynomial $p$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid !\underline{\mathcal{P}'_{\text{pke}}(n, \Sigma)} \;\leq\; !\underline{\underline{\mathcal{F}_{\text{pke}}(n, p, L)}}$$

*where $!\underline{\mathcal{P}'_{\text{pke}}(n, \Sigma)}$ is the multi-party version of $\mathcal{P}_{\text{pke}}$ where all input and output tapes are renamed as for $\mathcal{F}'_{\text{pke}}$ and, as above, $!\underline{\underline{\mathcal{F}_{\text{pke}}(n, p, L)}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{pke}}$ where the domain of SIDs is $D_{\text{sid}}$.*

*Proof.* By Theorem 7 (because $\Sigma$ is IND-CCA2 secure, $L'$ is length preserving, and $D_\Sigma = D_{L'}$), it holds that $\mathcal{P}'_{\text{pke}} \leq \mathcal{F}'_{\text{pke}}(p', L')$ for any polynomial $p'$ that bounds the runtime of the algorithms in $\Sigma$. From this, by the composition theorems (Theorems 1 and 2), we obtain that $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{P}'_{\text{pke}}} \leq !\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}'_{\text{pke}}(p', L')}$. By Theorem 10 and transitivity of $\leq$, we conclude that $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{P}'_{\text{pke}}} \leq !\underline{\underline{\mathcal{F}_{\text{pke}}(p, L)}}$ for some polynomial $p$. □

### 5.3 A Joint State Realization for Replayable Public-Key Encryption

The joint state realization for replayable public-key encryption is analogous to the joint state realization for public-key encryption. In fact, the same protocol $\mathcal{P}^{\text{js}}_{\text{pke}}$ (see Section 5.2) can be used. As before, the joint state theorem for replayable public-key encryption states that $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}'_{\text{rpke}}(L')}$ (where $\mathcal{F}'_{\text{rpke}}$ is obtained from $\mathcal{F}_{\text{rpke}}$ by renaming all external tapes) realizes the multi-session, multi-party version $!\underline{\underline{\mathcal{F}_{\text{rpke}}(L)}}$ where, again, $L'$ leaks the SID plus the information that $L$ leaks $(L'(1^\eta, (sid, x)) = (sid, L(1^\eta, x))$ for all $\eta, sid, x)$ and the domain of SIDs has to be restricted. The joint state theorem can be applied iteratively, as described in Section 3.

**Theorem 11.** *Let $n > 0$, $D_{\text{sid}}$ be a polynomially bounded domain of SIDs, and $L$ be a leakage algorithm. Then, for every polynomial $p$ there exists a polynomial $p'$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid !\underline{\mathcal{F}'_{\text{rpke}}(n, p, L')} \;\leq\; !\underline{\underline{\mathcal{F}_{\text{rpke}}(n, p', L)}}$$

*where the leakage algorithm $L'$ is defined as in Theorem 10, $!\underline{\mathcal{F}'_{\text{rpke}}(n, p, L')}$ is the multi-party version of $\mathcal{F}_{\text{rpke}}$ where all input and output tapes are renamed, as described above, and $!\underline{\underline{\mathcal{F}_{\text{rpke}}(n, p', L)}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{rpke}}$ where the domain of SIDs is $D_{\text{sid}}$.*[25]

*Proof.* The proof is similar to the proof of Theorem 10. To show that $\mathcal{P} := !\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid !\underline{\mathcal{F}'_{\text{rpke}}(n, p, L')}$ realizes $\mathcal{F} := !\underline{\underline{\mathcal{F}_{\text{rpke}}(n, p', L)}}$, by Theorem 3, if suffices to show that there exists a simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}}_{\sigma\text{-single}}(\mathcal{F})$ such that $\mathcal{E} \mid \mathcal{P} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ for every environment $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{P})$ that only uses a single PID *pid* (of course, $\mathcal{E}$ may use multiple SIDs), where $\sigma$ is the SID function defined in the proof of Theorem 9. This "single-PID" simulator $\mathcal{S}$ is defined as in the proof of Theorem 10 and we use the same polynomial $p'$. In particular, $\mathcal{S}$ uses the same algorithms $enc^{(sid)}$ and $dec^{(sid)}$.

Let $\mathcal{E} \in \text{Env}_{\sigma\text{-single}}(\mathcal{P})$, i.e., $\mathcal{E}$ uses only a single PID. Furthermore, let $\eta \in \mathbb{N}$ be a security parameter and $a \in \{0,1\}^*$ be some external input. To prove that $\Pr[(\mathcal{E} \mid \mathcal{P})(1^\eta, a) = 1] = \Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F})(1^\eta, a) = 1]$, we show that there exists a bijective mapping that maps every run $\rho$ of $(\mathcal{E} \mid \mathcal{P})(1^\eta, a)$ to a run $\rho'$ of $(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F})(1^\eta, a)$ such that both runs have the same probability and overall output. The definition of such a bijection coincides with the one in the proof of Theorem 10, except that $\mathcal{F}_{\text{pke}}$ is replaced by $\mathcal{F}_{\text{rpke}}$. The proof that this bijection has the desired properties is similar to the one in the proof of Theorem 10: By induction on the length of runs $\rho$, it can be shown that $\mathcal{E}$ has the same view in both runs $\rho$ and $\rho'$. The proof only differs from the one of Theorem 10 in the argument for the case of decryption:

Let $y$ be a ciphertext that is decrypted in $\rho$ with some SID *sid* and PID *pid* and let *enc*, *dec* be the algorithms and $pk, sk$ be the key pair provided previously by $\mathcal{E}$. Furthermore, let $\overline{x}$ be the plaintext/leakage

[25] Recall the definition of session versions with domain from Section 2.4.

computed by $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$ (in $\rho$), i.e., by simulating $dec(sk, y)$ at most $p(\eta + |sk| + |y|)$ steps (if more steps would be needed, $\overline{x} = \bot$). Analogously, let $\overline{x}^*$ be the plaintext/leakage computed by $\underline{\underline{\mathcal{F}_{\text{rpke}}}}[sid, pid]$ (in $\rho'$), i.e., by simulating $dec^{(sid)}(sk, y)$ at most $p'(\eta + |sk| + |y|)$ steps (if more steps would be needed, $\overline{x}^* = \bot$). By definition of $dec^{(sid)}$ and $p'$ (see (b) in the proof of Theorem 10), we have that

$$\overline{x} = (sid, \overline{x}^*) \text{ if and only if } \overline{x}^* \neq \bot \ . \tag{2}$$

Hence, if $\overline{x} = \bot$, then $\overline{x}^* = \bot$ and, therefore, decryption fails both in $\rho$ and $\rho'$ (i.e., $\bot$ is returned to $\mathcal{E}$). Now, assume that $\overline{x} \neq \bot$. We distinguish the following cases:

(i) *Ideal decryption*, i.e., $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$ (in $\rho$) is not corrupted and there exists $x$ such that $(x, \overline{x}) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$: First, we note that in this case $\underline{\underline{\mathcal{F}_{\text{rpke}}}}[sid, pid]$ (in $\rho'$) is not corrupted. Furthermore, by definition of $L'$ (since $L'$ leaks the SID), $\overline{x} = (sid', \overline{x}')$ for some $sid', \overline{x}'$. We now distinguish two cases:

- *Leakages collide*, i.e., there exist $x_0, x_1$ such that $x_0 \neq x_1$ and $(x_0, \overline{x}), (x_1, \overline{x}) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$: In this case, decryption fails in $\rho$. We now show that decryption also fails in $\rho'$.
  First, it is easy to see that $x_0$ and $x_1$ "belong" to the same session $sid'$, i.e., $x_0 = (sid', x'_0)$ and $x_1 = (sid', x'_1)$ for some $x'_0, x'_1$.[26]
  Now, if $sid = sid'$ (i.e., $x_0$ and $x_1$ "belong" to session $sid$), then, by (2), $\overline{x}' = \overline{x}^*$. Since ideal encryption is performed identically in $\rho$ and $\rho'$, we have that $(x'_0, \overline{x}^*), (x'_1, \overline{x}^*) \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{rpke}}}}[sid, pid]$. Since $x'_0 \neq x'_1$ (because $x_0 \neq x_1$), we conclude that decryption also fails in $\rho'$. Otherwise, i.e., $sid \neq sid'$, decryption also fails in $\rho'$ because, by (2), $\overline{x}^* = \bot$.
- *Leakages do not collide*, i.e., there exist $x$ such that $(x, \overline{x}) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$ and $x$ is unique with this property.
  If $x = (sid, x')$ for some $x'$, then the plaintext $x'$ is returned (to $\mathcal{E}$) in $\rho$. Furthermore, $sid' = sid$ and, by (2), $\overline{x} = (sid, \overline{x}^*)$. Since ideal encryption is performed identically in $\rho$ and $\rho'$, we have that $(x', \overline{x}^*) \in \mathsf{H}$ in $\underline{\underline{\mathcal{F}_{\text{rpke}}}}[sid, pid]$ and there is no other recorded plaintext for $\overline{x}^*$. Hence, $x'$ is returned in $\rho'$ too.
  Otherwise, i.e., $x = (sid'', x')$ for some $sid'', x'$ such that $sid \neq sid''$, by definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$, decryption fails in $\rho$ (i.e., $\bot$ is returned to $\mathcal{E}$). Furthermore, $sid' = sid''$, i.e., $\overline{x} = (sid'', \overline{x}')$. Hence, by (2), $\overline{x}^* = \bot$, i.e., decryption fails in $\rho'$ too.

(ii) *Non-ideal decryption*, i.e., $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$ (in $\rho$) is corrupted or there does not exist $x$ such that $(x, \overline{x}) \in \mathsf{H}$ in $\underline{\mathcal{F}'_{\text{rpke}}}[pid]$:

If $\overline{x} = (sid, \overline{x}')$ for some $\overline{x}'$, then, by definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$, the plaintext $\overline{x}'$ is returned to $\mathcal{E}$ in $\rho$. In this case, by (2), $\overline{x}' = \overline{x}^*$ and this plaintext is returned to $\mathcal{E}$ in $\rho'$ too.

Otherwise, i.e., $\overline{x} \neq (sid, \overline{x}')$ for any $\overline{x}'$, then decryption fails in $\rho$ (by definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$). By (2), $\overline{x}^* = \bot$ in this case and, hence, decryption fails in $\rho'$ too.

From this, we obtain that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$. Hence, $\mathcal{P} \leq_{\sigma\text{-single}} \mathcal{F}$. By Theorem 3, we conclude that $\mathcal{P} \leq \mathcal{F}$. □

Similarly to the case of public-key encryption, we note that the above theorem implies that we obtain joint state realizations for all realizations of $\mathcal{F}_{\text{rpke}}$. In particular, by Theorem 8 (and the composition theorems), we obtain that the joint state realization $!\mathcal{P}^{\text{js}}_{\text{pke}} \,|\, !\underline{\mathcal{P}_{\text{pke}}(\Sigma)}$ with an IND-RCCA secure public-key encryption scheme $\Sigma$ realizes the multi-party, multi-session version of $\mathcal{F}_{\text{rpke}}$.

[26] We note that in the proof of Theorem 10, to show this, the decryption test upon encryption was required. This is not needed here. Hence, the decryption test could be omitted in $\mathcal{F}_{\text{rpke}}$, as mentioned above.

**Corollary 5.** *Let $n > 0$, $D_{\text{sid}}$ be a polynomially bounded domain of SIDs, $\Sigma$ be an IND-RCCA secure public-key encryption scheme, and $L$ be a leakage algorithm such that the leakage algorithm $L'$ (as defined in Theorem 10) is length preserving, has high entropy, and $D_\Sigma = D_{L'}$ (i.e., $\Sigma$ and $L'$ have the same plaintext domain). Then, there exists a polynomial $p$ such that:*

$$!\mathcal{P}^{\text{js}}_{\text{pke}}(n, D_{\text{sid}}) \mid \underline{!\mathcal{P}'_{\text{pke}}(n, \Sigma)} \;\leq\; \underline{\underline{!\mathcal{F}_{\text{rpke}}(n, p, L)}}$$

*where $\underline{!\mathcal{P}'_{\text{pke}}(n, \Sigma)}$ is the multi-party version of $\mathcal{P}_{\text{pke}}$ where all input and output tapes are renamed as for $\mathcal{F}'_{\text{rpke}}$ and, as above, $\underline{\underline{!\mathcal{F}_{\text{rpke}}(n, p, L)}}$ is the multi-session, multi-party version of $\mathcal{F}_{\text{rpke}}$ where the domain of SIDs is $D_{\text{sid}}$.*

*Proof.* By Theorem 8 (because $\Sigma$ is IND-RCCA secure, $L'$ is length preserving and has high entropy, and $D_\Sigma = D_{L'}$), it holds that $\mathcal{P}'_{\text{pke}} \leq \mathcal{F}'_{\text{rpke}}(p', L')$ for any polynomial $p'$ that bounds the runtime of the algorithms in $\Sigma$. From this, by the composition theorems (Theorems 1 and 2), we obtain that $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid \underline{!\mathcal{P}'_{\text{pke}}} \leq !\mathcal{P}^{\text{js}}_{\text{pke}} \mid \underline{!\mathcal{F}'_{\text{rpke}}(p', L')}$. By Theorem 11 and transitivity of $\leq$, we conclude that $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid \underline{!\mathcal{P}'_{\text{pke}}} \leq \underline{\underline{!\mathcal{F}_{\text{rpke}}(p, L)}}$ for some polynomial $p$. □

We note that the above corollary holds in particular if $L$ is the leakage algorithm that returns a random bit string of the same length as the plaintext and if the plaintext domain contains only "long" plaintexts (e.g., only plaintexts of length $\geq \eta$ for security parameter $\eta$) and if pairing $(\cdot, \cdot)$ is length preserving, as defined in Section 5.2 (i.e., the length of a pair of an SID and a plaintext does not depend on the actual bits of the plaintext but only on the SID and the length of the plaintext). In this case, it is easy to see that $L'$ is length preserving and has high entropy.

# 6 Related Work

We have already discussed some related work in the introduction. In this section, we compare our ideal functionalities and results with the ones from the literature in more detail.

## 6.1 Digital Signatures

We first compare our formulation of the digital signature functionality with other formulations in the literature.

As mentioned in the introduction, most other formulations of digital signature functionalities are defined in a non-local way [1,7,13,14], i.e., all signatures are provided by the adversary, with the mentioned disadvantages. The only formulations with local computation in the literature, besides the one in the present paper, are the ones in [6] (see the version of December 2005) and [2].

The digital signature functionality in [2] is part of a Dolev-Yao style cryptographic library. A user does not obtain the actual signature but only a handle to this signature within the library. By this, the use of the signature is restricted for the user to the operations provided in the cryptographic library. The implementation for the digital signature functionality within the library does not use a standard UF-CMA secure digital signature scheme, but requires a specific stronger construction. Joint state realizations have not been considered. In fact, the library is expressed within the model by Pfitzmann and Waidner [26] which does not explicitly talk about copies of protocols/functionalities.

One problem of the formulation in [6] is that it does not seem to have any reasonable joint state realization, unlike claimed in [6]: The signature functionality in [6] uses only the signing and verification algorithms *sig* and *ver*, but no public/private keys *pk*/*sk*. It is argued that the public/private keys can be incorporated in the algorithms *ver*/*sig*. That is, the verification algorithm plays the role of the public key. Thus, in order to verify a message-signature pair $(x, \sigma)$ in addition to this pair a verification algorithm *ver*′ has to be provided and in the functionality it is then checked if *ver*′ = *ver*. If *ver*′ ≠ *ver*, the algorithm *ver*′ is run on $(x, \sigma)$, and the result of this algorithm is returned. While this integration of the keys into the algorithms works for the private key, it does not work for the public key. As argued next, failing to make the distinction between the

verification algorithm and the public key prevents to obtain joint state realizations following the "concatenate and sign" approach or any approach that somehow manipulates the signed messages in an observable way.

An environment $\mathcal{E}$ that distinguishes a joint state realization from the multi-session, multi-party version of the digital signature functionality in the ideal world works as follows: It sends an initialization message to some copy of the digital signature functionality and provides some algorithms *sig* and *ver*. It then requests to verify the message-signature pair $(x, \sigma)$, where $x$ is not of the form $(sid, x')$ for any $sid, x'$, with the verification algorithm $ver'$ where $ver' \neq ver$ is defined as follows: $ver'(x, \sigma)$ outputs $\mathsf{true}$ if the message $x$ is of shape $(sid, x')$, it outputs $\mathsf{false}$ otherwise. If $\mathcal{E}$ obtains $(\mathsf{VerResult}, \mathsf{true})$, it outputs 1, and 0 otherwise. It is easy to see that if $\mathcal{E}$ communicates with the joint state realization it will always output 1 since this realization forwards $(sid, x)$ to the digital signature functionality. Since $ver' \neq ver$, the functionality will call $ver'((sid, x), \sigma)$ and so 1 is returned. Conversely, in the ideal world where $\mathcal{E}$ communicates directly with a copy of the digital signature functionality, $\mathcal{E}$ will always output 0 since this copy runs $ver'(x, \sigma)$.

Another problem in Canetti's formulation of the digital signature functionality in [6] is that the signing algorithm *sig* is allowed to preserve some state (i.e., the signature values may depend on the messages signed so far). Note that in our formulation *sig* is stateless. It is easy to prove that with a stateful *sig*, joint state realizations, such as "concatenate and sign" or similar approaches, fail, depending on the kind of state that is used. The problem is that the signing algorithms in the real and the ideal world will have different states, and that this cannot be prevented by the simulator. If states of signing algorithms are predictable and observable to some extent, then an environment can easily distinguish between the real and the ideal world. Note that Canetti's joint state realization is based on his ideal digital signature functionality and this functionality accepts any signature and verification algorithms from the environment/simulator. Hence, one in particular has to deal with the described "problematic" algorithms, which, however, is not possible. An alternative would be to restrict the kind of stateful signing algorithms that may be provided by the environment/simulator. This class would have to be carefully defined in order to fulfill certain closure properties to be useful in the context of joint state realizations. In any case, it would have to exclude several existing stateful signature schemes as they are problematic in the sense described. Also, the analysis of complex protocols based on functionalities which are parametrized by certain classes of signing/verification algorithms would be more complex.

While we define corruption very thoroughly, other formulations of signature functionalities lack to do so. But when it comes to joint state realizations, this is crucial. For example, if corruption reveals the order of the messages that have been signed so far, then the environment is able to distinguish the joint state world from the ideal world because the simulator has no chance to determine the order in which messages of different sessions where signed in the ideal world. If the messages are revealed in random order or in some order that is independent from the moment of activation (e.g., in lexicographical order), the joint state theorem for digital signatures still holds because the simulator is able to obtain the messages from each copy of the digital signature functionality and can combine them such that they respect the expected ordering.

## 6.2 Public-Key Encryption

In the proof of the joint state theorem for public key encryption, several subtleties come up which were overlooked in other works, in particular [6, 11]. In these works, joint state theorems, similar to Theorem 10, for public-key encryption functionalities with local computation were mentioned. However, the joint state realizations were only sketched and no proofs were provided. It, in fact, turns out that the joint state theorems for these functionalities do not hold true. Let us first explain this for [6] (see the version of December 2005) and then for [11]. These explanations motivate and justify the definition of our functionality and the way our joint state theorem is stated.

**Problems with the joint state realization in [6].** There are two problems:

1. The public-key encryption functionality in [6], unlike our functionality, identifies the public/private keys with the encryption/decryption algorithms *enc*/*dec*. While this works for the private key, it is problematic for the public key as explained next. If the environment wishes to encrypt a message $x$,

it is supposed to also present an encryption *algorithm* $enc'$ (not just a key, as in our functionality). If $enc \neq enc'$, i.e., $enc'$ is different from the algorithm associated with the functionality, then the ciphertext returned is $enc'(x)$. Now, assume that the environment asks to encrypt some message $x$ with $enc'$ in session $sid$, where, say $enc'$ coincides with $enc$ except that $enc'$ uses a different public key. In the joint state world (i.e., in an interaction with $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}_{\text{pke}}}$), the ciphertext is computed as $enc'((sid, x))$. In the ideal world (i.e., in an interaction with the simulator and $!\underline{\underline{\mathcal{F}_{\text{pke}}}}$), the ciphertext is computed as $enc'(x)$. Since the two ciphertexts have different lengths, the environment can easily distinguish between the joint state and ideal world no matter what simulator is chosen.

2. In [6], the leakage is fixed to be the length of a message, i.e., instead of a message $x$ a fixed message $\mu_{|x|}$ of length $|x|$ is encrypted (e.g., $\mu_{|x|} = 1^{|x|}$). In particular, this is so also in the joint state world. Hence, the SID is not leaked. This is problematic: The kind of encryption and decryption algorithms that may be provided by the simulator/environment in the joint state and ideal world to the public-key encryption functionality are not restricted in any way. In particular, the encryption algorithm that is provided may be deterministic. But then, if the environment asks to encrypt two different messages of the same length in two different sessions for the same party, then the resulting ciphertexts will be the same, since in both cases some fixed message is encrypted. In the ideal world, the two ciphertexts can be decrypted, since they are stored in different sessions. In the joint state world, decryption fails: The decryption box has two entries with the same ciphertext but different plaintexts. (The leakage that we use prevents this.) Consequently, the environment can easily distinguish between the ideal and joint state world.

To circumvent the second problem, one might think that restricting the environment to only provide encryption and decryption algorithms that originate from probabilistic encryption schemes where the probability for clashes between ciphertexts are negligible solves the problem. Let us call such an encryption scheme a *valid* encryption scheme. However, this is not the case if, as in [6], SIDs are not leaked in the joint state world; even if the algorithms provided by the environment/simulator are assumed to be IND-CCA2 secure.

Upon encryption of some message $x_0$ with the proper public key $pk$ in some session $sid_0$ in the joint state world the ciphertext $y$ is computed as $enc(pk, \mu_{|(sid_0,x_0)|})$. Depending on $\mu_n$ and how pairings are encoded, we have that

$$\mu_{|(sid_0,x_0)|} = (sid_1, x_1) \tag{3}$$

for some SID $sid_1$ and some plaintext $x_1$. This is, for example, the case if SIDs are assumed to have fixed length (e.g., the length of the security parameter) and are simply appended at the beginning of a message. This is a natural encoding, but our argument also works for other encodings and choices of $\mu_n$ (see below). Note that the environment can even try to choose $x_0$ and $sid_0$ in order to make (3) true.

When trying to prove the joint state theorem, the obvious candidate for a simulator, subsequently called the *standard simulator*, is the following. If the standard simulator receives algorithms $enc(\cdot,\cdot)$, $dec(\cdot,\cdot)$ and the private/public key pair $sk/pk$ from the environment, then it provides the algorithms $enc^{(sid)}(\cdot,\cdot)$ and $dec^{(sid)}(\cdot,\cdot)$ and the key pair $sk/pk$ to the instance of $\mathcal{F}_{\text{pke}}$ with SID $sid$ where

$enc^{(sid)}(pk', x)$: if $pk = pk'$ and not corrupted then return $enc(pk, \mu_{|(sid,x)|})$ else return $enc(pk', (sid, x))$ [27]

and

$dec^{(sid)}(sk, y)$: $x := dec(sk, y)$; if $x = (sid', x')$ for some $x'$ and $sid' = sid$ then return $x'$ else return $\perp$

for all SIDs $sid$. This seems to be the only reasonable simulator because in the joint state world a ciphertext for a message $x$ in session $sid$ is computed as $enc(pk, \mu_{|(sid,x)|})$ and the plaintext of a ciphertext $y$ that

[27] Technically, $enc_{sid}$ cannot know whether the functionality is corrupted or not but if we assume only static corruption then the simulator is able to know whether the functionality is corrupted or not at the moment it is requested to present the algorithms and can hard-code this into $enc_{sid}$.

was not output by the functionality is computed as $x = dec(sk, y)$ and the joint state realization checks if $x = (sid', x')$ and outputs $x'$ if $sid' = sid$ and $\perp$ otherwise.

Now, we provide an environment $\mathcal{E}$ that distinguishes between the joint state and the ideal world for such a simulator. As we will see, $\mathcal{E}$ will not corrupt any parties. Therefore, the simulator will not do so either: In the UC model the simulator is prohibited to do so by the control function and in the IITM model $\mathcal{E}$ could check this by requesting the functionality if it is corrupted and then distinguish between joint state and ideal world.

First, $\mathcal{E}$ initializes two instances for the same party, say with PID $pid$; one in session (with SID) $sid_0$ and one in session $sid_1$. Furthermore, $\mathcal{E}$ provides algorithms $enc(\cdot,\cdot)$, $dec(\cdot,\cdot)$ and the public/private keys $pk/sk$ where $enc$, $dec$, $pk$, and $sk$ originate from a valid encryption scheme (e.g., they could belong to an IND-CCA2 secure encryption scheme). Then, $\mathcal{E}$ requests to encrypt the plaintext $x_0$ under the (correct) public key $pk$ of party $pid$ in session $sid_0$. Let $y$ denote the resulting ciphertext. Finally, $\mathcal{E}$ sends a decryption request for $y$ and party $pid$ in session $sid_1$. It outputs "joint state" (or 1) if the returned plaintext is $\perp$, and "ideal" (or 0) otherwise.

It is easy to see that $\mathcal{E}$ determines correctly whether it interacts with the joint state or the ideal world: In the joint state world, the plaintext returned by the ideal functionality upon the decryption request by $\mathcal{E}$ is $(sid_0, x_0)$ (this is the plaintext recorded in the ideal functionality along with $y$). Since $sid_0 \neq sid_1$, the joint state realization returns $\perp$ as plaintext. In the ideal world, since $y$ has not been recorded in session $sid_1$, $y$ is encrypted using $dec^{(sid_1)}(sk, y)$. That is, first $dec(sk, y) = \mu_{|(sid_0,x_0)|}$ is computed. Then, it is checked whether the first component of $\mu_{|(sid_0,x_0)|}$ is $sid_1$, which, because of $\mu_{|(sid_0,x_0)|} \overset{(3)}{=} (sid_1, x_1)$, is the case. Therefore, $x_1$ is returned as plaintext by $dec^{(sid_1)}(sk, y)$.

We note that even if in $\mathcal{F}_{\text{pke}}$ instead of a constant message randomly chosen messages are encrypted in the case of ideal encryption, the above argument still works in case SIDs are short, but the success probability of the environment will be smaller (still non-negligible).

**Problems with the joint state realization in [11].** In [11], a (certified) public-key encryption functionality with local computation is proposed which is parametrized by fixed encryption and decryption algorithms; the keys are embedded in the algorithms, and hence, are also fixed (below we discuss the case that keys are not fixed). For this functionality, a theorem similar to Theorem 10 is stated only informally and without a proof. One can only hope such a theorem to hold if one assumes that in the ideal world the ideal functionality is defined in such a way that its SID is given to the encryption and decryption algorithms by the functionality, and that the encryption and decryption algorithms make use of the SID in the same way as prescribed by the simulator in the proof of Theorem 10. So, already the ideal functionality has to mimic the joint state realization. However, the ideal functionality in the joint state world should be defined differently: It should ignore SIDs, because in the joint state world SIDs are handled outside of the ideal functionality. Hence, the joint state theorem would be defined with different ideal functionalities in the joint state and ideal world. This has not been mentioned in [11]. But even if this is done, the theorem would still not hold if in the joint state world SIDs are not leaked. The reasoning is similar to the one above for the joint state theorem in [6]. Note that since the keys as well as encryption and decryption algorithms are fixed, the environment can still decrypt messages on its own. To fix this problem, the ideal functionality in the joint state world would have to be modified to account for the leakage. Altogether, these modifications would mimic what is happening in Theorem 10 and our proof of this theorem.

Alternatively, instead of parametrizing the functionality with a fixed public-key, encryption and decryption algorithm, one could have the functionality generate its own keys. In this case in the ideal world for encryption different public keys would be used in different sessions for the same party while in the joint state world the same key would be used for all sessions of this party. For the joint state theorem to hold this would require the encryption scheme to hide the public key, which is not a property IND-CCA2 secure schemes have in general.

**Remarks on other functionalities in the literature.** As mentioned in the introduction, other formulations of public-key encryption functionalities, e.g., those in [7, 17], are defined in a non-local way, i.e.,

all ciphertexts are provided by the adversary, with the mentioned disadvantages. Formulations with local computation, besides the one discussed above, have been proposed in [2, 26].

The public-key encryption functionality in [2] is part of a Dolev-Yao style cryptographic library. It has similar restrictions as the digital signatures in this library: A user does not obtain the actual ciphertexts but only a handle to the ciphertexts within the library. By this, the use of ciphertexts by the user is restricted to the operations provided in the library. The implementation of the public-key encryption functionality within the library does not use a standard IND-CCA2 secure scheme, but requires a specific stronger construction.

In [26], formulations of public-key encryption functionalities with local computation are proposed which are parametrized by specific encryption and decryption algorithms, with the same drawbacks (concerning joint state) mentioned for [11].

We note that in [2, 26] joint state realizations of the proposed functionalities have not been considered.

**General remarks.** One general remark for joint state theorems is that specifying corruption precisely is vital, as we do in our work, since some forms of corruption do not allow for joint state realizations. For example, if upon corruption all messages encrypted so far would be given to the adversary in order of occurrence, the joint state and ideal world could be distinguished because the order in the joint state world cannot be reconstructed by the simulator in the ideal world. (See also the discussion of corruption for joint state for digital signatures in Section 6.1.)

### 6.3 Replayable Public-Key Encryption

Canetti et al. [12] define and motivate IND-RCCA secure encryption schemes and propose a public-key functionality with *non-local* computation that captures IND-RCCA security. In [6] (see the version of December 2005), Canetti sketches in a few lines how his public-key encryption functionality with local computation should be modified to obtain a functionality that mimics IND-RCCA security. However, the modification that Canetti proposes only makes sense in a setting with non-local computation of ciphertexts. A proof of equivalence of his functionality with IND-RCCA security is not provided. Also, neither in [12] nor in [6] the issue of joint state is mentioned in the context of IND-RCCA security. So, our formulation of replayable public-key encryption with local computation is the first such formulation. Also, we are the first to propose a joint state realization (see Section 5.3) in the context of IND-RCCA security.

The general remarks in Sections 6.1 and 6.2 about the features and advantages of our formulations of digital signature and public-key encryption functionalities compared to other formulations also apply to our formulation of the functionality for replayable public-key encryption.

### 6.4 Joint State Theorems Without Pre-Established Session Identifiers

The ideal functionalities and the realizations proposed here (and in other works) require parties to have pre-established and globally unique SIDs before using the functionalities/realizations. This implicitly requires protocols to use these SIDs in some essential way in order to prevent "interference" between different protocol sessions. In joint state realizations such SIDs are prefixed to the messages to be encrypted/signed.

While this is a good design principle, not all protocols use pre-established SIDs. This is, for example, the case for most real-world authentication, key exchange, and secure channel protocols.

Therefore, in [23], an alternative way of addressing multiple protocol session without pre-established and globally unique SIDs is presented within the IITM model. Also, composition and joint state theorems without such SIDs are presented. In the formulation in [23], parties merely use locally chosen and managed SIDs.

## References

[1] 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.

[2] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, pages 220–230. ACM, 2003.

[3] 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.

[4] M. Bellare, D. Hofheinz, and E. Kiltz. Subtleties in the Definition of IND-CCA: When and How Should Challenge-Decryption be Disallowed? Technical Report 2009/418, Cryptology ePrint Archive, 2009. Available at http://eprint.iacr.org/2009/418.

[5] J. Camenisch, R. R. Enderlein, S. Krenn, R. Küsters, and D. Rausch. Universal Composition with Responsive Environments. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security*, volume 10032 of *Lecture Notes in Computer Science*, pages 807–840. Springer, 2016. A full version is available at https://eprint.iacr.org/2016/034.

[6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. Technical Report 2000/067, Cryptology ePrint Archive, 2000. Available at http://eprint.iacr.org/2000/067 with new versions from December 2005, July 2013, and December 2018.

[7] 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.

[8] 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.

[9] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Time-Bounded Task-PIOAs: A Framework for Analyzing Security Protocols. In S. Dolev, editor, *20th International Symposium on Distributed Computing (DISC 2006)*, pages 238–253. Springer, 2006.

[10] 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.

[11] 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.

[12] R. Canetti, H. Krawczyk, and J. Nielsen. Relaxing Chosen-Ciphertext Security. In *Advances in Cryptology, 23rd Annual International Cryptology Conference (CRYPTO 2003)*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

[13] R. Canetti and T. Rabin. Universal Composition with Joint State. Technical Report 2002/047, Cryptology ePrint Archive, 2002. Version of Nov. 2003. Available at http://eprint.iacr.org/2002/047.

[14] 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.

[15] A. Datta, R. Küsters, J. Mitchell, and A. Ramanathan. On the Relationships Between Notions of Simulation-Based Security. In J. Kilian, editor, *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, volume 3378 of *Lecture Notes in Computer Science*, pages 476–494. Springer, 2005.

[16] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[17] D. Hofheinz, J. Mueller-Quade, and R. Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. Cryptology ePrint Archive, Report 2003/024, 2003. Available at http://eprint.iacr.org/2003/024.

[18] D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial Runtime in Simulatability Definitions. In *18th IEEE Computer Security Foundations Workshop (CSFW-18 2005)*, pages 156–169. IEEE Computer Society, 2005.

[19] D. Hofheinz, J. Müller-Quade, and D. Unruh. A simple model of polynomial time uc. One-page abstract of a talk given at the Workshop on Models for Cryptographic Protocols (MCP 2006), 2006.

[20] D. Hofheinz and V. Shoup. GNUC: A New Universal Composability Framework. Technical Report 2011/303, Cryptology ePrint Archive, 2011. Available at http://eprint.iacr.org/2011/303.

[21] 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. See [24] for a full and revised version.

[22] R. Küsters and M. Tuengerthal. Joint State Theorems for Public-Key Encryption and Digital Signature Functionalities with Local Computation. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF 2008)*, pages 270–284. IEEE Computer Society, 2008. The full version is available at https://eprint.iacr.org/2008/006.

[23] R. Küsters and M. Tuengerthal. Composition Theorems Without Pre-Established Session Identifiers. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS 2011)*, pages 41–50. ACM, 2011.

[24] R. Küsters, M. Tuengerthal, and D. Rausch. The IITM Model: a Simple and Expressive Model for Universal Composability. *Journal of Cryptology*. To appear. Available at http://eprint.iacr.org/2013/025.

[25] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In M. Yung, editor, *Advances in Cryptology, 22nd Annual International Cryptology Conference (CRYPTO 2002)*, volume 2442 of *Lecture Notes in Computer Science*, pages 191–214. Springer, 2002.

[26] 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, 2001.

# A Security Definitions for Cryptographic Primitives

In this section, we recall standard security notions for cryptographic schemes. They will be used to realize the ideal functionalities for cryptographic primitives that we present in this paper.

Traditionally, security notions for cryptographic primitives are defined with respect to adversaries that do not obtain external input, except for the security parameter. Universal composability frameworks such as the UC model [6, 7] or the IITM model deal with environments that receive external input (and where the runtime of the environment might depend on the security parameter and the length of the external

input). In order to realize ideal functionalities for cryptographic primitives, reduction proofs to the security of the underlying primitives are necessary, and hence, the notions have to be compatible. We chose to adapt the standard notions of security, i.e., we formulate them with respect to adversaries that receive external input. We note, however, that all our results carry over to the setting without external input, i.e., where all environments and adversaries do not receive external input (except for the security parameter).

To define the security notions, as usual in cryptographic literature, we use the following notation: By

$$\Pr\left[y \leftarrow A(x) : B\right]$$

we denote the probability of an event $B$ where the probability distribution is given by a probabilistic algorithm $A$ with input $x$: $y$ is a random variable that is distributed according to the probability distribution induced by $A$. This notion is extended naturally to allow for a sequence of algorithms $A_1, \ldots, A_n$ instead of $A$. For example, given algorithms $\mathsf{gen}$, $\mathsf{enc}$, and $A$; a security parameter $\eta \in \mathbb{N}$; and a bit string $x$, the probability that $A$ on input $y$ outputs 1 where (the probability distribution of) $y$ is obtained from running $\mathsf{gen}$ on input $1^\eta$ (to obtain $k$) and then running $\mathsf{enc}$ on input $k$ and $x$ is denoted by

$$\Pr\left[k \leftarrow \mathsf{gen}(1^\eta); y \leftarrow \mathsf{enc}(k, x) : A(y) = 1\right] \quad .$$

Furthermore, we use the notion of negligible functions as used in the IITM model (which we recalled at the end of the introduction).

## A.1 Digital Signatures

In this section, following [16], we recall standard notions for digital signature schemes.

**Definition 9.** A *(digital) signature scheme* $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ consists of three polynomial-time algorithms. The probabilistic key generation algorithm $\mathsf{gen}$ expects a security parameter (in unary form) and returns a pair of keys $(pk, sk)$, the public key $pk$ and the private key $sk$. The (possibly) probabilistic signing algorithm $\mathsf{sig}$ expects a private key and a message and returns a signature. The deterministic verification algorithm $\mathsf{ver}$ expects a public key, a message, and a signature and returns $\mathsf{true}$ (verification succeeds) or $\mathsf{false}$ (verification fails).

It is required that, for every security parameter $\eta \in \mathbb{N}$, public/private key pair $(pk, sk)$ generated by $\mathsf{gen}(1^\eta)$, message $m \in \{0,1\}^*$, and signature $\sigma$ generated by $\mathsf{sig}(sk, m)$, it holds that $\mathsf{ver}(pk, m, \sigma) = \mathsf{true}$.

We note that we do not restrict the domain of messages, i.e., any bit string is a valid message. However, all results presented in this paper could easily be extended to deal with other domains.

**Definition 10** (UF-CMA security)**.** A digital signature scheme $\Sigma$ is called *existentially unforgeable under adaptive chosen-message attacks (UF-CMA secure)* if for every probabilistic, polynomial-time algorithm $A^{O(\cdot)}$ with access to a signing oracle $O$, the *UF-CMA advantage* of $A$ against $\Sigma$

$$\begin{aligned}\mathrm{Adv}^{\text{uf-cma}}_{A,\Sigma}(1^\eta, a) := \Pr[(pk, sk) \leftarrow \mathsf{gen}(1^\eta); (m, \sigma) \leftarrow A^{\mathsf{sig}(sk,\cdot)}(1^\eta, a, pk) :\\ \mathsf{ver}(pk, m, \sigma) = \mathsf{true} \text{ and } A \text{ has not queried } \mathsf{sig}(sk, \cdot) \text{ with } m]\end{aligned}$$

is negligible (as a function in $\eta$ and $a$).

## A.2 Public-Key Encryption

In this section, we recall two security notions for public-key encryption schemes: *IND-CCA2* (following [3]) and *replayable IND-CCA2 (IND-RCCA)* (following [12]).

**Definition 11.** A *public-key encryption scheme* $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ consists of three polynomial-time algorithms. The probabilistic key generation algorithm $\mathsf{gen}$ expects a security parameter $1^\eta$ and returns a pair of keys $(pk, sk)$, the public key $pk$ and the private key $sk$. The probabilistic encryption algorithm $\mathsf{enc}$

expects a public key and a plaintext and returns a ciphertext. The deterministic decryption algorithm $\mathsf{dec}$ expects a private key and a ciphertext and returns a plaintext if decryption succeeds. Otherwise, it returns the special symbol $\perp$.

We assume that every public-key encryption scheme $\Sigma$ is associated with a polynomial-time decidable domain of plaintexts $D_\Sigma = \{D_\Sigma(\eta)\}_{\eta\in\mathbb{N}}$ for some $D_\Sigma(\eta) \subseteq \{0,1\}^*$ for all $\eta \in \mathbb{N}$. We require that, for every security parameter $\eta \in \mathbb{N}$, public/private key pair $(pk, sk)$ generated by $\mathsf{gen}(1^\eta)$, plaintext $x \in D_\Sigma(\eta)$, and ciphertext $y$ generated by $\mathsf{enc}(pk, x)$, it holds that $\mathsf{dec}(sk, y) = x$.

**IND-CCA2 security.** Following [3], IND-CCA2 security is defined as follows. We note that this notion is called IND-CCA-SE in the taxonomy of [4].

**Definition 12** (IND-CCA2 security)**.** A public-key encryption scheme $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ is called *IND-CCA2 secure* (indistinguishability of encryptions under adaptive chosen-ciphertext attacks) if, for every adversary $A = (A_1, A_2)$ that is a pair of probabilistic, polynomial-time algorithms such that:

1. $A_1^{O(\cdot)}$ expects a security parameter $1^\eta$, external input $a$, and a public key $pk$ as input, has access to an oracle $O$, and produces output of the form $(x_0, x_1, s)$ consisting of two plaintexts $x_0, x_1 \in D_\Sigma(\eta)$ of the same length ($|x_0| = |x_1|$) and a bit string $s$ (some information $A_1$ wants to pass to $A_2$), and
2. $A_2^{O(\cdot)}$ expects a bit string $s$ and a ciphertext $y \in \{0,1\}^*$ as input, has access to an oracle $O$ but never queries $O$ with input $y$, and outputs a bit $b' \in \{0,1\}$,

the *IND-CCA2 advantage* of $A$ against $\Sigma$

$$\mathrm{Adv}^{\text{ind-cca2}}_{A,\Sigma}(1^\eta, a) := \Big|2 \cdot \Pr\Big[(pk, sk) \leftarrow \mathsf{gen}(1^\eta); (x_0, x_1, s) \leftarrow A_1^{\mathsf{dec}(sk,\cdot)}(1^\eta, a, pk); b \xleftarrow{\$} \{0,1\}; \\ y \leftarrow \mathsf{enc}(pk, x_b); b' \leftarrow A_2^{\mathsf{dec}(sk,\cdot)}(s, y) : b = b'\Big] - 1\Big|$$

is negligible (as a function in $\eta$ and $a$).

**IND-RCCA security.** The notion IND-RCCA security (replayable IND-CCA2 security) for public-key encryption schemes is a relaxed form of IND-CCA2 security where modifications of the ciphertext that yield the same plaintext are permitted. In particular, IND-CCA2 security implies IND-RCCA security [12]. IND-RCCA security has been introduced by Canetti, Krawczyk, and Nielsen in [12]. As explained in [12], IND-RCCA security suffices in many applications where IND-CCA2 security is used. IND-RCCA security is defined as follows.

**Definition 13** (IND-RCCA security)**.** A public-key encryption scheme $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ is called *IND-RCCA secure* (replayable IND-CCA2 secure) if, for every adversary $A = (A_1, A_2)$ that is a pair of probabilistic, polynomial-time algorithms such that:

1. $A_1^{O(\cdot)}$ expects a security parameter $1^\eta$, external input $a$, and a public key $pk$ as input, has access to an oracle $O$, and produces output of the form $(x_0, x_1, s)$ consisting of two plaintexts $x_0, x_1 \in D_\Sigma(\eta)$ of the same length ($|x_0| = |x_1|$) and a bit string $s$ (some information $A_1$ wants to pass to $A_2$), and
2. $A_2^{O(\cdot)}$ expects a bit string $s$ and a ciphertext $y \in \{0,1\}^*$ as input, has access to an oracle $O$, and outputs a bit $b' \in \{0,1\}$,

the *IND-RCCA advantage* of $A$ against $\Sigma$

$$\mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}(1^\eta, a) := \Big|2 \cdot \Pr\Big[(pk, sk) \leftarrow \mathsf{gen}(1^\eta); (x_0, x_1, s) \leftarrow A_1^{\mathsf{dec}(sk,\cdot)}(1^\eta, a, pk); b \xleftarrow{\$} \{0,1\}; \\ y \leftarrow \mathsf{enc}(pk, x_b); b' \leftarrow A_2^{\overline{\mathsf{dec}}(x_0,x_1,sk,\cdot)}(s, y) : b = b'\Big] - 1\Big|$$

**Parameters:** – $n > 0$ {*number of I/O tape pairs*
– $\Sigma = (\mathsf{gen}, \mathsf{sig}, \mathsf{ver})$ {*signature scheme*
**Tapes:** from/to $\mathsf{IO}_i$ $(i \in \{1, \ldots, n\})$: $(\mathsf{io}_i^{\mathrm{in}}, \mathsf{io}_i^{\mathrm{out}})$; from/to NET: $(\mathsf{net}^{\mathrm{in}}_{\mathcal{P}_{\mathrm{sig}}}, \mathsf{net}^{\mathrm{out}}_{\mathcal{P}_{\mathrm{sig}}})$
**State:** – $\mathsf{pk}, \mathsf{sk} \in \{0,1\}^* \cup \{\bot\}$ {*key pair; initially* $\bot$
– $\mathsf{corrupted} \in \{\mathsf{false}, \mathsf{true}\}$ {*corruption status; initially* false
**CheckAddress:** Accept every input on every tape.
**Initialization:** Upon receiving the first message in mode Compute do:
**send** Init **to** NET; **recv** $(corrupt, pk, sk)$ **from** NET **s.t.** $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ {*ask adversary for corruption*
**if** $corrupt = \mathsf{true}$**:** $\mathsf{corrupted} := \mathsf{true}$; $\mathsf{pk} := pk$; $\mathsf{sk} := sk$ {*use key pair provided by adversary*
**else:** $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\eta)$ {*generate fresh key pair*
Then, continue processing the first request as defined below.
**Compute:**
**recv** PubKey? **from** $\mathsf{IO}_i$**: send** $(\mathsf{PubKey}, \mathsf{pk})$ **to** $\mathsf{IO}_i$ {*return public key*
**recv** $(\mathsf{Sign}, x)$ **from** $\mathsf{IO}_i$**:** $\sigma \leftarrow \mathsf{sig}(\mathsf{sk}, x)$; **send** $(\mathsf{Signature}, \sigma)$ **to** $\mathsf{IO}_i$ {*sign* $x$*, return signature*
**recv** $(\mathsf{Verify}, pk, x, \sigma)$ **from** $\mathsf{IO}_i$**:** $b \leftarrow \mathsf{ver}(pk, x, \sigma)$; **send** $(\mathsf{VerResult}, b)$ **to** $\mathsf{IO}_i$ {*verify, return result*
**recv** CorrStatus? **from** $\mathsf{IO}_i$**: send** $(\mathsf{CorrStatus}, \mathsf{corrupted})$ **to** $\mathsf{IO}_i$ {*corruption status request*
**recv** Corrupt **from** NET**:** $\mathsf{corrupted} := \mathsf{true}$; **send** $(\mathsf{Corrupted}, \mathsf{pk}, \mathsf{sk})$ **to** NET {*adaptive corruption*

Figure 9: The realization $\mathcal{P}_{\mathrm{sig}}$ of $\mathcal{F}_{\mathrm{sig}}$. See Section 4.1 for notational conventions.

is negligible (as a function in $\eta$ and $a$), where the oracle $\overline{\mathrm{dec}}$ is defined as follows:

$$\overline{\mathrm{dec}}(x_0, x_1, sk, y)\text{: } \textbf{if } \mathsf{dec}(sk, y) \in \{x_0, x_1\}\textbf{: return } \mathsf{test} \textbf{ else: return } \mathsf{dec}(sk, y) \ .$$

(The symbol test is a special symbol which is not confused with any bit string or $\bot$.)

# B Proofs of Theorems 6, 7, and 8

## B.1 Proof of Theorem 6

We now prove Theorem 6, i.e., that $\mathcal{P}_{\mathrm{sig}}$ realizes $\mathcal{F}_{\mathrm{sig}}$ if and only if $\Sigma$ is UF-CMA secure.

### B.1.1 $\Sigma$ is UF-CMA Secure $\Rightarrow \mathcal{P}_{\mathrm{sig}} \leq \mathcal{F}_{\mathrm{sig}}$

First, we assume that $\Sigma$ is UF-CMA secure and show that $\mathcal{P}_{\mathrm{sig}} \leq \mathcal{F}_{\mathrm{sig}}$.

We use the following straightforward simulator $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}_{\mathrm{sig}}}(\mathcal{F}_{\mathrm{sig}})$: $\mathcal{S}$ is a single IITM that accepts all messages in mode CheckAddress. In mode Compute, upon receiving Init from $\mathcal{F}_{\mathrm{sig}}$, $\mathcal{S}$ forwards Init to the environment and waits for receiving a message of the form $(corrupt, pk, sk)$ from the environment where $corrupt \in \{\mathsf{false}, \mathsf{true}\}$ and $pk, sk \in \{0,1\}^*$. Any message not of this form is ignored by $\mathcal{S}$, i.e., $\mathcal{S}$ ends the activation with empty output. When receiving $(corrupt, pk, sk)$ with $corrupt = \mathsf{false}$, i.e., the environment does not want to corrupt $\mathcal{P}_{\mathrm{sig}}$ upon initialization, then $\mathcal{S}$ generates a new public/private key pair $(pk', sk') \leftarrow \mathsf{gen}(1^\eta)$ using the key generation algorithm of $\Sigma$, sends $(\mathsf{false}, \mathsf{sig}, \mathsf{ver}, pk', sk')$ to $\mathcal{F}_{\mathrm{sig}}$ (where sig and ver are descriptions of the signing and verification algorithms of $\Sigma$). When receiving $(corrupt, pk, sk)$ with $corrupt = \mathsf{true}$, i.e., the environment wants to corrupt $\mathcal{P}_{\mathrm{sig}}$ upon initialization, then $\mathcal{S}$ sends $(\mathsf{true}, \mathsf{sig}, \mathsf{ver}, pk, sk)$ to $\mathcal{F}_{\mathrm{sig}}$ (i.e., $\mathcal{S}$ corrupts $\mathcal{F}_{\mathrm{sig}}$ upon initialization). Then, $\mathcal{S}$ waits for receiving Corrupt from the environment (any other input is ignored by $\mathcal{S}$, as above). If it receives Corrupt, $\mathcal{S}$ forwards Corrupt to $\mathcal{F}_{\mathrm{sig}}$, waits for receiving Corrupted from $\mathcal{F}_{\mathrm{sig}}$, and sends $(\mathsf{Corrupted}, pk, sk)$ to the network where $pk, sk$ is the key pair provided previously to $\mathcal{F}_{\mathrm{sig}}$ (i.e., either the key pair provided by the environment or the key pair generated by $\mathcal{S}$). Then, $\mathcal{S}$ again waits for receiving Corrupt from the environment and this request is processed in the same way as described above.

It is easy to see that $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}}$ is environmentally strictly bounded, i.e., $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}_{\mathrm{sig}}}(\mathcal{F}_{\mathrm{sig}})$.

Let $\mathcal{E} \in \mathrm{Env}(\mathcal{P}_{\mathrm{sig}})$ be an environment of $\mathcal{P}_{\mathrm{sig}}$. Using $\mathcal{E}$, we construct an UF-CMA adversary $A$ on $\Sigma$ such that, basically, $A$ is successful if $\mathcal{E}$ successfully distinguishes between $\mathcal{P}_{\mathrm{sig}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}}$. We define $A^{O(\cdot)}(1^\eta, a, pk)$ (where $O$ is a signing oracle) as follows: $A$ simulates a run of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ with the following exceptions:

(a) If $\mathcal{E}$ corrupts $\mathcal{P}_{\mathrm{sig}}$ (i.e., either upon initialization or later by sending the request $\mathsf{Corrupt}$ to $\mathcal{P}_{\mathrm{sig}}$ on the network tape), then $A$ aborts (i.e., in this case, $A$ fails to produce a forgery).

(b) Instead of using the public key generated by $\mathcal{P}_{\mathrm{sig}}$, the public key $pk$ is used. (The private key generated by $\mathcal{P}_{\mathrm{sig}}$ is never used because $O$ is used for signing, see below.)

(c) Whenever $\mathcal{P}_{\mathrm{sig}}$ wants to compute the signature $\sigma$ of a message $x$, then $A$ instead computes $\sigma \leftarrow O(x)$ using its signing oracle $O$.

(d) Whenever $\mathcal{P}_{\mathrm{sig}}$ verifies a signature $\sigma$ for a message $x$, then $A$ checks whether $(x, \sigma)$ constitutes a forgery, i.e., $x$ has never been signed before and $\mathsf{ver}(pk, x, \sigma) = \mathsf{true}$. If $(x, \sigma)$ is a forgery, then $A$ terminates with output $(x, \sigma)$. Otherwise, $A$ continues the simulation of $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}}$.

It is easy to see that $A$ is polynomial-time because $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}}$ is strictly bounded.

Let $B(1^\eta, a)$ be the set of runs of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ where, at some point during the run, $\mathcal{P}_{\mathrm{sig}}$ is uncorrupted and verifies a signature $\sigma$ for a message $x$ where $x$ has never been signed before and $\mathsf{ver}(pk, x, \sigma) = \mathsf{true}$. By $\overline{B(1^\eta, a)}$ we denote the complement, i.e., the set of runs of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ which are not in $B(1^\eta, a)$. Since every run of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ that is simulated by $A$ corresponds to a run of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$, it is easy to see that:

$$\begin{aligned}\Pr\left[B(1^\eta, a)\right] &= \Pr[(pk, sk) \leftarrow \mathsf{gen}(1^\eta), (x, \sigma) \leftarrow A^{\mathsf{sig}(sk,\cdot)}(1^\eta, a, pk) : \\ &\qquad \mathsf{ver}(pk, x, \sigma) = \mathsf{true} \text{ and } A \text{ has not previously called } \mathsf{sig}(sk, x)] \\ &= \mathrm{Adv}^{\text{uf-cma}}_{A,\Sigma}(1^\eta, a) \ .\end{aligned}$$

By assumption, $\Sigma$ is UF-CMA secure and, hence, $\Pr\left[B(1^\eta, a)\right]$ is negligible (as a function in $\eta$ and $a$).

Because the behavior of $\mathcal{P}_{\mathrm{sig}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}}$ is identical as long as no forgery occurs or once $\mathcal{P}_{\mathrm{sig}}$ (and hence, by definition of $\mathcal{S}$, $\mathcal{F}_{\mathrm{sig}}$) is corrupted, it is easy to see that every run $\rho$ of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ which is *not* in $B(1^\eta, a)$ corresponds to a run $\rho'$ of $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a)$ such that both $\rho$ and $\rho'$ have the same probability and the view and probabilistic choices of $\mathcal{E}$ are the same in both runs. Formally, there exists an injective mapping from runs $\rho$ of $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a)$ excluding $B(1^\eta, a)$ to runs $\rho'$ of $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a)$ such that both $\rho$ and $\rho'$ have the same probability and the same overall output on $\mathsf{decision}$. We obtain that:

$$\begin{aligned}\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a) = 1 \wedge \overline{B(1^\eta, a)}\right] &\leq \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a) = 1\right] \text{ and} \\ \Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a) \neq 1 \wedge \overline{B(1^\eta, a)}\right] &\leq \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a) \neq 1\right] \ .\end{aligned}$$

It immediately follows that:

$$\begin{aligned}0 &\leq \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a) = 1\right] - \Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a) = 1 \wedge \overline{B(1^\eta, a)}\right] \\ &\leq \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a) = 1\right] - \Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a) = 1 \wedge \overline{B(1^\eta, a)}\right] + \\ &\quad \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{sig}})(1^\eta, a) \neq 1\right] - \Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{sig}})(1^\eta, a) \neq 1 \wedge \overline{B(1^\eta, a)}\right] \\ &= \Pr\left[B(1^\eta, a)\right] \ .\end{aligned}$$

From this, we conclude that:

$$\begin{aligned}
&|\Pr[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a) = 1] - \Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}})(1^\eta, a) = 1]| \\
&\quad = \left|\Pr[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a){=}1 \wedge B(1^\eta, a)] + \Pr\left[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a){=}1 \wedge \overline{B(1^\eta, a)}\right] - \Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}})(1^\eta, a){=}1]\right| \\
&\quad \leq \Pr[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a){=}1 \wedge B(1^\eta, a)] + \left|\Pr\left[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a){=}1 \wedge \overline{B(1^\eta, a)}\right] - \Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}})(1^\eta, a){=}1]\right| \\
&\quad \leq \Pr[B(1^\eta, a)] + \Pr[B(1^\eta, a)] \\
&\quad = 2 \cdot \mathrm{Adv}^{\text{uf-cma}}_{A,\Sigma}(1^\eta, a) \ ,
\end{aligned}$$

which is negligible (as a function in $\eta$ and $a$) because $\Sigma$ is UF-CMA secure. Hence, $\mathcal{E} \mid \mathcal{P}_{\text{sig}} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}}$, i.e., $\mathcal{P}_{\text{sig}} \leq \mathcal{F}_{\text{sig}}$.

### B.1.2 $\mathcal{P}_{\text{sig}} \leq \mathcal{F}_{\text{sig}} \Rightarrow \Sigma$ is UF-CMA Secure

We now show that $\Sigma$ is UF-CMA secure if $\mathcal{P}_{\text{sig}} \leq \mathcal{F}_{\text{sig}}$. Therefore, we assume that $\Sigma$ is *not* UF-CMA secure, i.e., there exists a UF-CMA adversary $A^{O(\cdot)}$ such that the UF-CMA advantage of $A$ against $\Sigma$ is not negligible (see Definition 10). Using $A$, we construct an environment $\mathcal{E} \in \mathrm{Env}(\mathcal{P}_{\text{sig}})$ such that $\mathcal{E}$ distinguishes $\mathcal{P}_{\text{sig}}$ from $\mathcal{S} \mid \mathcal{F}_{\text{sig}}$ for every simulator $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}_{\text{sig}}}(\mathcal{F}_{\text{sig}})$.

We define $\mathcal{E}$ to be a master IITM (i.e., $\mathcal{E}$ has an input tape named $\mathsf{start}$) which has an output tape named $\mathsf{decision}$ and connects to the I/O and network tapes of $\mathcal{P}_{\text{sig}}$ (and, hence, $\mathcal{S} \mid \mathcal{F}_{\text{sig}}$). In mode $\mathsf{CheckAddress}$, $\mathcal{E}$ accepts every message. Next, we describe the mode $\mathsf{Compute}$ of $\mathcal{E}$ in an interaction with $\mathcal{P}_{\text{sig}}$, but, of course, $\mathcal{P}_{\text{sig}}$ can be replaced by $\mathcal{S} \mid \mathcal{F}_{\text{sig}}$:

(a) Upon the first activation with external input $a$ on tape $\mathsf{start}$, $\mathcal{E}$ sends $\mathsf{PubKey}?$ to $\mathcal{P}_{\text{sig}}$ on an I/O tape, say on $\mathsf{io}_1^{\text{in}}$. Then, $\mathcal{E}$ waits for receiving $\mathsf{Init}$ from $\mathcal{P}_{\text{sig}}$ on the network tape (i.e., on $\mathsf{net}^{\text{out}}_{\mathcal{P}_{\text{sig}}}$) and replies with $(\mathsf{false}, \varepsilon, \varepsilon)$ (where $\varepsilon$ is the empty bit string) on $\mathsf{net}^{\text{in}}_{\mathcal{P}_{\text{sig}}}$, i.e., $\mathcal{E}$ does not corrupt $\mathcal{P}_{\text{sig}}$ and $\mathcal{P}_{\text{sig}}$ generates a fresh key pair. Then, $\mathcal{E}$ waits for receiving $(\mathsf{PubKey}, pk)$ from $\mathcal{P}_{\text{sig}}$ on $\mathsf{io}_1^{\text{out}}$.

(b) Then, $\mathcal{E}$ simulates the algorithm $A^{O(\cdot)}(1^\eta, a, pk)$. Whenever $A$ asks its signing oracle $O$ to sign a message $x$, then $\mathcal{E}$ sends $(\mathsf{Sign}, x)$ to $\mathcal{P}_{\text{sig}}$ on $\mathsf{io}_1^{\text{in}}$ and waits for receiving $(\mathsf{Signature}, \sigma)$ from $\mathcal{P}_{\text{sig}}$ on $\mathsf{io}_1^{\text{out}}$. Then, $\mathcal{E}$ continues the simulation of $A$ as if $O$ returned $\sigma$. The output of $A$ will be a pair $(x_0, \sigma_0)$.

(c) Then, $\mathcal{E}$ checks whether $(x_0, \sigma_0)$ constitutes a forgery, i.e., $x_0$ has not been signed and $\mathcal{P}_{\text{sig}}$, upon request $(\mathsf{Verify}, pk, x_0, \sigma_0)$ on $\mathsf{io}_1^{\text{in}}$, returns $(\mathsf{VerResult}, \mathsf{true})$. If $(x_0, \sigma_0)$ constitutes a forgery and $\mathcal{P}_{\text{sig}}$ is not corrupted ($\mathcal{E}$ can check this by sending $\mathsf{CorrStatus}?$ to $\mathcal{P}_{\text{sig}}$ on $\mathsf{io}_1^{\text{in}}$), then $\mathcal{E}$ terminates with output 1 on $\mathsf{decision}$, otherwise with output 0. (Note that $\mathcal{E}$ never corrupts $\mathcal{P}_{\text{sig}}$ but $\mathcal{S}$ might have corrupted $\mathcal{F}_{\text{sig}}$.)

If at some point in the description above, $\mathcal{E}$ waits for receiving a message but the input is not as expected or on an unexpected tape (this will never happen in the real world, i.e., in a run of $\mathcal{E} \mid \mathcal{P}_{\text{sig}}$, but possibly in the ideal world, i.e., in a run of $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}}$), then $\mathcal{E}$ terminates with output 0 on $\mathsf{decision}$.

In the ideal world (i.e., in a run of $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}}$) $\mathcal{E}$ will never output 1 on $\mathsf{decision}$, i.e.:

$$\Pr[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{sig}})(1^\eta, a) = 1] = 0 \ ,$$

because, by definition, an uncorrupted $\mathcal{F}_{\text{sig}}$ will never return $(\mathsf{VerResult}, \mathsf{true})$ upon a $\mathsf{Verify}$ request for a message that has not previously been signed using $\mathcal{F}_{\text{sig}}$. The probability that $\mathcal{E}$ outputs 1 on $\mathsf{decision}$ in the real world is exactly the advantage of $A$ against $\Sigma$:

$$\begin{aligned}
\Pr[(\mathcal{E} \mid \mathcal{P}_{\text{sig}})(1^\eta, a) = 1] &= \Pr[(pk, sk) \leftarrow \mathsf{gen}(1^\eta), (x, \sigma) \leftarrow A^{\mathsf{sig}(sk,\cdot)}(1^\eta, a, pk) : \\
&\qquad \mathsf{ver}(pk, x, \sigma) = \mathsf{true} \text{ and } A \text{ has not previously called } \mathsf{sig}(sk, x)] \\
&= \mathrm{Adv}^{\text{uf-cma}}_{A,\Sigma}(1^\eta, a) \ .
\end{aligned}$$

**Parameters:** – $n > 0$ {*number of I/O tape pairs*
– $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$ {*public-key encryption scheme with associated plaintext domain* $D_\Sigma = \{D_\Sigma(\eta)\}_{\eta\in\mathbb{N}}$
**Tapes:** from/to $\mathsf{IO}_i$ ($i \in \{1, \ldots, n\}$): $(\mathsf{io}_i^{\text{in}}, \mathsf{io}_i^{\text{out}})$; from/to NET: $(\mathsf{net}^{\text{in}}_{\mathcal{P}_{\text{pke}}}, \mathsf{net}^{\text{out}}_{\mathcal{P}_{\text{pke}}})$
**State, CheckAddress, and Initialization:** Just like $\mathcal{P}_{\text{sig}}$, see Figure 9.
**Compute:**

**recv** PubKey? **from** $\mathsf{IO}_i$**: send** (PubKey, pk) **to** $\mathsf{IO}_i$ {*return public key*

**recv** (Enc, $pk, x$) **from** $\mathsf{IO}_i$ **s.t.** $x \in D_\Sigma(\eta)$**:** $y \leftarrow \mathsf{enc}(pk, x)$; **send** (Ciphertext, $y$) **to** $\mathsf{IO}_i$ {*encrypt x, return ciphertext*

**recv** (Dec, $y$) **from** $\mathsf{IO}_i$**:** $x \leftarrow \mathsf{dec}(\mathsf{sk}, y)$; **send** (Plaintext, $x$) **to** $\mathsf{IO}_i$ {*decrypt y, return plaintext*

**recv** CorrStatus? **from** $\mathsf{IO}_i$**: send** (CorrStatus, corrupted) **to** $\mathsf{IO}_i$ {*corruption status request*

Figure 10: The realization $\mathcal{P}_{\text{pke}}$ of $\mathcal{F}_{\text{pke}}$. See Section 4.1 for notational conventions.

We obtain that:

$$|\Pr[(\mathcal{E} \,|\, \mathcal{P}_{\text{sig}})(1^\eta, a) = 1] - \Pr[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{sig}})(1^\eta, a) = 1]| = \text{Adv}^{\text{uf-cma}}_{\Sigma,A}(1^\eta, a) \ .$$

Hence, by assumption that $A$ is successful, we conclude that $\mathcal{E} \,|\, \mathcal{P}_{\text{sig}} \not\equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{sig}}$, i.e., $\mathcal{P}_{\text{sig}} \not\leq \mathcal{F}_{\text{sig}}$.

This concludes the proof of Theorem 6. □

## B.2 Proof of Theorem 7

We prove Theorem 7, i.e., that $\mathcal{P}_{\text{pke}}$ realizes $\mathcal{F}_{\text{pke}}$ if and only if $\Sigma$ is IND-CCA2 secure. This proof is along the lines of the proof in [6] (version of December 2005). Let $n$, $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$, $p$, and $L$ be given as in the theorem.

### B.2.1 $\Sigma$ is IND-CCA2 Secure $\Rightarrow \mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}}$

First, we assume that $\Sigma$ is IND-CCA2 secure and show that $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}}$.

We use a straightforward simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{pke}})$ that accepts all messages in mode CheckAddress, forwards the Init request from $\mathcal{F}_{\text{pke}}$ to the environment, and completes initialization with a freshly generated key pair in the uncorrupted case and with the key pair provided by the environment upon corruption. More formally, $\mathcal{S}$ is defined exactly as the simulator in the proof of Theorem 6 (see Appendix B.1.1), except that Corrupt requests from the environment are not forwarded to $\mathcal{F}_{\text{pke}}$ (recall that $\mathcal{P}_{\text{pke}}$ and $\mathcal{F}_{\text{pke}}$ both do not allow adaptive corruption, in contrast to $\mathcal{P}_{\text{sig}}$ and $\mathcal{F}_{\text{sig}}$). It is easy to see that $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$ is environmentally strictly bounded, i.e., $\mathcal{S} \in \text{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{pke}})$.

Let $\mathcal{E} \in \text{Env}(\mathcal{P}_{\text{pke}})$ be an environment of $\mathcal{P}_{\text{pke}}$. Using $\mathcal{E}$, we construct an IND-CCA2 adversary $A$ on $\Sigma$ such that, basically, $A$ is successful if $\mathcal{E}$ successfully distinguishes between $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$.

To simplify the presentation of the adversary $A$, without loss of generality, we assume the following:

(i) The first request $\mathcal{E}$ sends to $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$) in any run is an PubKey? request. Then, $\mathcal{E}$ receives Init from $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$) on the network tape and completes initialization of $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$) by sending $(\mathsf{false}, \varepsilon, \varepsilon)$ ($\varepsilon$ is the empty bit string) to the network interface of $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$). That is, $\mathcal{E}$ does not corrupt $\mathcal{P}_{\text{pke}}$ (or $\mathcal{F}_{\text{pke}}$) and directly completes initialization. (It is easy to see that, upon corruption, $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$ are indistinguishable; in fact, the observational behavior of $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$ would be exactly the same.) Furthermore, $\mathcal{E}$ never sends a second PubKey? request.

(ii) In any run, $\mathcal{E}$ never sends corruption status requests (CorrStatus?). (By definition of $\mathcal{S}$, $\mathcal{P}_{\text{pke}}$ and $\mathcal{F}_{\text{pke}}$ always agree on the corruption status. Hence, this request would not help $\mathcal{E}$ to distinguishes between $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$.)

(iii) In any run, $\mathcal{E}$ only sends encryption requests with the correct public key, i.e., the public key $pk$ in every Enc request is the public key that $\mathcal{E}$ received as response to the PubKey? request. (It is easy to see that $\mathcal{E}$ has no advantage of sending Enc requests with a different key.)

(iv) There exists a polynomial $n_{\mathsf{Enc}}$ such that the overall number of encryption requests that $\mathcal{E}$ sends in any run (with security parameter $\eta$ and external input $a$) is exactly $n_{\mathsf{Enc}}(\eta + |a|)$. (Note that the number of $\mathsf{Enc}$ requests sent by $\mathcal{E}$ is polynomially bounded in $\eta + |a|$ because $\mathcal{E}$ is universally bounded.) In the following, we just write $n_{\mathsf{Enc}}$ to denote $n_{\mathsf{Enc}}(\eta + |a|)$.

We now define an IND-CCA2 adversary $A = (A_1, A_2)$ against $\Sigma$. The first part $A_1^{O(\cdot)}(1^\eta, a, pk)$ (where $O(\cdot)$ is a decryption oracle) is defined as follows: At first $A_1$ chooses $h \in \{1, \ldots, n_{\mathsf{Enc}}\}$ uniformly at random. Then, $A_1$ simulates a run of $\mathcal{E}$ as follows:

- $A_1$ starts the simulation of $\mathcal{E}$ with security parameter $\eta$ and external input $a$.
- When $\mathcal{E}$ sends the $\mathsf{PubKey}?$ request, $A_1$ sends $\mathsf{Init}$ on the network tape to $\mathcal{E}$. (This is what $\mathcal{E}$ expects in a run with $\mathcal{P}_{\text{pke}}$ or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$.)
- When $\mathcal{E}$ replies with $(\mathsf{false}, \varepsilon, \varepsilon)$, $A_1$ sends the public key $(\mathsf{PubKey}, pk)$ to $\mathcal{E}$, as the response to the $\mathsf{PubKey}?$ request. (This is what $\mathcal{E}$ expects in a run with $\mathcal{P}_{\text{pke}}$ or $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$.)
- When $\mathcal{E}$ sends an $\mathsf{Enc}$ request, say the $i$-th encryption request and the plaintext is $x_i$ (for some $i \in \{1, \ldots, h\}$), then $A_1$ does the following: If $i < h$, then $A_1$ computes $y_i \leftarrow \mathsf{enc}(pk, x_i)$, records the pair $(x_i, y_i)$ for later decryption, and sends $(\mathsf{Ciphertext}, y_i)$ to $\mathcal{E}$. Otherwise (i.e., $i = h$), $A_1$ computes $\overline{x}_h \leftarrow L(1^\eta, x_h)$. Furthermore, $A_1$ computes a bit string $s$ that encodes $pk$, $x_h$, all recorded plaintext/ciphertext pairs, and all information that $A_2$ needs to continue the simulation of $\mathcal{E}$. Then, $A_1$ outputs $(x_h, \overline{x}_h, s)$.
- When $\mathcal{E}$ sends a $\mathsf{Dec}$ request, say for the ciphertext $y$, then $A_1$ does the following: At first, similarly to $\mathcal{F}_{\text{pke}}$, $A_1$ checks whether there exists a plaintext $x$ such that the pair $(x, y)$ has been recorded upon encryption (see above). If there exists more than one such plaintext, then $A_1$ sends the error message $(\mathsf{Plaintext}, \perp)$ to $\mathcal{E}$. If there exists exactly one such $x$, then $A_1$ sends $(\mathsf{Plaintext}, x)$ to $\mathcal{E}$. And if there exists no such $x$, then $A_1$ decrypts $y$ using its decryption oracle $O$ and sends the obtained plaintext to $\mathcal{E}$.

The second part $A_2^{O(\cdot)}(s, y^*)$ reconstructs the information stored in $s$, records the pair $(x_h, y^*)$, and continues the simulation of the run of $\mathcal{E}$ as follows:

- First, $A_2$ sends $(\mathsf{Ciphertext}, y^*)$ to $\mathcal{E}$. (Recall that $\mathcal{E}$ just sent an encryption request and is waiting for a ciphertext.)
- When $\mathcal{E}$ sends an $\mathsf{Enc}$ request, say the $i$-th encryption request and the plaintext is $x_i$ (for some $i \in \{h+1, \ldots, n_{\mathsf{Enc}}\}$), then, similarly to $\mathcal{F}_{\text{pke}}$, $A_2$ computes $\overline{x}_i \leftarrow L(1^\eta, x_i)$ and $y \leftarrow \mathsf{enc}(pk, \overline{x}_i)$. Then, $A_2$ records the pair $(x_i, y)$ (for later decryption) and sends $(\mathsf{Ciphertext}, y)$ to $\mathcal{E}$.
- When $\mathcal{E}$ sends a $\mathsf{Dec}$ request, then $A_2$ behaves exactly as $A_1$, see above.
- When the simulated run stops, then $A_2$ outputs 1 if $\mathcal{E}$ has output 1 on $\mathsf{decision}$, otherwise, $A_2$ outputs 0.

Note that it always holds that $|x_h| = |\overline{x}_h|$ because $L$ is length preserving. Furthermore, $A_2$, by definition, never asks its oracle $O$ for the decryption of $y^*$. Since $\mathcal{E}$ is universally bounded, $A_1$ and $A_2$ are polynomial-time. Hence, $A$ is a valid IND-CCA2 adversary against $\Sigma$.

Before we analyze the advantage of $A$ against $\Sigma$, we note that the following is easy to see:

$$\mathrm{Adv}^{\text{ind-cca2}}_{A,\Sigma}(1^\eta, a) = \left| \Pr\left[\mathrm{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1\right] - \Pr\left[\mathrm{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1\right] \right| \tag{4}$$

where for all $b \in \{0, 1\}$:

$$\begin{aligned} \mathrm{Exp}^{\text{ind-cca2-}b}_{A,\Sigma}(1^\eta, a)\colon\ & (pk, sk) \leftarrow \mathsf{gen}(1^\eta); (x_0, x_1, s) \leftarrow A_1^{\mathsf{dec}(sk,\cdot)}(1^\eta, a, pk); \\ & y \leftarrow \mathsf{enc}(pk, x_b); b' \leftarrow A_2^{\mathsf{dec}(sk,\cdot)}(s, y); \textbf{return } b' \ . \end{aligned} \tag{5}$$

By construction of $A$, it is easy to see that for all $\eta, a$:

$$\Pr\left[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{pke}})(1^\eta, a) = 1\right] = \Pr\left[\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = 1\right] \text{ and} \tag{6}$$

$$\Pr\left[(\mathcal{E} \mid \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right] = \Pr\left[\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = n_{\mathsf{Enc}}\right] . \tag{7}$$

Furthermore, it is easy to see that for all $\eta, a$ and all $i \in \{1, \ldots, n_{\mathsf{Enc}} - 1\}$:

$$\Pr\left[\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = i\right] = \Pr\left[\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = i+1\right] \tag{8}$$

because, by construction of $A$, in both experiments ($\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}$ with $h = i$ and $\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}$ with $h = i+1$), it is the case that the first $i$ encryptions are encryptions of the real messages and all later encryptions are encryptions of leakages.

We conclude that for all $\eta, a$:

$$\begin{aligned}
&\text{Adv}^{\text{ind-cca2}}_{A,\Sigma}(1^\eta, a) \\
&\overset{(4)}{=} \left|\Pr\left[\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1\right] - \Pr\left[\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1\right]\right| \\
&= \frac{1}{n_{\mathsf{Enc}}} \cdot \left|\sum_{i=1}^{n_{\mathsf{Enc}}} \Pr\left[\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = i\right] - \Pr\left[\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = i\right]\right| \\
&\overset{(8)}{=} \frac{1}{n_{\mathsf{Enc}}} \cdot \left|\Pr\left[\text{Exp}^{\text{ind-cca2-1}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = 1\right] - \Pr\left[\text{Exp}^{\text{ind-cca2-0}}_{A,\Sigma}(1^\eta, a) = 1 \mid h = n_{\mathsf{Enc}}\right]\right| \\
&\overset{(6),(7)}{=} \frac{1}{n_{\mathsf{Enc}}} \cdot \left|\Pr\left[(\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{pke}})(1^\eta, a) = 1\right] - \Pr\left[(\mathcal{E} \mid \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right]\right| .
\end{aligned}$$

By assumption, $\Sigma$ is IND-CCA2 secure. Hence, $\text{Adv}^{\text{ind-cca2}}_{A,\Sigma}(1^\eta, a)$ is negligible (as a function in $\eta$ and $a$). Since $n_{\mathsf{Enc}}$ is a polynomial in $\eta$ and $a$, we obtain that $\mathcal{E} \mid \mathcal{P}_{\text{pke}} \equiv \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\text{pke}}$, i.e., $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}}$.

#### B.2.2 $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}} \Rightarrow \Sigma$ is IND-CCA2 Secure

We now show that $\Sigma$ is IND-CCA2 secure if $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}}$. Therefore, we assume that $\Sigma$ is *not* IND-CCA2 secure, i.e., there exists an IND-CCA2 adversary $A = (A_1, A_2)$ with non-negligible IND-CCA2 advantage against $\Sigma$ (see Definition 12). Using $A$, we construct an environment $\mathcal{E} \in \text{Env}(\mathcal{P}_{\text{pke}})$ such that $\mathcal{E}$ distinguishes $\mathcal{P}_{\text{pke}}$ from $\mathcal{S} \mid \mathcal{F}_{\text{pke}}$ for every simulator $\mathcal{S} \in \text{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{pke}})$.

We define $\mathcal{E}$ to be a master IITM (i.e., an IITM with a tape named $\mathsf{start}$) with an output tape named $\mathsf{decision}$ and tapes to connect to $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \mid \mathcal{F}_{\text{pke}}$ for any $\mathcal{S} \in \text{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{pke}})$). In what follows, when we say $\mathcal{E}$ encrypts/decrypts using $\mathcal{P}_{\text{pke}}$, we mean using $\mathcal{P}_{\text{pke}}$ or $\mathcal{S} \mid \mathcal{F}_{\text{pke}}$, depending on which system $\mathcal{E}$ is connected to. In mode $\mathsf{CheckAddress}$ $\mathcal{E}$ accepts every incoming message and in mode $\mathsf{Compute}$ it operates as follows:

- Upon the first activation with external input $a$ on tape $\mathsf{start}$, $\mathcal{E}$ sends $\mathsf{PubKey}?$ to $\mathcal{P}_{\text{pke}}$ on an I/O tape, say on $\mathsf{io}^{\text{in}}_1$. Then, $\mathcal{E}$ waits for receiving $\mathsf{Init}$ from $\mathcal{P}_{\text{pke}}$ on the network tape (i.e., on $\mathsf{net}^{\text{out}}_{\mathcal{P}_{\text{pke}}}$) and replies with $(\mathsf{false}, \varepsilon, \varepsilon)$ (where $\varepsilon$ is the empty bit string) on $\mathsf{net}^{\text{in}}_{\mathcal{P}_{\text{pke}}}$, i.e., $\mathcal{E}$ does not corrupt $\mathcal{P}_{\text{pke}}$ and $\mathcal{P}_{\text{pke}}$ generates a fresh key pair. Then, $\mathcal{E}$ waits for receiving $(\mathsf{PubKey}, pk)$ from $\mathcal{P}_{\text{pke}}$ on $\mathsf{io}^{\text{out}}_1$.
- Then, $\mathcal{E}$ simulates a run of the adversary $A_1^{O(\cdot)}(1^\eta, a, pk)$ as follows:
  - Whenever $A_1$ asks its decryption oracle $O$ to decrypt a ciphertext, say $y$, then $\mathcal{E}$ decrypts $y$ using $\mathcal{P}_{\text{pke}}$, i.e., $\mathcal{E}$ sends $(\mathsf{Dec}, y)$ to $\mathcal{P}_{\text{pke}}$ and waits for receiving $(\mathsf{Plaintext}, x)$ from $\mathcal{P}_{\text{pke}}$. Then, $\mathcal{E}$ continues simulating $A_1$ as if $O$ returned $x$.

– When $A_1$ halts and outputs $(x_0, x_1, s)$, then $\mathcal{E}$ chooses a bit $b \in \{0,1\}$ uniformly at random and encrypts $x_b$ under $pk$ using $\mathcal{P}_{\text{pke}}$ by sending $(\mathsf{Enc}, pk, x_b)$ the request and waits for receiving $(\mathsf{Ciphertext}, y^*)$.

- Then, $\mathcal{E}$ simulates a run of $A_2^{O(\cdot)}(1^\eta, s, y^*)$ as follows:
  – Whenever $A_2$ asks its decryption oracle $O$ to decrypt a ciphertext, say $y$, then $\mathcal{E}$ decrypts $y$ using $\mathcal{P}_{\text{pke}}$ and continues simulating $A_2$ as described above for $A_1$.
  – When $A_2$ halts and outputs a bit $b' \in \{0,1\}$, then $\mathcal{E}$ does the following: First, $\mathcal{E}$ checks whether $\mathcal{P}_{\text{pke}}$ is corrupted (it should not be corrupted because $\mathcal{E}$ did not corrupt $\mathcal{P}_{\text{pke}}$ but if $\mathcal{E}$ interacts with $\mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$ instead of $\mathcal{P}_{\text{pke}}$, then $\mathcal{S}$ might have corrupted $\mathcal{F}_{\text{pke}}$), i.e., $\mathcal{E}$ sends $\mathsf{CorrStatus}?$ to $\mathcal{P}_{\text{pke}}$ and waits for receiving $(\mathsf{Corrupted}, \mathit{corrupt})$ from $\mathcal{P}_{\text{pke}}$. If $\mathit{corrupt} = \mathsf{false}$ and $b = b'$, then $\mathcal{E}$ outputs 1 on the tape $\mathsf{decision}$. Otherwise, if $\mathit{corrupt} = \mathsf{false}$ (i.e., $b \neq b'$), then $\mathcal{E}$ outputs 0 on $\mathsf{decision}$. Otherwise (i.e., $\mathit{corrupt} = \mathsf{true}$), then $\mathcal{E}$ outputs a random bit on $\mathsf{decision}$ (i.e., $\mathcal{E}$ chooses $b'' \in \{0,1\}$ uniformly at random and outputs $b''$ on $\mathsf{decision}$).

If at some point above $\mathcal{E}$ waits for receiving a message but this message is not as expected, then $\mathcal{E}$ outputs a random bit on $\mathsf{decision}$. It is easy to see that $\mathcal{E}$ is universally bounded, i.e., $\mathcal{E} \in \text{Env}(\mathcal{P}_{\text{pke}})$.

In the real world (i.e., in runs of $\mathcal{E} \,|\, \mathcal{P}_{\text{pke}}$), $\mathcal{E}$ always receives what it expects because of the definition of $\mathcal{P}_{\text{pke}}$. Furthermore, $\mathcal{P}_{\text{pke}}$ never gets corrupted (i.e., $\mathit{corrupt} = \mathsf{false}$) and, hence, the simulation of $A$ is exactly as in the IND-CCA2 experiment, i.e., for all $\hat{b} \in \{0,1\}$:

$$\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1 \mid b = \hat{b}\right] = \Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-}\hat{b}}(1^\eta, a) = \hat{b}\right] \quad . \tag{9}$$

(See (5) in Appendix B.2.1 for the definition of $\text{Exp}_{A,\Sigma}^{\text{ind-cca2-}\hat{b}}$.) From this, we obtain:

$$\begin{aligned}
\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right] &= \frac{1}{2}\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1 \mid b = 1\right] + \frac{1}{2}\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1 \mid b = 0\right] \\
&\overset{(9)}{=} \frac{1}{2}\Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-1}}(1^\eta, a) = 1\right] + \frac{1}{2}\Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-0}}(1^\eta, a) = 0\right] \\
&= \frac{1}{2}\Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-1}}(1^\eta, a) = 1\right] + \frac{1}{2} - \frac{1}{2}\Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-0}}(1^\eta, a) = 1\right] \quad .
\end{aligned} \tag{10}$$

In the ideal world (i.e., in runs of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$), $\mathcal{E}$ outputs 1 with probability exactly 1/2: If $\mathcal{E}$ receives some unexpected input or if $\mathcal{F}_{\text{pke}}$ gets corrupted (i.e., $\mathit{corrupt} = \mathsf{true}$) then, by definition of $\mathcal{E}$, $\mathcal{E}$ outputs a random bit. Otherwise, $\mathcal{E}$ always receives what it expects and $\mathcal{F}_{\text{pke}}$ is uncorrupted. In this case, $\mathcal{E}$ outputs 1 iff $b = b'$. Because the leakage algorithm $L$ leaks at most the length, by definition, there exists a PPT algorithm $T$ such that for all $x$ (in particular for $x \in \{x_0, x_1\}$) the distribution of $T(1^\eta, 1^{|x|}))$ equals the distribution of $L(1^\eta, x)$. That is, since $A_1$ outputs plaintexts of the same length (i.e., $|x_0| = |x_1|$), the input to $A_2$ (as a random variable) is independent of $b$ (as a random variable) and, hence, the output of $A_2$ (as a random variable) is independent of $b$. We conclude that $b = b'$ occurs with probability exactly 1/2 and we obtain:

$$\Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{pke}})(1^\eta, a) = 1\right] = \frac{1}{2} \quad . \tag{11}$$

We conclude that:

$$\begin{aligned}
&\left|\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right] - \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{pke}})(1^\eta, a) = 1\right]\right| \\
&\overset{(10),(11)}{=} \frac{1}{2} \cdot \left|\Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-1}}(1^\eta, a) = 1\right] - \Pr\left[\text{Exp}_{A,\Sigma}^{\text{ind-cca2-0}}(1^\eta, a) = 1\right]\right| \\
&\overset{(4)}{=} \frac{1}{2} \cdot \text{Adv}_{A,\Sigma}^{\text{ind-cca2}}(1^\eta, a) \quad ,
\end{aligned}$$

which, by assumption, is non-negligible (as a function in $\eta$ and $a$). Hence, $\mathcal{E} \,|\, \mathcal{P}_{\text{pke}} \not\equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{pke}}$ and we conclude that $\mathcal{P}_{\text{pke}} \not\leq \mathcal{F}_{\text{pke}}$.

This concludes the proof of Theorem 7. □

## B.3 Proof of Theorem 8

We now prove Theorem 8, i.e., that $\mathcal{P}_{\text{pke}}$ realizes $\mathcal{F}_{\text{rpke}}$ if and only if $\Sigma$ is IND-RCCA secure. Let $n$, $\Sigma = (\mathsf{gen}, \mathsf{enc}, \mathsf{dec})$, $p$, and $L$ be given as in the theorem.

### B.3.1 $\Sigma$ is IND-RCCA Secure $\Rightarrow \mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$

First, we assume that $\Sigma$ is IND-RCCA secure and show that $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$.

Let $\mathcal{S}$ be the simulator defined in Appendix B.2.1 (to prove that $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{pke}}$). It is easy to see that $\mathcal{S} \in \text{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{rpke}})$, i.e., $\mathcal{S}$ is also a valid simulator for $\mathcal{F}_{\text{rpke}}$.

Let $\mathcal{E} \in \text{Env}(\mathcal{P}_{\text{pke}})$ be an environment of $\mathcal{P}_{\text{pke}}$. Using $\mathcal{E}$, we construct an IND-RCCA adversary $A$ on $\Sigma$ such that, basically, $A$ is successful if $\mathcal{E}$ successfully distinguishes between $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$.

As in Appendix B.2.1, to simplify the presentation of the adversary $A$, without loss of generality, we assume the following:

(i) The first request $\mathcal{E}$ sends to $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$) in any run is an $\mathsf{PubKey}?$ request. Then, $\mathcal{E}$ receives $\mathsf{Init}$ from $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$) on the network tape and completes initialization of $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$) by sending $(\mathsf{false}, \varepsilon, \varepsilon)$ ($\varepsilon$ is the empty bit string) to the network interface of $\mathcal{P}_{\text{pke}}$ (or $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$). That is, $\mathcal{E}$ does not corrupt $\mathcal{P}_{\text{pke}}$ (or $\mathcal{F}_{\text{rpke}}$) and directly completes initialization. (It is easy to see that, upon corruption, $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$ are indistinguishable; in fact, the observational behavior of $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$ would be exactly the same.) Furthermore, $\mathcal{E}$ never sends a second $\mathsf{PubKey}?$ request.

(ii) In any run, $\mathcal{E}$ never sends corruption status requests ($\mathsf{CorrStatus}?$). (By definition of $\mathcal{S}$, $\mathcal{P}_{\text{pke}}$ and $\mathcal{F}_{\text{rpke}}$ always agree on the corruption status. Hence, this request would not help $\mathcal{E}$ to distinguishes between $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$.)

(iii) In any run, $\mathcal{E}$ only sends encryption requests with the correct public key, i.e., the public key $pk$ in every $\mathsf{Enc}$ request is the public key that $\mathcal{E}$ received as response to the $\mathsf{PubKey}?$ request. (It is easy to see that $\mathcal{E}$ has no advantage of sending $\mathsf{Enc}$ requests with a different key.)

(iv) There exists a polynomial $n_{\mathsf{Enc}}$ such that the overall number of encryption requests that $\mathcal{E}$ sends in any run (with security parameter $\eta$ and external input $a$) is exactly $n_{\mathsf{Enc}}(\eta + |a|)$. (Note that the number of $\mathsf{Enc}$ requests sent by $\mathcal{E}$ is polynomially bounded in $\eta + |a|$ because $\mathcal{E}$ is universally bounded.) In the following, we just write $n_{\mathsf{Enc}}$ to denote $n_{\mathsf{Enc}}(\eta + |a|)$.

In the following, to simplify notation, let a security parameter $\eta \in \mathbb{N}$ and an external input $a \in \{0,1\}^*$ be fixed. Furthermore, we often omit the arguments $(1^\eta, a)$ and $(\eta + |a|)$. For example, we simply write $n_{\mathsf{Enc}}$ instead of $n_{\mathsf{Enc}}(\eta + |a|)$ and $\Pr[\mathcal{E} \,|\, \mathcal{P} = 1]$ instead of $\Pr[(\mathcal{E} \,|\, \mathcal{P})(1^\eta, a) = 1]$.

We now define an IND-RCCA adversary $A = (A_1, A_2)$ against $\Sigma$. It is defined similarly to the adversary $A$ in Appendix B.2.1. It only differs slightly upon encryption and decryption requests. $A_1^{O(\cdot)}(1^\eta, a, pk)$ first chooses $h \in \{1, \ldots, n_{\mathsf{Enc}}\}$ uniformly at random and then simulates a run of $\mathcal{E}$ with security parameter $\eta$ and external input $a$: Upon the $\mathsf{PubKey}?$ request of $\mathcal{E}$, $A_1$ sends the public key $pk$ to $\mathcal{E}$. The first $h-1$ encryption requests (i.e., for the plaintexts $x_1, \ldots, x_{h-1}$) are answered by simply encrypting the plaintext under the public key $pk$. In contrast to $A$ in Appendix B.2.1, the plaintext/ciphertext pair is not recorded for later decryption. Decryption requests are answered by using the decryption oracle $O$ of $A_1$. When $\mathcal{E}$ sends the $h$-th encryption request, then $A_1$ halts and outputs $(x_h, \overline{x}_h, s)$ where $x_h$ is the plaintext in this encryption request, $\overline{x}_h$ is the leakage $\overline{x}_h \leftarrow L(1^\eta, x_h)$ of $x_h$, and $s$ is a bit string that encodes all information that $A_2$ needs to continue the simulation of $\mathcal{E}$. In the IND-RCCA experiment, $A_2^{O(\cdot)}(1^\eta, s, y^*)$ then receives the encryption $y^*$ of $x_h$ if $b = 0$ and of $\overline{x}_h$ of $b = 1$ and has to guess $b$. Similar to $A_1$, $A_2$ continues the simulation of $\mathcal{E}$ as follows: Recall that $\mathcal{E}$ has just sent an encryption request and is still waiting to receive a ciphertext. $A_2$ returns $y^*$ to $\mathcal{E}$ as the ciphertext and continues the simulation of $\mathcal{E}$. Now, encryption requests are handled as in $\mathcal{F}_{\text{rpke}}$. More precisely: When $\mathcal{E}$ sends the $i$-th encryption request for the plaintext $x_i$, with $i \in \{h+1, \ldots, n_{\mathsf{Enc}}\}$, then $A_2$ computes the leakage $\overline{x}_i \leftarrow L(1^\eta, x_i)$ of $x_i$, records the pair $(x_i, \overline{x}_i)$ for later

decryption, encrypts $\overline{x}_i$ under $pk$, and returns the obtained ciphertext to $\mathcal{E}$. When $\mathcal{E}$ sends a decryption request, say for the ciphertext $y$, then, similarly to $\mathcal{F}_{\text{rpke}}$, $A_2$ decrypts $y$ using its decryption oracle $O$; let $\overline{x} := O(y)$. If $\overline{x} = \mathsf{test}$, then $A_2$ sends $x_h$ (recall that $x_h$ is the plaintext in the $h$-th encryption request, i.e., the left part of the challenge output by $A_1$) to $\mathcal{E}$. Otherwise (i.e., $\overline{x} \neq \mathsf{test}$), $A_2$ does the following: If there exists exactly one plaintext $x$ such that the pair $(x, \overline{x})$ has been recorded upon encryption, then $A_2$ returns this $x$ to $\mathcal{E}$. Otherwise, if there exist more than one such $x$, $A_2$ sends an error message to $\mathcal{E}$. Otherwise (i.e., there exists no such $x$), $A_2$ sends $\overline{x}$ to $\mathcal{E}$. When the simulated run stops, then $A_2$ outputs 1 if $\mathcal{E}$ has output 1 on $\mathsf{decision}$, otherwise, $A_2$ outputs 0.

Note that it always holds that $|x_h| = |\overline{x}_h|$ because $L$ is length preserving. Since $\mathcal{E}$ is universally bounded, $A_1$ and $A_2$ are polynomial-time. Hence, $A$ is a valid IND-RCCA adversary against $\Sigma$.

Before we analyze the advantage of $A$ against $\Sigma$, we note that it is easy to see that:

$$\mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}(1^\eta, a) = \left|\Pr\left[\mathrm{Exp}^{\text{ind-rcca-1}}_{A,\Sigma} = 1\right] - \Pr\left[\mathrm{Exp}^{\text{ind-rcca-0}}_{A,\Sigma} = 1\right]\right| \tag{12}$$

where for all $b \in \{0,1\}$:

$$\begin{aligned}\mathrm{Exp}^{\text{ind-rcca-}b}_{A,\Sigma}(1^\eta, a)\colon (pk, sk) \leftarrow \mathsf{gen}(1^\eta); (x_0, x_1, s) \leftarrow A_1^{\mathsf{dec}(sk,\cdot)}(1^\eta, a, pk);\\ y \leftarrow \mathsf{enc}(pk, x_b); b' \leftarrow A_2^{\overline{\mathsf{dec}}(x_0,x_1,sk,\cdot)}(s, y); \textbf{return } b'\end{aligned} \tag{13}$$

where $\overline{\mathsf{dec}}(x_0, x_1, sk, \cdot)$ is defined in Definition 13.

For every $i \in \{1, \ldots, n_{\mathsf{Enc}}\}$, by $A^{(i)} = (A_1^{(i)}, A_2^{(i)})$ we denote the adversary which, instead of choosing $h$ randomly, sets $h$ to $i$ and then behaves exactly as $A$. The proof proceeds similar to the proof of Theorem 7 (Appendix B.2.1): We consider the experiments $\mathrm{Exp}^{\text{ind-rcca-}b}_{A^{(i)},\Sigma}$ for $i = 1, \ldots, n_{\mathsf{Enc}}$. To simplify notation, we define

$$\mathrm{Exp}^0_0 := \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}} \qquad \mathrm{Exp}^b_i := \mathrm{Exp}^{\text{ind-rcca-}b}_{A^{(i)},\Sigma} \qquad \mathrm{Exp}^1_{n_{\mathsf{Enc}}+1} := \mathcal{E} \,|\, \mathcal{P}_{\text{pke}}$$

for all $i \in \{1, \ldots, n_{\mathsf{Enc}}\}$ and $b \in \{0,1\}$. By construction of $A$, it is easy to see that:

$$\begin{aligned}\mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}(1^\eta, a) &\overset{(12)}{=} \left|\Pr\left[\mathrm{Exp}^{\text{ind-rcca-1}}_{\Sigma,A} = 1\right] - \Pr\left[\mathrm{Exp}^{\text{ind-rcca-0}}_{\Sigma,A} = 1\right]\right|\\ &= \frac{1}{n_{\mathsf{Enc}}} \cdot \left|\sum_{i=1}^{n_{\mathsf{Enc}}} \Pr\left[\mathrm{Exp}^1_i = 1\right] - \Pr\left[\mathrm{Exp}^0_i = 1\right]\right| .\end{aligned} \tag{14}$$

Now, as in the proof of Theorem 7, we would like to prove that $\Pr\left[\mathrm{Exp}^0_i = 1\right] = \Pr\left[\mathrm{Exp}^1_{i+1} = 1\right]$ for all $i \in \{0, \ldots, n_{\mathsf{Enc}} - 1\}$ because this would imply $|\Pr[\mathcal{E} \,|\, \mathcal{P}_{\text{pke}} = 1] - \Pr[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}} = 1]| = n_{\mathsf{Enc}} \cdot \mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}$. This however is not possible because the systems differ slightly (see below). Instead, we show that there exists a negligible function $f_B$ such that:

$$f_B(\eta, a) \geq \left|\Pr\left[\mathrm{Exp}^0_i = 1\right] - \Pr\left[\mathrm{Exp}^1_{i+1} = 1\right]\right| \text{ for all } i \in \{0, \ldots, n_{\mathsf{Enc}}\}. \tag{15}$$

Before we prove (15), we show how this implies $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$. It holds that:

$$\begin{aligned}&|\Pr[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}} = 1] - \Pr[\mathcal{E} \,|\, \mathcal{P}_{\text{pke}} = 1]|\\ &\quad= \left|\Pr\left[\mathrm{Exp}^0_0 = 1\right] - \Pr\left[\mathrm{Exp}^1_{n_{\mathsf{Enc}}+1} = 1\right]\right|\\ &\quad= \left|\sum_{i=0}^{n_{\mathsf{Enc}}} \Pr\left[\mathrm{Exp}^0_i = 1\right] - \Pr\left[\mathrm{Exp}^1_{i+1} = 1\right] + \sum_{i=1}^{n_{\mathsf{Enc}}} \Pr\left[\mathrm{Exp}^1_i = 1\right] - \Pr\left[\mathrm{Exp}^0_i = 1\right]\right|\\ &\quad\leq \sum_{i=0}^{n_{\mathsf{Enc}}} \left|\Pr\left[\mathrm{Exp}^0_i = 1\right] - \Pr\left[\mathrm{Exp}^1_{i+1} = 1\right]\right| + \left|\sum_{i=1}^{n_{\mathsf{Enc}}} \Pr\left[\mathrm{Exp}^1_i = 1\right] - \Pr\left[\mathrm{Exp}^0_i = 1\right]\right|\\ &\quad\overset{(14),(15)}{\leq} (n_{\mathsf{Enc}} + 1) \cdot f_B(\eta, a) + n_{\mathsf{Enc}} \cdot \mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}(1^\eta, a) .\end{aligned}$$

By the assumption that $\Sigma$ is IND-RCCA secure, $\mathrm{Adv}^{\text{ind-rcca}}_{A,\Sigma}$ is negligible (as a function in $\eta$ and $a$). Hence, because $n_{\mathsf{Enc}}$ is a polynomial (in $\eta$ and $a$) and $f_B$ is negligible, we conclude that $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}} \equiv \mathcal{E} \,|\, \mathcal{P}_{\text{pke}}$, i.e., $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$.

Finally, we show (15); here we will need that $L$ has high entropy (Definition 8). For every $i \in \{0, \ldots, n_{\mathsf{Enc}}\}$, we define $B_i(1^\eta, a)$ ($B_i$ for short) to be the event that, in a run of $\mathrm{Exp}^0_i(1^\eta, a)$, (at least) one of the following things happens:[28]

1. *Collision of a leakage with the $i$-th plaintext:* $x_i \in \{\overline{x}_{i+1}, \ldots, \overline{x}_{n_{\mathsf{Enc}}}\}$ and $0 < i < n_{\mathsf{Enc}}$.
2. *Collision of a leakage with the $(i+1)$-st plaintext:* $x_{i+1} \in \{\overline{x}_{i+2}, \ldots, \overline{x}_{n_{\mathsf{Enc}}}\}$ and $i < n_{\mathsf{Enc}} - 1$.
3. *Collision of a leakage with the $(i+1)$-st leakage:* $\overline{x}_{i+1} \in \{\overline{x}_{i+2}, \ldots, \overline{x}_{n_{\mathsf{Enc}}}\}$ and $i < n_{\mathsf{Enc}} - 1$.
4. *The ciphertext in a decryption request of $\mathcal{E}$ decrypts to the $i$-th leakage:* $i > 0$ and $\mathcal{E}$ sends a decryption request for some ciphertext $y$ such that $\mathsf{dec}(sk, y) = \overline{x}_i$, where $sk$ is the private key that has been generated in the experiment.

Note that $B_0$ is the event that in a run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{pke}} = \mathrm{Exp}^0_0$ it holds that $x_1 \in \{\overline{x}_2, \ldots, \overline{x}_{n_{\mathsf{Enc}}}\}$ (i.e., the first plaintext collides with some leakage) or $\overline{x}_1 \in \{\overline{x}_2, \ldots, \overline{x}_{n_{\mathsf{Enc}}}\}$ (i.e., the first leakage collides with some other leakage).

We show that the event $B_i$ occurs with negligible probability. More precisely, there exists a negligible function $f_B$ such that:

$$\Pr\left[B_i(1^\eta, a)\right] \leq f_B(\eta, a) \text{ for all } i \in \{0, \ldots, n_{\mathsf{Enc}}\}. \tag{16}$$

It is easy to see that (16) holds: Since $L$ has high entropy, freshly generated leakages do not collide (except with negligible probability) with $x_i$, $x_{i+1}$, and $\overline{x}_{i+1}$. Furthermore, $\mathcal{E}$ does not (except with negligible probability) send a decryption request for a ciphertext $y$ such that $y$ decrypts to $\overline{x}_i$ because $\overline{x}_i$ is a leakage and the view of $\mathcal{E}$ is independent (as a random variable) of $\overline{x}_i$ until $\mathcal{E}$ sends this decryption request. Note that $\overline{x}_i$ is only used in the decryption oracle $O$ because in $\mathrm{Exp}^0_i$ the ciphertext $y^*$ is the encryption of $x_i$ and not the encryption of $\overline{x}_i$. Hence, we find a polynomial $q$ (because $\mathcal{E}$ is universally bounded) such that

$$f_B(\eta, a) := q(\eta + |a|) \cdot \sup\{\Pr\left[\overline{x} \leftarrow L(1^\eta, x), \overline{x}' \leftarrow L(1^\eta, x') : \overline{x} = \overline{x}'\right] \mid x, x' \in D_L(\eta)\} \ ,$$

where $D_L$ is the domain of plaintexts associated with $L$ (and $\Sigma$), satisfies (16). Since $L$ has high entropy (i.e., the supremum in the definition of $f_B$ is negligible) and $q$ is a polynomial (in $\eta$ and $a$), $f_B$ is negligible.

For every $i \in \{0, \ldots, n_{\mathsf{Enc}}\}$, it is now easy to see that every run of $\mathrm{Exp}^0_i$ where $B_i$ does not occur corresponds to a run of $\mathrm{Exp}^1_{i+1}$ such that both runs have the same probability and the same overall output. More formally, we can define an injective mapping from runs of $\mathrm{Exp}^0_i$ where $B_i$ does not occur to runs of $\mathrm{Exp}^1_{i+1}$ such that $\mathcal{E}$ and every call to the encryption and leakage algorithm uses the same randomness in both runs. One can then show that $\mathcal{E}$ has the same view in both runs (the view of $\mathcal{E}$ would only differ if $B_i$ would occur). Furthermore, both runs have the same probability. Hence:

$$\left|\Pr\left[\mathrm{Exp}^0_i = 1\right] - \Pr\left[\mathrm{Exp}^1_{i+1} = 1\right]\right| \leq \Pr\left[B_i(1^\eta, a)\right] \overset{(16)}{\leq} f_B(\eta, a) \ .$$

This shows (15), which, as shown above, implies $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$.

### B.3.2 $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}} \Rightarrow \Sigma$ is IND-RCCA Secure

We now show that $\Sigma$ is IND-RCCA secure if $\mathcal{P}_{\text{pke}} \leq \mathcal{F}_{\text{rpke}}$. The proof is very similar to the corresponding part of the proof of Theorem 7 (Appendix B.2.2). Assuming that $\Sigma$ is *not* IND-RCCA secure, we use a successful adversary $A = (A_1, A_2)$ against $\Sigma$ to construct an environment $\mathcal{E} \in \mathrm{Env}(\mathcal{P}_{\text{pke}})$ that distinguishes between $\mathcal{P}_{\text{pke}}$ and $\mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$ for any simulator $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}_{\text{pke}}}(\mathcal{F}_{\text{rpke}})$. The environment $\mathcal{E}$ is defined as in Appendix B.2.2 except that whenever $A_2$ asks its decryption oracle to decrypt a ciphertext $y$ then $\mathcal{E}$ does the following: It

[28] Formally, $B_i(1^\eta, a)$ is the set of all runs of $\mathrm{Exp}^0_i(1^\eta, a)$ with the mentioned property.

asks $\mathcal{P}_{\text{pke}}$ to decrypt $y$; let $x$ be the returned plaintext. If $x = x_0$ or $x = x_1$ (where $x_0, x_1$ are the challenge plaintexts that have been output by $A_1$), then $\mathcal{E}$ continues the simulation of $A_2$ as if the decryption oracle returned $\mathsf{test}$. Otherwise, $\mathcal{E}$ continues the simulation as if the oracle returned $x$.

Analogously to the proof in Appendix B.2.2, we can show that in the real world (i.e., in runs of $\mathcal{E} \,|\, \mathcal{P}_{\text{pke}}$), the simulation of $A$ is exactly like in the IND-RCCA experiment and we obtain:

$$\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right] = \frac{1}{2} \Pr\left[\text{Exp}^{\text{ind-rcca-1}}_{A,\Sigma}(1^\eta, a) = 1\right] + \frac{1}{2} - \frac{1}{2} \Pr\left[\text{Exp}^{\text{ind-rcca-0}}_{A,\Sigma}(1^\eta, a) = 1\right] \quad . \tag{17}$$

(See (13) in Appendix B.3.1 for the definition of $\text{Exp}^{\text{ind-rcca-}b}_{A,\Sigma}$ for $b \in \{0, 1\}$.)

As for the ideal world, again analogously to the proof in Appendix B.2.2, one can show that in runs of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$, $\mathcal{E}$ outputs 1 with probability exactly 1/2:

$$\Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}})(1^\eta, a) = 1\right] = \frac{1}{2} \quad . \tag{18}$$

We conclude that:

$$\begin{aligned}
&\left|\Pr\left[(\mathcal{E} \,|\, \mathcal{P}_{\text{pke}})(1^\eta, a) = 1\right] - \Pr\left[(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}})(1^\eta, a) = 1\right]\right| \\
&\quad \overset{(17),(18)}{=} \frac{1}{2} \cdot \left|\Pr\left[\text{Exp}^{\text{ind-rcca-1}}_{A,\Sigma}(1^\eta, a) = 1\right] - \Pr\left[\text{Exp}^{\text{ind-rcca-0}}_{A,\Sigma}(1^\eta, a) = 1\right]\right| \\
&\quad \overset{(12)}{=} \frac{1}{2} \cdot \text{Adv}^{\text{ind-rcca}}_{A,\Sigma}(1^\eta, a) \quad ,
\end{aligned}$$

which, by assumption, is non-negligible (as a function in $\eta$ and $a$). Hence, $\mathcal{E} \,|\, \mathcal{P}_{\text{pke}} \not\equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{rpke}}$ and we conclude that $\mathcal{P}_{\text{pke}} \not\leq \mathcal{F}_{\text{rpke}}$.

This concludes the proof of Theorem 8. □