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

Ralf Küsters and Max Tuengerthal

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

Abstract. Composition theorems in simulation-based approaches allow to build complex protocols from sub-protocols in a modular way. 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 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 are shown to be unsuitable. Our work is based on a recently proposed, rigorous model for simulation-based security by Küsters, called 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.

1 Introduction

In the simulation-based security approach (see, e.g., [3, 5, 7, 15, 17, 21, 22, 23, 28]) the security of protocols and functionalities is defined w.r.t. *ideal protocols/functionalities*. Composition theorems proved within this approach guarantee secure concurrent composition of a protocol with arbitrary other protocols, including copies of itself. As a result, complex protocols can be built and analyzed in a modular fashion. 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 copies of a protocol have disjoint state. In particular, the random coins used in different copies 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 copy of the protocol. This is completely impractical and motivates to generalize composition theorems to so-called joint state theorems, where different copies 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.

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 (Section 5, 6 and 7), ii) a joint state theorem which is more general than other formulations and corrects flaws in these formulations (Section 3), and iii) based on this theorem, joint state realizations and theorems for (replayable) public-key encryption and digital signatures (Section 5.3, 6.3 and 7.3). Unfortunately, all other joint state theorems claimed in the literature for such

* Part of this work was carried out while the authors were at ETH Zurich, partially supported by SNF grant 200021-116596/1.

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 simulation-based security that we use, the IITM model [23]. 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 of the IITM model. More precisely, our contributions are as follows.

We formulate three functionalities: public-key encryption, replayable public-key encryption, and digital signatures. Our formulation of replayable public-key encryption is meant to model in a simulation-based setting the recently proposed notion of replayable CCA-security (RCCA security) [11]. This relaxation of CCA-security permits anyone to generate new ciphertexts that decrypt to the same plaintext as a given ciphertext. As argued in [11], RCCA-security suffices for most existing applications of CCA-security. In our formulations of the above mentioned functionalities ciphertexts and signatures are determined by local computations, and hence, as needed in many applications, a priori do not reveal signed messages or ciphertexts. In other formulations, e.g., those in [14, 6, 1, 11, 20], signatures and ciphertexts are determined by interaction with the adversary. This has 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 CCA-secure (RCCA-secure), for a statically corruptible decryptor and adaptively corruptible encryptors. We also prove equivalence between EU-CMA security of digital signatures schemes and our digital signature functionality, with adaptive corruption of both signer and verifiers.

In the spirit of Canetti and Rabin [14], we state a general joint state theorem. However, in contrast to Canetti's original UC model as employed in [14] and his new UC model [7], 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 one in [14, 7] (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 cases they are satisfied, the theorem does not necessarily hold true.

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. As 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 [5]. They propose a 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 joint state theorem, as mentioned, has several problems and limitations, which are mainly due to the kind of ITMs used (see Section 3).

In [7], 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 [11]. As for joint state, Canetti only points to [14], with the limitations and problems inherited from this work. Moreover, as further discussed in Section 5.4, 6.4 and 7.4, the joint state theorems claimed for the public-key encryption and digital signature functionalities in [7] are flawed. The same is true for the work by Canetti and Herzog in [9], 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, 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 (non-trivial) goals of this work.

Within simulation-based models, realizations of protocols with joint state across sessions were, for example, proposed in [10, 25, 12, 27].

Backes, Pfitzmann, and Waidner [2] consider implementations of digital signatures and public-key encryptions within their so-called cryptographic library, which requires somewhat non-standard cryptographic constructions and does not provide the user with the actual signatures and ciphertexts. Within their proofs they use a public-key encryption functionality proposed by Pfitzmann and Waidner [28]. Joint state theorems are not considered in these works. In fact, joint state theorems talk about copies of protocols, but the results of Backes, Pfitzmann, and Waidner are based on a version of the PIOA model which does not explicitly handle copies of protocols [28, 3].

Functionalities for digital signatures and public-key encryption with *non-local* computation, i.e. where signatures and ciphertexts are provided by the adversary, have been proposed in [6, 1, 5, 20]; however, joint state theorems have not been proven in these papers.

In [8], Canetti et al. study simulation-based security 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. The global setup can be considered as joint state. But it is joint state among *all* entities, unlike the joint state settings considered here, where the joint state is only shared within copies of functionalities. Therefore the results proved in [8] do not apply to the problem studied in this paper.

Structure of the paper. In the following section, we 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 general conventions for formulating IITMs are presented. In the following three sections we then present our formulations of functionalities for digital signatures, public-key encryption, and replayable public-key encryption, respectively, including proofs of equivalence to game-based security notions, joint state realizations, and comparison with other formulations.

2 The IITM Model

In this section, we briefly recall the IITM model for simulation-based security (see [23] for details). Based on a relatively simple, but very expressive general computational model in which IITMs and systems of IITMs are defined, simulation-based security notions are formalized, and a general composition theorem can be proved in the IITM model.

2.1 The General Computational Model

We first define IITMs and then systems of IITMs. We note that this general computation model is also useful in contexts other than simulation-based security [16].

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

Computation of IITMs. An IITM is activated with one message on one of its input tapes and it writes at most one output message per activation on one of the output tapes. The runtime of the IITM *per activation* is polynomially bounded in the security parameter, the current input, and the size of the current configuration. This allows the IITM to "scan" the complete incoming message and its complete current configuration, and to react to all incoming messages, no matter how often the IITM is activated. In

particular, an IITM can not be exhausted (therefore the name *inexhaustible* interactive Turing machine). The length of the configuration and the length of the overall output an IITM can produce is polynomially bounded in the security parameter and the length of the overall input received on *enriching* input tapes so far, i.e., writing messages on these tapes increases the resources (runtime, potential size of the configuration, and potential length of the output) of the IITM. An IITM runs in one of two modes, CheckAddress (deterministic computation) and Compute (probabilistic computation). The CheckAddress mode will be used to address different copies of IITMs in a system of IITMs (see below). This is a very generic addressing mechanism: Details of how an IITM is addressed are not fixed up-front, but left to the specification of the IITM itself.

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

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

where M ranges of the set of IITMs. No two input tapes occurring in IITMs in $\mathcal{S}$ are allowed to have the same names, i.e. every input tape name belongs to exactly one IITM in the system. The system $\mathcal{S}_1 \parallel \mathcal{S}_2$ is the *concurrent composition* of the two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ and $!\mathcal{S}$ is the *concurrent composition of an unbounded number of copies* of (machines in) the system $\mathcal{S}$. Each machine M that occurs in a subexpression $!\mathcal{S}'$ of $\mathcal{S}$ is said to be in the *scope of a bang*. Below, we define the way a system runs. From that it will be clear that every system $\mathcal{S}$ is equivalent to a system of the shape $M_1 \parallel \cdots \parallel M_k \parallel !M'_1 \parallel \cdots \parallel !M'_{k'}$ where M_i for all $i \in \{1, \ldots, k\}$ and M'_j for all $j \in \{1, \ldots, k'\}$ are IITMs.

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

We will only consider so-called well-formed systems [23], which satisfy a simple syntactic condition that guarantees polynomial runtime of systems and suffices for applications since it allows to always provide sufficient resources to IITMs via enriching tapes.

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

Theorem 1 ([23]). *(informal) Well-formed systems run in polynomial time.*

We write $\mathrm{Prob}[\mathcal{S}(1^\eta, a) \rightsquigarrow 1]$ to denote the probability that the overall output (the message written on tape decision, if any) of a run of a (well-formed) system $\mathcal{S}$ with security parameter η and auxiliary input a for the master IITM is 1. Two well-formed systems $\mathcal{P}$ and $\mathcal{Q}$ are called *equivalent* or *indistinguishable* ($\mathcal{P} \equiv \mathcal{Q}$) iff the function $f(1^\eta, a) = |\mathrm{Prob}[\mathcal{P}(1^\eta, a) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{Q}(1^\eta, a) \rightsquigarrow 1]|$ is negligible, i.e., for all polynomials p and q there exists $\eta_0 \in \mathbb{N}$ such that for all $\eta > \eta_0$ and all bit strings $a \in \{0,1\}^*$ with length $|a| \leq q(\eta)$ we have that $f(1^\eta, a) \leq \frac{1}{p(\eta)}$. Analogously, two well-formed systems $\mathcal{P}$ and $\mathcal{Q}$ are called *equivalent* or *indistinguishable without auxiliary input* ($\mathcal{P} \equiv^{noaux} \mathcal{Q}$) iff the function $f(1^\eta) = |\mathrm{Prob}[\mathcal{P}(1^\eta, \varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{Q}(1^\eta, \varepsilon) \rightsquigarrow 1]|$ is negligible, i.e., for all polynomials p there exists $\eta_0 \in \mathbb{N}$ such that for all $\eta > \eta_0$ we have that $f(1^\eta) \leq \frac{1}{p(\eta)}$. Clearly, $\mathcal{P} \equiv \mathcal{Q}$ implies $\mathcal{P} \equiv^{noaux} \mathcal{Q}$.

Given an IITM M, we will often use its *identifier version* $\underline{M}$ to be able to address multiple copies of M (see [23, 24] for a detailed definition). The identifier version $\underline{M}$ of M is an IITM which simulates M within a "wrapper". The wrapper requires that all messages received have to be prefixed by a particular

identifier, e.g., a session ID (SID) or party ID (PID); other messages will be rejected in the CheckAddress mode. Before giving a message to M, the wrapper strips off the identifier. Messages sent out by M are prefixed with this identifier by the wrapper. The identifier that $\underline{M}$ will use is the one with which $\underline{M}$ was first activated. We often refer to $\underline{M}$ by *session version* or *party version* of M if the identifier is meant to be a SID or PID, respectively. For example, if M specifies an ideal functionality, then $!\underline{M}$ denotes a system which can have an unbounded number of copies of $\underline{M}$, all with different SIDs. If M specifies the actions performed by a party in a multi-party protocol, then $!\underline{M}$ specifies the multi-party protocol where every copy of $\underline{M}$ has a different PID. Note that one can consider an identifier version $\underline{\underline{M}}$ of $\underline{M}$, which effectively means that the identifier is a tuple of two identifiers. Of course, this can be iterated further. Given a system $\mathcal{S}$, its *identifier version* $\underline{\mathcal{S}}$ is obtained by replacing all IITMs by their identifier version. For example, with $\mathcal{S} = M_1 \parallel \cdots \parallel M_k \parallel !M'_1 \parallel \cdots \parallel !M'_{k'}$ as above, we obtain $\underline{\mathcal{S}} = \underline{M_1} \parallel \ldots \parallel \underline{M_k} \parallel !\underline{M'_1} \parallel \ldots \parallel !\underline{M'_{k'}}$. Note that for all i, all copies of $\underline{M'_i}$ in a run of $\underline{\mathcal{S}}$ will have different identifiers.

2.2 Notions of Simulation-Based Security

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

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

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

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

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

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

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

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

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

messages between the network and I/O interface. A system $\mathcal{A}$ is called an *adversarial system* if it is well-formed and $\mathcal{A}$ has no tape named start or decision. (All external tapes of $\mathcal{A}$ may be enriching.) The set of all adversarial systems is denoted by $\mathbf{A}$ or $\mathbf{S}$. A system $\mathcal{E}$ is called an *environmental system* if it is well-formed, tape start may be enriching and all other external tapes are consuming. The set of all environmental systems is denoted by $\mathbf{E}$.

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

Definition 1 (Strong Simulatability (SS); [23]).
Let $\mathcal{P}$ and $\mathcal{F}$ be I/O compatible protocol systems, the real and the ideal protocol, respectively. Then, $\mathcal{P}$ SS-realizes *$\mathcal{F}$ ($\mathcal{P} \leq^{SS} \mathcal{F}$) iff there exists an adversarial system $\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}}_{\mathbf{S}}(\mathcal{F})$ such that for all environmental systems $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P})$ it holds that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$. Analogously, $\mathcal{P}$* SS-realizes *$\mathcal{F}$* without auxiliary input *($\mathcal{P} \leq^{SS\text{-}noaux} \mathcal{F}$) if in the above it holds $\mathcal{E} \,|\, \mathcal{P} \equiv^{noaux} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$.*

In a similar way, other equivalent security notions such as black-box simulatability and (dummy) UC can be defined [23]. We emphasize that in these and the above definitions, no specific addressing or corruption mechanism is fixed. This can be defined in a rigorous, convenient, and flexible way as part of the real/ideal protocol specifications.

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

Note that $\mathcal{P} \leq^{SS} \mathcal{F}$ implies $\mathcal{P} \leq^{SS\text{-}noaux} \mathcal{F}$. As above, simulatability without auxiliary is transitive and reflexive. In fact, all results for strong simulatability used in this paper hold both for strong simulatability with and without auxiliary input. We therefore will not distinguish between the two variants.

2.3 Composition Theorems

We restate the composition theorems from [23]. The first composition theorem describes concurrent composition of a fixed number of protocol systems while the second one the composition of an unbounded number of copies of a protocol system.

Theorem 2 ([23]). *Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ and $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be protocol systems such that $\mathcal{P}_1 \,|\, \ldots \,|\, \mathcal{P}_k$ and $\mathcal{F}_1 \,|\, \ldots \,|\, \mathcal{F}_k$ are well-formed and for every $j \in \{1, \ldots, k\}$ the following conditions are satisfied:*

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

Then,

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

Theorem 3 ([23]). *Let $\mathcal{P}$ and $\mathcal{F}$ be protocol systems such that $\mathcal{P}$ and $\mathcal{F}$ are I/O compatible and $\mathcal{P} \leq^{SS} \mathcal{F}$. Then,*

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

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

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

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

Iterated application of Theorem 2 and 3 allows to construct very complex systems, e.g., protocols using several levels of an unbounded number of copies of sub-protocols. Unlike the UC model, super-protocols can directly access sub-protocols across levels, yielding simpler and possibly more efficient implementations. In the UC model, a protocol has to completely shield its sub-protocol from the environment, and hence, from super-protocols on higher levels. In [8], the composition operator therefore had to be extended to allow access to a globally available functionality. No such extension would have been necessary in the IITM model to obtain the results proved in this work. We also note that Theorem 3 cannot only be interpreted as yielding multi session realizations from single session realizations, but also providing multi party realizations from single party realizations (when $\underline{\mathcal{P}}$ and $\underline{\mathcal{F}}$ are considered as multi party versions).

3 The Joint State Theorem

In this section, we present our general joint state theorem along the lines of the theorem by Canetti and Rabin [14]. However, as we will see, in the IITM model, the theorem can be stated in a much more elegant and general way, and it follows immediately from the composition theorem. We also point out problems of the joint state theorem by Canetti and Rabin.

Let us first recall the motivation for joint state from the introduction, using the notation from the IITM model. Assume that $\mathcal{F}$ is an ideal protocol (formally, a protocol system) that describes an ideal functionality used by multiple parties in one session. For example, $\mathcal{F} = !\underline{\mathcal{F}'}$ could be a multi-party version of a single party functionality $\mathcal{F}'$, e.g., a public-key encryption or signature box. Assume that the protocol $\mathcal{P}$ realizes $\mathcal{F}$, i.e., $\mathcal{P} \leq^{SS} \mathcal{F}$, and that $\mathcal{P}$ is of the form $!\underline{\mathcal{P}'}$, where $\underline{\mathcal{P}'}$ is the party version of some $\mathcal{P}'$, i.e., each copy of $\underline{\mathcal{P}'}$ in $!\underline{\mathcal{P}'}$ is "owned" by one party. Now, by Theorem 3, we have that $!\underline{\underline{\mathcal{P}'}} = !\underline{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$ (note that $!!Q$ is equivalent to $!Q$), i.e., the multi-session version of $\mathcal{P}$ realizes the multi-session version of $\mathcal{F}$. Unfortunately, in the realization $!\underline{\mathcal{P}}$ of $!\underline{\mathcal{F}}$, one new copy of $\underline{\mathcal{P}'}$ is created per party per session. This is impractical. For example, if $\mathcal{P}/\mathcal{F}$ are functionalities for public-key encryption, then in $!\underline{\underline{\mathcal{P}'}}$ every party has to create a new key pair for every session.

To allow for more efficient realizations, Canetti and Rabin [14] introduce a new composition operation, called *universal composition with joint state (JUC)*, which takes two protocols as arguments: First, a protocol $\mathcal{Q}$, which uses multiple sessions with multiple parties of some ideal functionality $\mathcal{F}$, i.e., $\mathcal{Q}$ works in an $\mathcal{F}$-hybrid model, and second, a realization $\widehat{\mathcal{P}}$ of $\widehat{\mathcal{F}}$, where $\widehat{\mathcal{F}}$ is a single machine which simulates the multi-session multi-party version of $\mathcal{F}$. In the IITM model, instead of $\widehat{\mathcal{F}}$, one could simply write $!\underline{\mathcal{F}}$, and require that $\widehat{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$. However, this cannot directly be formulated in the UC model. In the resulting JUC composed protocol $\mathcal{Q}^{[\widehat{\mathcal{P}}]}$, calls from $\mathcal{Q}$ to $\mathcal{F}$ are translated to calls to $\widehat{\mathcal{P}}$ where only one copy of $\widehat{\mathcal{P}}$ is created per party and this copy handles all sessions of this party, i.e., $\widehat{\mathcal{P}}$ may make use of joint state. The general joint state theorem in [14] then states that if $\widehat{\mathcal{P}}$ realizes $\widehat{\mathcal{F}}$, then $\mathcal{Q}^{[\widehat{\mathcal{P}}]}$ realizes $\mathcal{Q}$ in the $\mathcal{F}$-hybrid model.

An analog of this theorem can elegantly and rigorously be stated in the IITM model as follows:

Theorem 4. *Let $\mathcal{Q}, \widehat{\mathcal{P}}, \mathcal{F}$ be protocol systems such that $\mathcal{Q} \,|\, \widehat{\mathcal{P}}$ and $\mathcal{Q} \,|\, !\underline{\mathcal{F}}$ are well-formed, $\mathcal{Q}$ is environmentally connectible for $\widehat{\mathcal{P}}$ and $!\underline{\mathcal{F}}$, and $\widehat{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$. Then, $\mathcal{Q} \,|\, \widehat{\mathcal{P}} \leq^{SS} \mathcal{Q} \,|\, !\underline{\mathcal{F}}$.*

Proof. By Theorem 2 and the reflexivity of $\leq^{SS}$, we conclude from $\widehat{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$ that $\mathcal{Q} \,|\, \widehat{\mathcal{P}} \leq^{SS} \mathcal{Q} \,|\, !\underline{\mathcal{F}}$.

The fact that Theorem 4 immediately follows from Theorem 2 shows that in the IITM model, there is no need for an explicit joint state theorem. The reason it is needed in the UC model lies in the restricted expressivity it provides in certain respects: First, one has to define an ITM $\widehat{\mathcal{F}}$, and cannot simply write $!\underline{\mathcal{F}}$, as multi-party, multi-session versions only exist as part of a hybrid model. In particular, $\widehat{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$ cannot be stated directly. Second, the JUC operator has to be defined explicitly since it cannot be directly stated that only one instance of $\widehat{\mathcal{P}}$ is invoked by $\mathcal{Q}$; in the IITM model we can simply write $\mathcal{Q} \,|\, \widehat{\mathcal{P}}$. Also, a composition theorem corresponding to Theorem 2, which is used to show that $\widehat{\mathcal{P}}$ can be replaced by $!\underline{\mathcal{F}}$, is not directly available in the UC model, only a composition theorem corresponding to Corollary 1. (To obtain a theorem similar to Theorem 2, in the UC model one has to make sure that only one instance is invoked by $\mathcal{Q}$.) Finally, due to the addressing mechanism employed in the UC model, redirection of messages have to be made explicit. While all of this makes it necessary to have an explicitly stated joint state theorem in the UC model, due to the kind of ITMs employed in the UC model, there are also problems with the joint state theorem itself (see below).

We note that despite the trivial proof of Theorem 4 in the IITM model (given the composition theorem), the statement that Theorem 4 makes is stronger than that of the joint state theorem in the UC model [14, 7]. Inherited from our composition theorems, and unlike the theorem in the UC model, Theorem 4 does not require that $\mathcal{Q}$ completely shields the sub-protocol from the environment, and hence, from super-protocols on higher levels. Granting access to the sub-protocol across protocol levels can lead to simpler systems and more efficient implementations, for example in case of global setups [8].

Limitations and problems of the joint state theorem in the UC model. The ITMs used in the UC model, unlike IITMs, cannot block useless messages without consuming resources and their *overall* runtime is bounded by a polynomial in the security parameter and, in the UC model as presented in [7], the overall

length of the input on the I/O interface. A consequence of this is that in general a single ITM cannot simulate a concurrent composition of a fixed finite or an unbounded number of (copies of) ITMs: If one of the ITMs, say M', in the concurrent composition halts, then the ITM, say M, simulating the composition should block messages to M'. Otherwise, M has to process messages which are sent to M'. This consumes resources of M and can lead to the exhaustion of M. Hence, M is not capable of simulating the other machines in the composition anymore. In the original model of the UC model [5], this kind of exhaustion can happen no matter whether messages are sent on the network or the I/O interface. In the new UC model [7], the exhaustion can happen when messages are sent on the network interface.

Now, this causes problems in the joint state theorem of the UC model: Although the ITM $\widehat{\mathcal{F}}$ in this joint state theorem is intended to simulate the multi-party, multi-session version of $\mathcal{F}$, for the reason explained above, it cannot do this in general; it can only simulate some approximated version. The same is true for $\widehat{\mathcal{P}}$. 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 $\widehat{\mathcal{P}}$ that realizes $\widehat{\mathcal{F}}$, 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 will first illustrate the problem of realizing $\widehat{\mathcal{F}}$ in the original UC model, i.e., the one presented in [5], on which the work in [14] is based. We then explain the corresponding problem for the new version of the UC model [7].

The ITM $\widehat{\mathcal{F}}$ is intended to simulate the multi-party, multi-session version of $\mathcal{F}$, e.g., a digital signature functionality. The realization $\widehat{\mathcal{P}}$ 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, $\widehat{\mathcal{F}}$ 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 $\widehat{\mathcal{P}}$, because these requests are handled by other ITMs, which are not exhausted. Consequently, an environment can easily distinguish between the ideal ($\widehat{\mathcal{F}}$) and real world ($\widehat{\mathcal{P}}$). 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 $\widehat{\mathcal{F}}$ in the original UC model.

In the new version of the UC model as presented in [7], the problem of realizing $\widehat{\mathcal{F}}$ is similar. However, ITMs cannot be exhausted (forced to stop) via communication over the I/O interface. Nevertheless, exhaustion is possible via the network interface. Assume that $\widehat{\mathcal{P}}$ tries to realize $\widehat{\mathcal{F}}$ in an $\mathcal{F}$-hybrid model, where for every party one instance of $\widehat{\mathcal{P}}$ and $\mathcal{F}$ is generated, if any.[1] The environment (via a dummy adversary) can access any copy of $\mathcal{F}$ in the $\mathcal{F}$-hybrid model directly via the network interface. In this way, the environment can send many messages to the copy of $\mathcal{F}$, and hence, exhaust this copy, i.e., force it to stop, after some time. (Recall that an ITM cannot prevent being exhausted since it cannot block messages without using resources.) 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 has to know when a copy of $\mathcal{F}$ would stop in the hybrid model, because it then must not forward messages addressed to this copy of $\mathcal{F}$ to $\widehat{\mathcal{F}}$. Otherwise, $\widehat{\mathcal{F}}$ 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 $\mathcal{F}$ in the $\mathcal{F}$-hybrid world (via $\widehat{\mathcal{P}}$ 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 $\widehat{\mathcal{F}}$ is not. However, in general $\mathcal{S}$ does not necessarily know when an instance in the hybrid model is exhausted, e.g., because it does not know how much resources have been provided/used by the functionalities upon receiving input on the I/O interface, to which $\mathcal{S}$ does not have access. Hence, in this case $\mathcal{S}$ always has to forward messages, because the functionality might still have enough resources to react. But this then leads to the exhaustion of $\widehat{\mathcal{F}}$, 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

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

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 the messages as long as it gets the expected message, and only stops if it runs out of runtime. Different interpretations of how the runtime is defined or ill-defined functionalities can then lead to the mentioned problems. Even if there is a realization of $\widehat{\mathcal{F}}$ that would work, proving this can become quite tricky because of the described exhaustion problem and its consequences.

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 $\widehat{\mathcal{P}}$ realizes $\widehat{\mathcal{F}}$ within the $\mathcal{F}$-hybrid model, where as usual, at most one copy of $\widehat{\mathcal{P}}$ and $\mathcal{F}$ per party is created. The following arguments apply to both the original UC model [5] and the new version [7]. According to the joint state theorem in the UC model, we should have that $\mathcal{Q}^{[\widehat{\mathcal{P}}]}$ (real world) realizes $\mathcal{Q}$ in the $\mathcal{F}$-hybrid model (ideal world), where as mentioned, we assume $\widehat{\mathcal{P}}$ to work in the $\mathcal{F}$-hybrid model as well. However, the following problems occur: An environment can directly access (via a dummy adversary) a copy of $\mathcal{F}$ in the real world. By sending many messages to this copy, this copy will be exhausted. This copy of $\mathcal{F}$, call it $\mathcal{F}[pid]$, which together with a copy of $\widehat{\mathcal{P}}$ handles all sessions of a party pid, corresponds to several copies $\mathcal{F}[pid, sid]$ of $\mathcal{F}$, for SIDs sid, in the ideal world. Hence, once $\mathcal{F}[pid]$ in the real world is exhausted, the simulator also has to exhaust all its corresponding copies $\mathcal{F}[pid, sid]$ in the ideal world for every sid, because otherwise an environment could easily distinguish the two worlds. (While $\mathcal{F}[pid]$ cannot respond, some of the copies $\mathcal{F}[pid, sid]$ still can.) Consequently, for the simulation to work, $\mathcal{F}$ 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 can be fixed by assuming this feature for functionalities. A more serious problem is that the simulator might not know whether $\mathcal{F}[pid]$ in the real model is exhausted, 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 $\widehat{\mathcal{F}}$, 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 [5] there is yet another, but smaller problem with the joint state theorem. Since in the original UC model the number of copies of $\mathcal{F}$ that $\widehat{\mathcal{F}}$ can simulate is bounded by a polynomial in the security parameter, this number typically also has to be bounded in the realization $\widehat{\mathcal{P}}$. However, now the environment can instruct $\mathcal{Q}$ to generate many copies of $\mathcal{F}$ for one party. In the real world, after some time no new copies of $\mathcal{F}$ for this party can be generated because $\widehat{\mathcal{P}}$ 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 $\mathcal{Q}$ is big enough such that the environment can generate, through $\mathcal{Q}$, more copies than $\widehat{\mathcal{P}}$ can produce. So, this problem can easily be fixed by assuming that the runtime of $\mathcal{Q}$ is bounded appropriately. Conversely, given $\mathcal{Q}$, the runtime of $\widehat{\mathcal{P}}$ 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 more rigorous and elegant framework. Coming up with such a framework and applying it, is one of the main goals of our work.

Applying the Joint State Theorem. Theorem 4, just like the joint state theorem in the UC model, does not by itself yield practical realizations, as it does not answer the question of how a practical realization $\widehat{\mathcal{P}}$ can be found. A desirable instantiation of $\widehat{\mathcal{P}}$ would be of the form $!\mathcal{P}_{\text{js}} \,|\, \mathcal{F}$ where $!\mathcal{P}_{\text{js}}$ is a very basic 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

$$!\mathcal{P}_{\text{js}} \,|\, \mathcal{F} \leq^{SS} !\underline{\mathcal{F}} \ .^{2} \tag{1}$$

[2] Strictly speaking, one has to rename the network tapes of $\mathcal{F}$ on the left-hand side, to ensure both sides to be I/O compatible.

Note that with $\mathcal{P} \leq^{SS} \mathcal{F}$, the composition theorems together with the transitivity of $\leq^{SS}$ imply that $!\mathcal{P}_{\text{js}} \,|\, \mathcal{P} \leq^{SS} !\underline{\mathcal{F}}$. 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., $\mathcal{P}' \leq^{SS} \mathcal{F}'$, then $!\mathcal{P}_{\text{js}} \,|\, !\underline{\mathcal{P}'} \leq^{SS} !\mathcal{P}_{\text{js}} \,|\, !\underline{\mathcal{F}'} \leq^{SS} !\underline{\mathcal{F}} = !\underline{\underline{\mathcal{F}'}}$, 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 $\mathcal{F}'$.

The seamless treatment of joint state in the IITM model allows for iterative applications of the joint state theorem. Consider a protocol $\mathcal{Q}$, e.g., a key exchange protocol, that uses $\mathcal{F}$, e.g., the multi-party version $\mathcal{F} = !\underline{\mathcal{F}'}$ of a public-key encryption box $\mathcal{F}'$ for one party (see above), in multiple sessions for multiple parties. In short, we consider the system $\mathcal{Q} \,|\, !\underline{\mathcal{F}}$. Furthermore, assume that multiple sessions of $\mathcal{Q}$ are used within a more complex protocol, e.g., a protocol for establishing secure channels. Such a protocol uses the system $!\underline{(\mathcal{Q} \,|\, !\underline{\mathcal{F}})} = !\underline{\mathcal{Q}} \,|\, !\underline{\underline{\mathcal{F}}}$. 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: $!\underline{\mathcal{Q}} \,|\, !\underline{\underline{\mathcal{F}}} = !\underline{(\mathcal{Q} \,|\, !\underline{\mathcal{F}})} \geq^{SS} !\underline{(\mathcal{Q} \,|\, (\,!\mathcal{P}_{\text{js}} \,|\, \mathcal{F}))} = !\underline{\mathcal{Q}} \,|\, !\underline{\mathcal{P}_{\text{js}}} \,|\, !\underline{\mathcal{F}} \geq^{SS} !\underline{\mathcal{Q}} \,|\, !\underline{\mathcal{P}_{\text{js}}} \,|\, !\mathcal{P}_{\text{js}} \,|\, \mathcal{F}$, i.e., $!\underline{\mathcal{P}_{\text{js}}} \,|\, !\mathcal{P}_{\text{js}} \,|\, \mathcal{F}$ is the joint state realization of $!\underline{\underline{\mathcal{F}}}$. Note that in this realization only the single instance $\mathcal{F}$ is used for all parties.

4 Notational and Conceptual Conventions for IITMs

In this section we present general conventions for formulating IITMs. These will be used in the subsequent sections.

4.1 Describing IITMs

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

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

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

We use the following convention for names of tapes. Let A be a name associated with an IITM M (or functionality). The names of tapes of this IITM will have a special shape, namely $\text{io}(A, B)$, $\text{io}(B, A)$, $\text{net}(A, B)$, and $\text{net}(B, A)$ where B is another name (that is associated with another IITM or some entity, such as the environment or the adversary/simulator). The pair (A, B) represents the direction of the tape and the prefix the type, e.g. $\text{io}(A, B)$ is an I/O output tape of M and $\text{net}(B, A)$ is a network input tape of M.

In the description of M, we abbreviate "output m on $\text{io}(A, B)$" by "send m to B" and "receive m on $\text{io}(T, M)$" by "receive m from T". Similarly for network tapes.

4.2 Running External Code

Sometimes, an IITM M obtains the description of an algorithm A as input and has to execute it. We write $y \leftarrow \text{sim}_n \, A(x)$ to say that the IITM simulates algorithm A on input x for n steps. The random coins that might be used by A are chosen by M. The variable y is set to the output of A if A terminates

after at most n steps. Otherwise, y is set to the error symbol $\perp$. If we want to enforce M to simulate A in a deterministic way we write $y \leftarrow \text{sim-det}_n\, A(x)$. If A uses random coins, M can simply use the zero bit string. If several transitions are possible in one step, M uses the first one in the description of A (or the smallest one in a lexicographical order).

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

One could generalize the above to algorithms that keep state. However, this is not needed for our purposes.

4.3 Processing Arbitrary Many Messages of Arbitrary Length

We note that protocol systems can process and forward arbitrary many messages of arbitrary length received via the I/O interface (and hence, enriching tapes) because of our definition of polynomial time (see Section 2). In particular, our functionalities for encryption and signing can be used to encrypt/sign an unbounded number of messages, each of arbitrary length.

Since the network interface of protocol systems uses consuming tapes it is not a priori possible to process arbitrary many messages of arbitrary length coming from the network interface. However, this is no loss of expressivity. The following solution is always possible: A functionality can be defined in such a way that before it accepts (long) input from the network interface, it expects to receive input (resources) from the environment on the I/O interface, e.g., on a designated "resource tape". Note that the environment controls part of the I/O interface, including the resource tape, and the complete network interface.[3] Hence, right before the environment wants to send long messages via the network interface, it can simply provide enough resources via the I/O interface. The environment does not have to be specified explicitly, since in the security notions one quantifies over all environments.

This generic mechanism of providing resources via the I/O interface can always be employed to guarantee enough resources. In complex systems these resources can travel from super-protocols to the sub-protocol which needs these resources. For example, we employ this mechanism for dealing with corruption (see below), where arbitrary many and arbitrary long messages have to be forwarded from the network interface to the I/O interface.

An alternative to declaring network interfaces to consist of consuming tapes, is to use enriching tapes. However, this leads to more involved security notions and more complex restrictions for composing protocols (see, e.g., [22, 23]). Whether or not to use this alternative is a matter of taste.

4.4 Corruption

In the UC model [5, 7], corruption of a functionality, such as encryption or digital signatures, is typically not specified precisely. However, this is important for the joint state theorems (see Section 3). Recall that in the IITM model corruption is not hard-wired into the model but can be specified in a rigorous and flexible way as part of the protocols/functionalities.

One possible way of specifying corruption is depicted in Figure 1. This "macro" can be used in the specification of functionalities. It models that if an IITM is corrupted, i.e., receives a corrupt message from the adversary/simulator on the network interface, it will expose the information *corrMsg* to the adversary and from now on forwards all messages between the network and I/O interface, which is possible by the mechanism discussed in Section 4.3: Before a message is forwarded from the network interface to the I/O interface (the user), the environment is supposed to provide resources for this action, i.e., send a message of the form (Res, r) via the I/O interface.

In (a) and (b) variable *initialized* is used to make sure that the functionality in which the macro is used has already been activated. This is important for joint state realizations. The environment can ask whether the IITM is corrupted; security notions otherwise would not make sense: a simulator $\mathcal{S}$ could

[3] In other security notion, e.g., black-box and (dummy) UC, an adversary controls the network interface. However, adversary and environment can freely communicate, and hence, coordinate their actions, which effectively gives the environment full access to the network interface.

$\texttt{Corr}(\mathit{corrupted} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{corruptible} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{initialized} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{corrMsg}, T_{\text{adv}}, \mathcal{T}_{\text{user}}, T_{\text{env}})$
Types of tapes: $\mathcal{T}_{\text{user}}$, T_{env} are enriching, T_{adv} is consuming
Local variable: *res* (initial value: 0)

(a) *Corruption Request:* If recv ($\mathsf{Corrupted}?$) from T_{env}, and *initialized* do: send (*corrupted*) to T_{env}
(b) *Corruption:* If recv ($\mathsf{Corrupt}$) from T_{adv}, *corruptible*, *initialized*, and not *corrupted* do: *corrupted* $\leftarrow \mathsf{true}$, send ($\mathsf{Corrupted}$, *corrMsg*) to T_{adv}
(c) *Forward to A:* If recv m from $T \in \mathcal{T}_{\text{user}}$, and *corrupted* do: *res* $\leftarrow 0$, send (Recv, m, T) to T_{adv}
(d) *Forward to user:* If recv (Send, m, T) from T_{adv} where $T \in \mathcal{T}_{\text{user}}$, *corrupted*, and $0 < |m| \leq \mathit{res}$ do: *res* $\leftarrow 0$, send m to T
(e) *Resources:* If recv (Res, r) from T_{env}, and *corrupted* do: *res* $\leftarrow |r|$, send (Res, r) to T_{adv}

Fig. 1. Macro to model adaptive and non-adaptive corruption behavior.

corrupt a functionality at the beginning and then mimic the behavior of the real protocol. The variable *corruptible* allows to block corruptions. If it is set to true all the time, then our macro models adaptive corruption. However, a functionality using our macro can set *corruptible* to false at some point. For example, *corruptible* could be set to false by a functionality after some initialization in order to capture non-adaptive corruption. In this paper, we will use our macro to model both adaptive and non-adaptive corruption.

We note that other forms of corruption could also be modeled. For example, if the adversary does not get to know the whole internal state of a functionality, it might make sense to first let the functionality compute output. However, before sending the output, one could allow the adversary to manipulate this output. To model passive adversaries, one would only inform the adversary about the output that has been produced, but would not provide the adversary with means to manipulate the output. While these forms of corruption can we interesting in some cases, the kinds of corruptions captured by our macro suffices for our purposes.

5 Digital Signatures

In this section, we present our functionality for digital signatures with local computation as explained in the introduction, show that a signature scheme realizes this functionality iff it is EU-CMA secure (where we allow for adaptive corruption of both signers and verifiers), and then provide a joint state realization of our functionality. In Section 5.4, we will compare our formulation of the digital signature functionality with other formulations proposed in the literature. To state our functionality, we first introduce some notational and conceptual conventions, which we will also use for other functionalities and IITMs.

5.1 Ideal Digital Signature Functionality

The basic idea of an ideal functionality for digital signatures is to provide a registration service where the signer can register message signature pairs and the verifiers can check if a pair is registered [5, 7, 1].

Our ideal signature functionality $\mathcal{F}_{\text{SIG}}$ is defined as follows

$$\mathcal{F}_{\text{SIG}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p) = F_{\text{sig}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p) \mid !\underline{F_{\text{ver}}(\mathcal{T}_{\text{ver}})}$$

where F_{sig} and F_{ver} are IITMs.

The IITM $F_{\text{ver}} = F_{\text{ver}}(\mathcal{T}_{\text{ver}})$, as defined in Figure 3, represents the verifier's part and is parameterized by a set of names of tapes $\mathcal{T}_{\text{ver}}$ that is used by verifiers to connect to F_{ver}. Note that one tape might be used by an unbounded number of entities. The party version of F_{ver} is used in $\mathcal{F}_{\text{SIG}}$ to model that every verifier has its own local procedure that she can query to verify messages. Upon initialization, i.e. when the verifier sends an init message, a message is sent to the IITM F_{sig} to guarantee that an instance of it is created. This instance will later be used by F_{ver} upon receiving a verification request (see below). Then, the initialization request is forwarded to the adversary who is supposed to answer it whereon the control is given back to the verifier.

The actual functionality of F_{ver}, i.e. to verify a message signature pair with respect to a public key, is left to F_{sig}. Upon a signature verification request the request is forwarded to F_{sig} (see Figure 3 (c)).

Functionality $F_{\text{sig}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p)$

I/O-tapes: in: $\text{io}(T, \text{sig})$ for each $T \in \mathcal{T}_{\text{sig}}$, $\text{io}(E_{\text{sig}}, \text{sig})$, $\text{io}(\text{ver}, \text{sig})$ (enriching)
out: $\text{io}(\text{sig}, T)$ for each $T \in \mathcal{T}_{\text{sig}} \cup \mathcal{T}_{\text{ver}}$, $\text{io}(\text{sig}, E_{\text{sig}})$, $\text{io}(\text{sig}, \text{ver})$
net-tapes: in: $\text{net}(A_{\text{sig}}, \text{sig})$ (consuming) out: $\text{net}(\text{sig}, A_{\text{sig}})$

Initialization: $s, v, k \leftarrow \bot$; $H \leftarrow \emptyset$; *state* $\leftarrow$ init ; *nokey* $\leftarrow$ true ; *corrupted* $\leftarrow$ false

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{sig}}$, and *state* = init do: *state* $\leftarrow$ (wait, T) ; send (Init) to A_{sig}

(b) *Initialization Response:* If recv (Inited) from A_{sig}, not *nokey*, and *state* = (wait, T) do:
state $\leftarrow$ ok ; send (PublicKey, k) to T

(c) *Wake up:* If recv (pid, WakeUpFromVer) from ver do: send (pid, Ack) to ver

(d) *Key Generation:* If recv (AlgorithmsAndKey, s', v', k') from A_{sig}, *nokey*, and $|s'|, |v'|, |k'| \leq p(\eta)$ do:
$(s, v, k) \leftarrow (s', v', k')$; *nokey* $\leftarrow$ false ; send (Ack) to A_{sig}

(e) *Signature Generation:* If recv (Sign, m) from $T \in \mathcal{T}_{\text{sig}}$, *state* = ok, and not *corrupted* do:
$\sigma \leftarrow \text{sim}_{p(|m|+\eta)}\, s(m)$;
$b \leftarrow \text{sim-det}_{p(|m|+|\sigma|+\eta)}\, v(m, \sigma, k)$;
if $b \neq 1$ then $\sigma \leftarrow \bot$ else $H \leftarrow H \cup \{(m, \sigma)\}$ end ;
send (Signature, σ) to T

(f) *Signature Verification:* If recv (pid, Verify, m, σ, k', T) from ver where $T \in \mathcal{T}_{\text{ver}}$, and not *nokey* do:
$b \leftarrow \text{sim-det}_{p(|m|+|\sigma|+\eta)}\, v(m, \sigma, k')$;
if $b \notin \{0, 1\}$ then $f \leftarrow \bot$ end ;
if $k = k'$ and not *corrupted* and $b = 1$ and not $\exists \sigma' \colon (m, \sigma') \in H$ then $b \leftarrow \bot$ end ;
send (pid, Verified, b) to T

(g) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $\varepsilon, A_{\text{sig}}, \mathcal{T}_{\text{sig}}, E_{\text{sig}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 2. Ideal signature functionality $\mathcal{F}_{\text{SIG}} = F_{\text{sig}} \mid !\underline{F_{\text{ver}}}$, the signer's part F_{sig}.

The only purpose of F_{ver} is to handle registration and corruption in a more uniform and simpler way. One could as well define $\mathcal{F}_{\text{SIG}}$ in a single IITM.

The IITM $F_{\text{sig}} = F_{\text{sig}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p)$, as defined in Figure 2, represents the signer's part of $\mathcal{F}_{\text{SIG}}$ and is parameterized by a polynomial p and two disjoint sets of names of tapes $\mathcal{T}_{\text{sig}}$ and $\mathcal{T}_{\text{ver}}$ which are used by the signer or verifiers (resp.) to connect to F_{sig}. During the registration, the adversary has to provide algorithms s and v for generating and verifying signatures and a public key k. The functionality F_{sig} has *adaptive corruption behavior*. The polynomial p is used to bound the size of s, v and k and the runtime of s and v as described in Section 4. Note that the polynomial does not limit the power of the adversary since upon corruption, the adversary is not anymore restricted to the polynomial. Also, every potential signing or verifying algorithm has polynomial runtime. Therefore, it possible to choose a polynomial such that $\mathcal{F}_{\text{SIG}}$ executes the algorithms as expected.

At first the owner of F_{sig}, i.e. the signer, has to initialize the functionality by sending the message (Init). This message is forwarded to the adversary who is supposed to provide algorithms and a public key which then is forwarded to the signer. Then, the signer can sign messages by sending (Sign, m) to F_{sig}. The signature string σ is generated by running s on m. Then, the pair (m, σ) is recorded and σ is sent to the signer if (m, σ) verifies according to algorithm v with the proper verification key k, i.e. if $v(m, \sigma, k) = 1$. If the pair (m, σ) does not verify, the error symbol $\bot$ is returned to the signer.

Similarly to the signer, each verifier has to initialize its copy of $\underline{F_{\text{ver}}}$. Then, a message is sent to the IITM F_{enc} to guarantee that a copy of it is created. This copy will later be used by $\underline{F_{\text{ver}}}$ to process verification requests. Then, the initialization request is forwarded to the adversary who is supposed to answer it whereon the control is given back to the verifier. If a verifier sends a verification request for a message m and a signature σ with respect to the verification key k' to F_{ver} then this request is forwarded to F_{sig} which outputs the error symbol $\bot$ if the input is a forgery, i.e. if $k = k'$, $v(m, \sigma, k) = 1$, the signer is not corrupted and m was never signed by the signer (there is no σ' such that (m, σ') is recorded). Otherwise, F_{sig} outputs the result of $v(m, \sigma, k')$ to the verifier.

The external tapes of $\mathcal{F}_{\text{SIG}}$ and the connection between F_{sig} and F_{ver} are pictured in Figure 4.

Next, we summarize some properties of $\mathcal{F}_{\text{SIG}}$ to show that it models an ideal signature functionality:

Functionality $F_{\text{ver}}(\mathcal{T}_{\text{ver}})$

I/O-tapes: in: $\text{io}(T, \text{sig})$ for each $T \in \mathcal{T}_{\text{ver}}$, $\text{io}(E_{\text{ver}}, \text{sig})$ (enriching), $\text{io}(\text{sig}, \text{ver})$ (consuming)
out: $\text{io}(\text{sig}, T)$ for each $T \in \mathcal{T}_{\text{ver}}$, $\text{io}(\text{sig}, E_{\text{ver}})$, $\text{io}(\text{ver}, \text{sig})$
net-tapes: in: $\text{net}(A_{\text{ver}}, \text{sig})$ (consuming) out: $\text{net}(\text{sig}, A_{\text{ver}})$

Initialization: *state* $\leftarrow$ init ; *corrupted* $\leftarrow$ false

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{ver}}$, and *state* = init do:
send (WakeUpFromVer) to sig ; wait for recv (Ack) from sig ; *state* $\leftarrow$ (wait, T) ; send (Init) to A_{ver}

(b) *Initialization Response:* If recv (Inited) from A_{ver}, and *state* = (wait, T) do: *state* $\leftarrow$ ok; send (Inited) to T

(c) *Signature Verification:* If recv (Verify, m, σ, k) from $T \in \mathcal{T}_{\text{ver}}$, *state* = ok, and not *corrupted* do:
send (Verify, m, σ, k, T) to sig

(d) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $\varepsilon, A_{\text{ver}}, \mathcal{T}_{\text{ver}}, E_{\text{ver}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 3. Ideal signature functionality $\mathcal{F}_{\text{SIG}} = F_{\text{sig}} \mid !\underline{F_{\text{ver}}}$, the verifier's part F_{ver}.

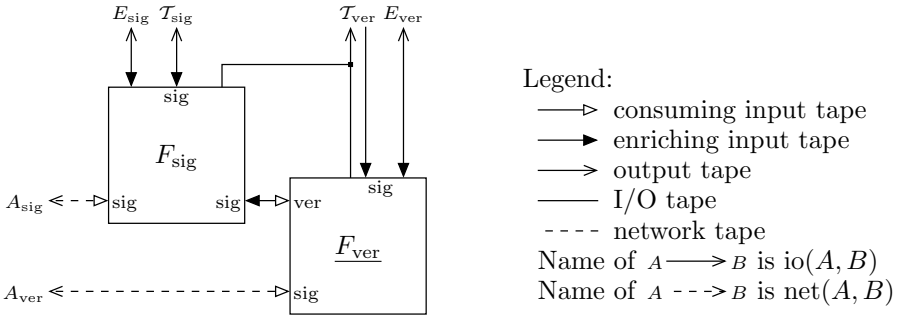

Fig. 4. Graphical representation of the ideal functionality for public-key encryption $\mathcal{F}_{\text{SIG}} = F_{\text{sig}} \mid !\underline{F_{\text{ver}}}$.

– Since the adversary provides the algorithms s and v, no requirements are put on these algorithms.
– Assuming that the signer is not corrupted and a verifier asks to verify (m, σ) with the correct key k we have that:
  $\mathcal{F}_{\text{SIG}}$ *guarantees usability*, i.e. if a message was signed, then a verifier can verify: if the signer signed m before and obtained signature σ then $\mathcal{F}_{\text{SIG}}$ will return 1 because (m, σ) is recorded, v is simulated probabilistically and $v(m, \sigma, k) = 1$ was checked when the signature was generated.
  $\mathcal{F}_{\text{SIG}}$ *guarantees security.* If the signer has not signed m before, then $\mathcal{F}_{\text{SIG}}$ will either return 0 or the error symbol $\perp$. Note that σ could differ from the signature string generated by the signer and $\mathcal{F}_{\text{SIG}}$ returns 1. This captures the fact that it is not insecure that given a signature string for the some message it is possible to produce a different signature string for the same message.
– The verification process is consistent. If a verifier obtained $b \in \{0, 1\}$ from $\mathcal{F}_{\text{SIG}}$ upon a verification request with m, σ, k' then every later verification request (of a not corrupted verifier) with m, σ, k' will result in the same response b.
– If the signer is corrupted then nothing is guaranteed upon verification, i.e. algorithm v alone determines the result of the verification. This enables the adversary to claim signatures of corrupted signers.
– If the verifier provides a wrong key $k' \neq k$ then nothing is guaranteed upon verification as if the signer is corrupted. This models the fact that in a signature scheme signatures are not bound to a party but to a verification key. The functionality $\mathcal{F}_{\text{SIG}}$ does not model a PKI.

5.2 Implementation by an EU-CMA Signature Scheme

In this section it is shown that the ideal signature functionality $\mathcal{F}_{\text{SIG}}$ can be implemented/realized by a signature scheme which is existentially unforgeable under adaptive chosen-message attacks (EU-CMA). Even more, it is proved that if a signature scheme realizes $\mathcal{F}_{\text{SIG}}$ then it is EU-CMA. We allow signers and verifiers to be corrupted adaptively. In what follows, we first define signature schemes and EU-CMA security [19].

A *signature scheme* $\Sigma = (\text{gen}, \text{sig}, \text{ver})$ consists of two probabilistic algorithms gen and sig and a deterministic algorithm ver. The key generation algorithm gen takes 1^η as an input (where η is the security parameter) and outputs a pair of keys (k_s, k_v), the secret (or signing) key k_s and the public (or verification) key k_v. The signature generation algorithm sig expects a secret key k_s and message m as input and produces a signature σ. The signature verification algorithm ver outputs 0 or 1 upon input of a message m, a signature σ and a public key k_v. It outputs 1 iff the message signature pair verifies according to the public key. It is required that:

(a) $\text{gen}(1^\eta)$ can be computed in polynomial time in η,
(b) $\text{sig}(k_s, m)$ can be computed in polynomial time in $|m| + \eta$, and
(c) $\text{ver}(m, \sigma, k_v)$ can be computed in polynomial time in $|m| + |\sigma| + \eta$.

Definition 2. *A signature scheme* $\Sigma = (\text{gen}, \text{sig}, \text{ver})$ *is called* existentially unforgeable under adaptive chosen-message attacks (EU-CMA) *if the following two properties are satisfied:*

(a) Completeness: *For each message* m*,*

$$\text{Prob}[(k_s, k_v) \leftarrow \mathit{gen}(1^\eta), \sigma \leftarrow \text{sig}(k_s, m) \colon 0 = \text{ver}(m, \sigma, k_v)]$$

is negligible (as a function in η*).*

(b) Unforgeability: *For each probabilistic polynomial time Turing machine* F *that can make use of the signing oracle* $\text{sig}(k_s, \cdot)$*,*

$$\text{Prob}[(k_s, k_v) \leftarrow \text{gen}(1^\eta), (m, \sigma) \leftarrow F(\text{sig}(k_s, \cdot), k_v, 1^\eta) \colon 1 = \text{ver}(m, \sigma, k_v) \textit{ and } F \textit{ never asked } \text{sig} \textit{ to sign } m]$$

is negligible (as a function in η*).*

Given a signature scheme $\Sigma = (\text{gen}, \text{sig}, \text{ver})$, it is straight-forward to obtain a system of IITMs $\mathcal{P}_{\text{SIG}}$ that models the signature scheme as a protocol in the IITM model. Let $\mathcal{T}_{\text{sig}}$ and $\mathcal{T}_{\text{ver}}$ be two disjoint sets of names of tapes, like in Section 5.1. We define

$$\mathcal{P}_{\text{SIG}}(\Sigma, \mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}) = P_{\text{sig}}(\text{gen}, \text{sig}, \mathcal{T}_{\text{sig}}) \mid !\underline{P_{\text{ver}}(\text{ver}, \mathcal{T}_{\text{ver}})}$$

Realization $P_{\text{sig}}(\text{gen}, \text{sig}, \mathcal{T}_{\text{sig}})$

I/O-tapes: in: $\text{io}(T, \text{sig})$ for each $T \in \mathcal{T}_{\text{sig}}$, $\text{io}(E_{\text{sig}}, \text{sig})$ (enriching)
out: $\text{io}(\text{sig}, T)$ for each $T \in \mathcal{T}_{\text{sig}}$, $\text{io}(\text{sig}, E_{\text{sig}})$
net-tapes: in: $\text{net}(A'_{\text{sig}}, \text{sig})$ (consuming) out: $\text{net}(\text{sig}, A'_{\text{sig}})$
Initialization: *state* $\leftarrow$ init ; $k_s, k_v \leftarrow \perp$; *corrupted* $\leftarrow$ false
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{sig}}$, and *state* = init do:
$(k_s, k_v) \leftarrow \text{gen}(1^\eta)$; *state* $\leftarrow$ ok ; send (PublicKey, k_v) to T
(b) *Signature Generation:* If recv (Sign, m) from $T \in \mathcal{T}_{\text{sig}}$, *state* = ok, and not *corrupted* do:
$\sigma \leftarrow \text{sig}(k_s, m)$; send (Signature, σ) to T
(c) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $(k_s, k_v), A'_{\text{sig}}, \mathcal{T}_{\text{sig}}, E_{\text{sig}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 5. Realization of a digital signature scheme $\mathcal{P}_{\text{SIG}} = P_{\text{sig}} \mid !\underline{P_{\text{ver}}}$, the signer's part P_{sig}.

Realization $P_{\text{ver}}(\text{ver}, \mathcal{T}_{\text{ver}})$

I/O-tapes: in: $\text{io}(T, \text{sig})$ for each $T \in \mathcal{T}_{\text{ver}}$, $\text{io}(E_{\text{ver}}, \text{sig})$ (enriching)
out: $\text{io}(\text{sig}, T)$ for each $T \in \mathcal{T}_{\text{ver}}$, $\text{io}(\text{sig}, E_{\text{ver}})$
net-tapes: in: $\text{net}(A'_{\text{ver}}, \text{sig})$ (consuming) out: $\text{net}(\text{sig}, A'_{\text{ver}})$
Initialization: *state* $\leftarrow$ init ; *corrupted* $\leftarrow$ false
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{ver}}$, and *state* = init do: *state* $\leftarrow$ ok ; send (Inited) to T
(b) *Signature Verification:* If recv (Verify, m, σ, k) from $T \in \mathcal{T}_{\text{ver}}$, *state* = ok, and not *corrupted* do:
$b \leftarrow \text{ver}(m, \sigma, k)$; send (Verified, b) to T
(c) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $\varepsilon, A'_{\text{ver}}, \mathcal{T}_{\text{ver}}, E_{\text{ver}}$) (See Figure 1 for the definition of Corr)

If no rule above fires then produce no output.

Fig. 6. Realization of a digital signature scheme $\mathcal{P}_{\text{SIG}} = P_{\text{sig}} \mid !\underline{P_{\text{ver}}}$, the verifier's part P_{ver}.

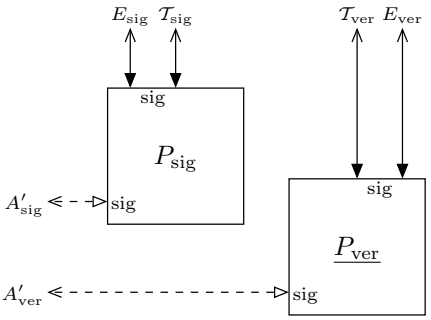

Fig. 7. Graphical representation of the implementation of a digital signature scheme $\mathcal{P}_{\text{SIG}} = P_{\text{sig}} \mid !\underline{P_{\text{ver}}}$. See Figure 4 for a legend.

where P_{sig} and P_{ver} are two IITMs as specified in Figure 5 and 6, respectively. A graphical representation of $\mathcal{P}_{\text{SIG}}$ is depicted in Figure 7.

The IITM $P_{\text{sig}} = P_{\text{sig}}(\text{gen}, \text{sig}, \mathcal{T}_{\text{sig}})$ belongs to the signer. Upon receiving an initialization request, a private and a public key are generated with the key generation algorithm gen and the public key is returned to the signer. When the signer requests to sign a message m then P_{sig} computes the signature string σ with the signing algorithm $\text{sig}(k_s, m)$ where k_s is the previously generated private key and returns σ to the signer. P_{sig} has adaptive corruption behavior, i.e. the signer is adaptive corruptible, as described in Section 4 and reveals the private and public key upon corruption.

Each copy of $\underline{P_{\text{ver}}} = \underline{P_{\text{ver}}(\text{ver}, \mathcal{T}_{\text{ver}})}$ belongs to one verifier. We denote the instance of a party with PID *pid* by $P_{\text{ver}}[pid]$. Initialization requests are directly answered positively. When the verifier *pid* requests to verify (m, σ, k) then $P_{\text{ver}}[pid]$ computes $b \leftarrow \text{ver}(m, \sigma, k)$ and returns the bit b to the verifier *pid*. Just as P_{sig}, P_{ver} has adaptive corruption behavior.

The following theorem shows that an EU-CMA signature scheme realizes/implements the ideal functionality $\mathcal{F}_{\text{SIG}}$ from Section 5.1 and that if $\mathcal{P}_{\text{SIG}}(\Sigma)$ implements $\mathcal{F}_{\text{SIG}}$ then Σ is EU-CMA. The basic idea of the proof is similar to Canetti's proofs [6, 7] and the one by Backes and Hofheinz [1].

Let p be a polynomial. A signature scheme $\Sigma = (\text{gen}, \text{sig}, \text{ver})$ is called *p-bounded* if the runtime of $\text{gen}(1^\eta)$, $\text{sig}(k_s, m)$ and $\text{ver}(m, \sigma, k_v)$ is bounded by $p(\eta)$, $p(|m| + \eta)$ and $p(|m| + |\sigma| + \eta)$ (resp.) for every η, m, σ, k_s and k_v and if the length of the description of gen, sig and ver is bounded by $p(\eta)$. Note that for each signature scheme, there is a polynomial p such that it is p-bounded.

The following theorem shows that a signature scheme is EU-CMA iff it realizes $\mathcal{F}_{\text{SIG}}$ in the context of environments without auxiliary input. One could alternatively define EU-CMA by allowing auxiliary input to the forger. Then, a signature would be EU-CMA iff it would realize $\mathcal{F}_{\text{SIG}}$ in the context of environments with auxiliary input.

Theorem 5. *Let $\Sigma = (\text{gen}, \text{sig}, \text{ver})$ be a p-bounded signature scheme. Then, Σ is EU-CMA if and only if $\mathcal{P}_{\text{SIG}}(\Sigma, \mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\text{SIG}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p)$.*

Proof. We abbreviate $\mathcal{P}_{\text{SIG}}(\Sigma, \mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}})$ by $\mathcal{P}_{\text{SIG}}$ and $\mathcal{F}_{\text{SIG}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p)$ by $\mathcal{F}_{\text{SIG}}$. It is easy to see that $\mathcal{P}_{\text{SIG}}$ and $\mathcal{F}_{\text{SIG}}$ are I/O compatible protocols.

At first we prove that if $\mathcal{P}_{\text{SIG}} \leq^{SS\text{-}noaux} \mathcal{F}_{\text{SIG}}$ then Σ is EU-CMA by contraposition. Therefore, we assume that Σ is not EU-CMA and show that for all simulators $\mathcal{S} \in \text{Sim}_{\mathbf{S}}^{\mathcal{P}_{\text{SIG}}}(\mathcal{F}_{\text{SIG}})$ there is an environment $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{P}_{\text{SIG}})$ such that $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}} \not\equiv^{noaux} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{SIG}}$, i.e. that there is no negligible function g with

$$g(1^\eta) = |\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \text{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \ .$$

If Σ is not complete, i.e. it violates Definition 2 (a), then

$$\text{Prob}[(k_s, k_v) \leftarrow \text{gen}(1^\eta), \sigma_0 \leftarrow \text{sig}(k_s, m_0), 0 = \text{ver}(m_0, \sigma_0, k_v)]$$

is not negligible (as a function in η) for some message m_0.

The environment $\mathcal{E}$ can be defined independently from the simulator $\mathcal{S}$. Let $T \in \mathcal{T}_{\text{sig}}$ and $T' \in \mathcal{T}_{\text{ver}}$. We define $\mathcal{E}$ to be a master IITM (an IITM with a tape named start) with an output tape named decision and tapes to connect to $\mathcal{P}_{\text{SIG}}$. In mode CheckAddress $\mathcal{E}$ accepts every incoming message and in mode Compute it operates as follows:

(a) Upon first activation (on tape start) output (Init) on tape $\text{io}(T, \text{sig})$.
(b) Upon receiving $(\text{PublicKey}, k)$ on tape $\text{io}(\text{sig}, T)$ for some k, store k and output (Sign, m_0) on $\text{io}(T, \text{sig})$.
(c) Upon receiving $(\text{Signature}, \sigma)$ on $\text{io}(\text{sig}, T)$ for some σ, store σ and output $(pid, \text{Verify}, m_0, \sigma, k)$ on $\text{io}(T', \text{sig})$.
(d) Upon receiving $(pid, \text{Verified}, 0)$ on $\text{io}(\text{sig}, T')$ do: check if the signer is corrupted or if *pid* is a corrupted verifier, i.e.:
  - Send (Corrupted?) on $\text{io}(\text{sig}, T)$.
  - Upon receiving (true) on $\text{io}(T, \text{sig})$ output 1 on tape decision and halt else send $(pid, \text{Corrupted?})$ on $\text{io}(\text{sig}, T')$.
  - Upon receiving (true) $\text{io}(T', \text{sig})$ output 1 on tape decision and halt else output 0 on tape decision and halt.

If at some point above $\mathcal{E}$ waits for a message to receive and the input is not as expected or on an unexpected tape then $\mathcal{E}$ outputs 1 on tape decision and halts. One easily verifies that $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P}_{\mathrm{SIG}})$.

In the ideal world $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ outputs 1 for each simulator $\mathcal{S} \in \mathrm{Sim}_{\mathbf{S}}^{\mathcal{P}_{\mathrm{SIG}}}(\mathcal{F}_{\mathrm{SIG}})$, security parameter η and initial input a. Assume that in a run of $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ the environment $\mathcal{E}$ outputs 0, i.e. $\mathcal{E}$ reaches the last line in (d). Then, the signer and verifier with PID *pid* are both not corrupted, $\mathcal{F}_{\mathrm{SIG}}$ sent $(\mathsf{Signature}, \sigma)$ and $\mathcal{F}_{\mathrm{SIG}}$ sent $(pid, \mathsf{Verified}, 0)$. $\mathcal{F}_{\mathrm{SIG}}$ sent $(\mathsf{Signature}, \sigma)$ implies that $(m_0, \sigma) \in H$ and $ver(m_0, \sigma, k) = 1$. Thus, $\mathcal{F}_{\mathrm{SIG}}$ returned $(pid, \mathsf{Verified}, 1)$ upon the verification request of $\mathcal{E}$ which is a contradiction.

In the real world $\mathcal{E}$ will always receive what it expects, except in (d) where it possibly receives $(pid, \mathsf{Verified}, 1)$ instead of $(pid, \mathsf{Verified}, 0)$, because of the definition of $\mathcal{P}_{\mathrm{SIG}}$ and because $\mathcal{E}$ does not corrupt anyone. Thus, $(\mathcal{E} \,|\, \mathcal{P}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ outputs 0 if and only if $\mathcal{P}_{\mathrm{SIG}}$ returned $(pid, \mathsf{Verified}, 0)$, which occurs with probability

$$\mathrm{Prob}[(k_s, k_v) \leftarrow gen(1^\eta), \sigma \leftarrow \mathrm{sig}(k_s, m_0) \colon 0 = ver(m_0, \sigma, k_v)]$$

(for each η) which is not negligible by assumption. Thus, we have that

$$\begin{aligned}
&|\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \\
&= |1 - \mathrm{Prob}[(k_s, k_v) \leftarrow gen(1^\eta), \sigma_0 \leftarrow \mathrm{sig}(k_s, m_0) \colon 0 = ver(m_0, \sigma_0, k_v)] - 1| \\
&= \mathrm{Prob}[(k_s, k_v) \leftarrow gen(1^\eta), \sigma_0 \leftarrow \mathrm{sig}(k_s, m_0) \colon 0 = \mathrm{ver}(m_0, \sigma_0, k_v)]
\end{aligned}$$

is not negligible and therefore $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{SIG}} \not\equiv^{noaux} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}}$.

On the other hand, if Σ is forgeable, i.e. it violates Definition 2 (b), then there is a probabilistic polynomial time Turing machine F that can make use of the signing oracle $\mathrm{sig}(k_s, \cdot)$ and

$$\begin{aligned}
\mathrm{Prob}[(k_s, k_v) \leftarrow gen(1^\eta), (m, \sigma) \leftarrow F(\mathrm{sig}(k_s, \cdot), k_v, 1^\eta) \colon 1 = \mathrm{ver}(m, \sigma, k_v) \text{ and} \\
F \text{ never asked } sig \text{ to sign } m]
\end{aligned}$$

is not negligible (as a function in η).

The environment $\mathcal{E}$ can be defined independently from the simulator $\mathcal{S}$. Let $T \in \mathcal{T}_{\mathrm{sig}}$ and $T' \in \mathcal{T}_{\mathrm{ver}}$. We define $\mathcal{E}$ to be a master IITM (an IITM with a tape named start) with an output tape named decision and tapes to connect to $\mathcal{P}_{\mathrm{SIG}}$. In mode CheckAddress $\mathcal{E}$ accepts every incoming message and in mode Compute it operates as follows:

(a) Upon first activation (on tape start), output (Init) on tape $\mathrm{io}(T, \mathrm{sig})$.
(b) Wait for receiving $(\mathsf{PublicKey}, k)$ on tape $\mathrm{io}(\mathrm{sig}, T)$, store k.
(c) Simulate the forger F with input k as the public key. If F asks its signing oracle to sign a message m then output (Sign, m) on $\mathrm{io}(T, \mathrm{sig})$ and wait for receiving $(\mathsf{Signature}, \sigma)$ on $\mathrm{io}(\mathrm{sig}, T)$. Then continue simulating F as if the oracle returned σ. The output of F will be a pair (m_0, σ_0).
(d) If the message m_0 was signed before then halt with output 0 on tape decision else output the message $(pid, \mathsf{Verify}, m_0, \sigma_0, k)$ on $\mathrm{io}(T', \mathrm{sig})$.
(e) Upon receiving $(pid, \mathsf{Verified}, 1)$ on $\mathrm{io}(\mathrm{sig}, T')$ do: check if the signer is corrupted or if *pid* is a corrupted verifier, i.e.:
   - Send (Corrupted?) on $\mathrm{io}(\mathrm{sig}, T)$.
   - Upon receiving (true) on $\mathrm{io}(T, \mathrm{sig})$ output 0 on tape decision and halt else send $(pid, \mathsf{Corrupted}?)$ on $\mathrm{io}(\mathrm{sig}, T')$.
   - Upon receiving (true) on $\mathrm{io}(T', \mathrm{sig})$ output 0 on tape decision and halt else output 1 on tape decision and halt.

If at some point above $\mathcal{E}$ waits for a message to receive and the input is not as expected or on an unexpected tape then $\mathcal{E}$ outputs 0 on tape decision and halts. One easily verifies that $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P}_{\mathrm{SIG}})$.

In the ideal world, $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ outputs 0 for each simulator $\mathcal{S} \in \mathrm{Sim}_{\mathbf{S}}^{\mathcal{P}_{\mathrm{SIG}}}(\mathcal{F}_{\mathrm{SIG}})$ and security parameter η. Assume that in a run of $(\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ the environment $\mathcal{E}$ outputs 1, i.e. $\mathcal{E}$ reaches the last line in (e). Then, m_0 was never signed before, the signer and verifier with PID *pid* are both not corrupted and $\mathcal{F}_{\mathrm{SIG}}$ sent $(pid, \mathsf{Verified}, 1)$. Since m_0 was not signed before, $(m_0, \sigma) \notin H$ for all σ. Thus, $\mathcal{F}_{\mathrm{SIG}}$ did *not* send $(pid, \mathsf{Verified}, 1)$ by the definition of F_{sig} (see Figure 2 (f)). Which is a contradiction.

In the real world, $\mathcal{E}$ will always receive what it expects, except in (e) where it possibly receives $(pid, \mathsf{Verified}, 0)$ instead of $(pid, \mathsf{Verified}, 1)$, because of the definition of $\mathcal{P}_{\mathrm{SIG}}$ and because $\mathcal{E}$ does not

Simulator $\mathcal{S}_{\mathrm{SIG}}(\Sigma)$

net-tapes: in: $\mathrm{net}(\mathrm{sig}, A_{\mathrm{sig}})$, $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$, $\mathrm{net}(A'_{\mathrm{sig}}, \mathrm{sig})$, $\mathrm{net}(A'_{\mathrm{ver}}, \mathrm{sig})$ (enriching)
out: $\mathrm{net}(A_{\mathrm{sig}}, \mathrm{sig})$, $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$, $\mathrm{net}(\mathrm{sig}, A'_{\mathrm{sig}})$, $\mathrm{net}(\mathrm{sig}, A'_{\mathrm{ver}})$

Initialization: *nokey* $\leftarrow$ true ; $(k_s, k_v) \leftarrow \mathrm{gen}(1^\eta)$

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

(a) *Initialization (Sig):* If recv (Init) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{sig}})$ do:
if *nokey* then
send $(\mathsf{AlgorithmsAndKey}, \mathrm{sig}(k_s, \cdot), \mathrm{ver}(\cdot, \cdot, \cdot), k_v)$ to $\mathrm{net}(A_{\mathrm{sig}}, \mathrm{sig})$;
wait for recv (Ack) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{sig}})$; *nokey* $\leftarrow$ false
end ;
send (Inited) to $\mathrm{net}(A_{\mathrm{sig}}, \mathrm{sig})$

(b) *Forward messages from A (Sig and Ver):* If recv m on $\mathrm{net}(A'_{\mathrm{sig}}, \mathrm{sig})$ or $\mathrm{net}(A'_{\mathrm{ver}}, \mathrm{sig})$ do:
send m to $\mathrm{net}(A_{\mathrm{sig}}, \mathrm{sig})$ or $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$ (resp.)

(c) *Forward messages to A (Sig):* If recv m on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{sig}})$ and $m \neq$ (Init) do:
if $m =$ (Corrupted) then send $(\mathsf{Corrupted}, (k_s, k_v))$ to $\mathrm{net}(\mathrm{sig}, A'_{\mathrm{sig}})$ else send m to $\mathrm{net}(\mathrm{sig}, A'_{\mathrm{sig}})$ end

(d) *Initialization (Ver):* If recv (pid, Init) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ do:
if *nokey* then
send $(\mathsf{AlgorithmsAndKey}, \mathrm{sig}(k_s, \cdot), \mathrm{ver}(\cdot, \cdot, \cdot), k_v)$ to $\mathrm{net}(A_{\mathrm{sig}}, \mathrm{sig})$;
wait for recv (Ack) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{sig}})$; *nokey* $\leftarrow$ false
end ;
send (pid, Inited) to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$

(e) *Forward messages to A (Ver):* If recv m on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ and $m \neq (pid, \mathsf{Init})$ do: send m to $\mathrm{net}(\mathrm{sig}, A'_{\mathrm{ver}})$

If no rule above fires then produce no output.

Fig. 8. Simulator $\mathcal{S}_{\mathrm{SIG}}$ for the proof of $\mathcal{P}_{\mathrm{SIG}}$ SS-realizes $\mathcal{F}_{\mathrm{SIG}}$

corrupt anyone. Thus, $(\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}})(1^\eta, \varepsilon)$ outputs 1 if and only if $\mathcal{P}_{\mathrm{SIG}}$ returned $(pid, \mathsf{Verified}, 1)$ which occurs with probability

$$\mathrm{Prob}[(k_s, k_v) \leftarrow \mathrm{gen}(1^\eta), (m, \sigma) \leftarrow F(\mathrm{sig}(k_s, \cdot), k_v, 1^\eta) \colon 1 = ver(m, \sigma, k_v) \text{ and } F \text{ never asked } sig \text{ to sign } m]$$

which is not negligible by assumption. As above, we conclude that $\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}} \not\equiv^{noaux} \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{SIG}}$.

Now, we prove by contraposition that if Σ is EU-CMA, then

$$\mathcal{P}_{\mathrm{SIG}}(\Sigma, \mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p).$$

Assume that for all simulators $\mathcal{S} \in \mathrm{Sim}_{\mathbf{S}}^{\mathcal{P}_{\mathrm{SIG}}}(\mathcal{F}_{\mathrm{SIG}})$ there is an environment $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P}_{\mathrm{SIG}})$ such that $\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}} \not\equiv^{noaux} \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{SIG}}$. If Σ is not complete then we are done. Thus, we assume that Σ is complete. For each $\mathcal{S} \in \mathrm{Sim}_{\mathbf{S}}^{\mathcal{P}_{\mathrm{SIG}}}(\mathcal{F}_{\mathrm{SIG}})$ there is an environment $\mathcal{E} \in \mathrm{Con}_{\mathbf{E}}(\mathcal{P}_{\mathrm{SIG}})$ such that $\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}} \not\equiv^{noaux} \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{SIG}}$, especially for the simulator $\mathcal{S}_{\mathrm{SIG}}(\Sigma)$ ($\mathcal{S}_{\mathrm{SIG}}$ for short) as defined in Figure 8. A graphical representation of $\mathcal{S}_{\mathrm{SIG}}$ and its connection to $\mathcal{F}_{\mathrm{SIG}}$ is depicted in Figure 9. One easily verifies that $\mathcal{S}_{\mathrm{SIG}} \in \mathrm{Sim}_{\mathbf{S}}^{\mathcal{P}_{\mathrm{SIG}}}(\mathcal{F}_{\mathrm{SIG}})$. We will show how $\mathcal{E}$ can be used to construct a successful forger F.

Because $\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}} \not\equiv^{noaux} \mathcal{E} \mid \mathcal{S} \mid \mathcal{F}_{\mathrm{SIG}}$ there are polynomials p such that for all $\eta_0 \in \mathbb{N}$ there is $\eta > \eta_0$ such that

$$|\mathrm{Prob}[\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{E} \mid \mathcal{S}_{\mathrm{SIG}} \mid \mathcal{F}_{\mathrm{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| > \frac{1}{p(\eta)} \quad . \tag{2}$$

Let $\eta_0 \in \mathbb{N}$ and choose $\eta > \eta_0$ such that (2) holds. The input of F is a signing oracle $S(\cdot)$, a public key k, and 1^η. F simulates a run of $\mathcal{E} \mid \mathcal{P}_{\mathrm{SIG}}$ with the security parameter η with the following exceptions:

(a) If P_{sig} wants to simulate $\mathrm{gen}(1^\eta)$ (Figure 5 (a)) then continue the simulation as if $\mathrm{gen}(1^\eta)$ returned 0 as the signing key and k as the verification key.
(b) If P_{sig} wants to simulate $\mathrm{sig}(k_s, m)$ (Figure 5 (b)) then F computes $\sigma \leftarrow S(m)$ with its signing oracle and continues the simulation as if the simulation of $\mathrm{sig}(k_s, m)$ in P_{sig} returned σ.
(c) If some copy of $\underline{P_{\mathrm{ver}}}$ wants to simulate $\mathrm{ver}(m, \sigma, k')$ (Figure 6 (b)) then F checks if (m, σ) is a forgery, i.e. if m was never signed by S before and $\mathrm{ver}(m, \sigma, k) = 1$. If it is a forgery F outputs (m, σ)

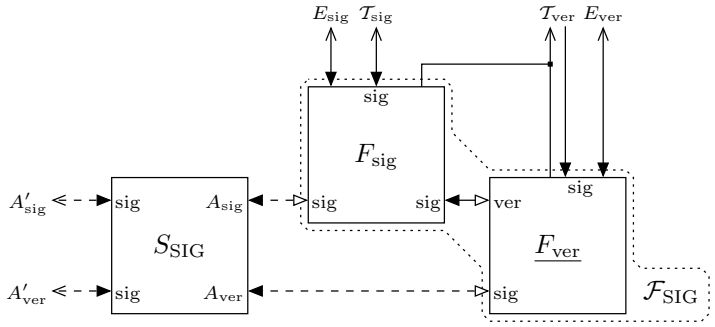

Fig. 9. Graphical representation of the simulator S_{SIG} for $\mathcal{F}_{\text{SIG}} = F_{\text{sig}} \mid !\underline{F_{\text{ver}}}$. See Figure 4 for a legend.

and halts, else F computes $b \leftarrow \text{ver}(m, \sigma, k')$ and continues the simulation as if the simulation of $\text{ver}(m, \sigma, k')$ in $\underline{P_{\text{ver}}}$ returned b.

(d) If the signer gets corrupted, i.e. if P_{sig} sets *corrupted* $\leftarrow$ true, then F halts and produces a failure output.

We now analyze the success probability of F, i.e. the probability that F produces a message signature pair that constitutes a forgery:

$$\begin{aligned}\text{Prob}[(k_s, k_v) \leftarrow \text{gen}(1^\eta), (m, \sigma) \leftarrow F(\text{sig}(k_s, \cdot), k_v, 1^\eta) \colon 1 = \text{ver}(m, \sigma, k_v) \text{ and} \\ F \text{ never asked sig to sign } m] \ .\end{aligned}$$

In analogy to [7], let B denote the event that in a run of $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon)$ some copy of $\underline{P_{\text{ver}}}$ wants to simulate $\text{ver}(m, \sigma, k')$ where $\text{ver}(m, \sigma, k) = 1$, the signer is not corrupted and P_{sig} never computed $\text{sig}(k_s, m)$.

We will prove that as long as event B does not occur, $\mathcal{E}$ can not distinguish the real world from the ideal world. Note that Σ is p-bounded and therefore $\mathcal{F}_{\text{SIG}}$ accepts the algorithms received from $\mathcal{S}_{\text{SIG}}$ and is always able to simulate them till the end. A careful look at $\mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}$ and $\mathcal{P}_{\text{SIG}}$ shows that the only two reasons where $\mathcal{E}$ can distinguish $\mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}$ from $\mathcal{P}_{\text{SIG}}$ are

(a) upon signature generation, if in F_{sig} a signature is produced that does not verify, i.e. if $v(m, \sigma, k) = 0$, and

(b) upon signature verification, if the signer is not corrupted, F_{sig} computes $v(m, \sigma, k') = 1$ and m was not signed before, i.e. there is no σ' such that $(m, \sigma') \in H$.

If (b) would happen then event B would occur. Since we have assumed that Σ is complete, (a) occurs only with negligible probability. Also, $\mathcal{E}$ can only sign polynomially many messages, so $\mathcal{E}$'s view differs only with negligible probability as long as B does not occur.

At next we prove that B will occur in the run of $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon)$ with non-negligible probability. Therefore, we assume that it occurs with negligible probability and deduce a contradiction. By the triangle inequality we obtain

$$\begin{aligned}&|\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \\ &= |\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, B] + \text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, \text{not } B] \\ &\qquad - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \\ &\leq |\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, B]| \\ &\qquad + |\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, \text{not } B] - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \\ &\leq \text{Prob}[B] + |\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, \text{not } B] \\ &\quad - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \ .\end{aligned}$$

We assumed that

$$|\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1, \text{not } B] - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]|$$

and Prob[B] is negligible, thus,

$$|\text{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \text{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}(1^\eta, \varepsilon) \rightsquigarrow 1]|$$

is negligible, too, which implies that $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}} \equiv^{noaux} \mathcal{E} \,|\, \mathcal{S}_{\text{SIG}} \,|\, \mathcal{F}_{\text{SIG}}$. This contradiction proves that event B occurs with non-negligible probability in the run of $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon)$.

The run that F simulates does not differ from a run of $\mathcal{E} \,|\, \mathcal{P}_{\text{SIG}}(1^\eta, \varepsilon)$ and since B can only occur before the signer is corrupted it follows that whenever B occurs, F produces a forgery, i.e.,

$$\begin{aligned}\text{Prob}[B] = \text{Prob}[(k_s, k_v) \leftarrow \text{gen}(1^\eta), (m, \sigma) \leftarrow F(\text{sig}(k_s, \cdot), k_v, 1^\eta) \colon 1 = \text{ver}(m, \sigma, k_v) \text{ and} \\ F \text{ never asked sig to sign } m] \ .\end{aligned}$$

Since Prob[B] is non-negligible, Σ is not unforgeable, so, not EU-CMA. □

5.3 Joint State for Digital Signatures

We now present a joint state realization of $\mathcal{F}_{\text{SIG}}$. In this realization, only one copy of $\mathcal{F}_{\text{SIG}}$ per party instead of one copy per session per party is used.

First we specify the joint state realization $\mathcal{P}^{\text{JS}}_{\text{SIG}}$, which runs with one copy of $\mathcal{F}_{\text{SIG}}$ per party. The joint state theorem for digital signatures then basically says

$$!\mathcal{P}^{\text{JS}}_{\text{SIG}} \,|\, !\underline{\mathcal{F}'_{\text{SIG}}} \leq^{SS} !\underline{\underline{\mathcal{F}_{\text{SIG}}}}$$

where $\mathcal{F}'_{\text{SIG}}$ is identical to $\mathcal{F}_{\text{SIG}}$ except that tapes have been renamed because the left side has to be I/O compatible to the right side. As described Section 3, on the right-hand side we have a multi-session multi-party version, i.e. the inner part $!\underline{\mathcal{F}_{\text{SIG}}}$ is the multi-party version of $\mathcal{F}_{\text{SIG}}$ where we have one copy of $\underline{\mathcal{F}_{\text{SIG}}}$ per party and the outer part is the multi-session version of the multi-party version of $\mathcal{F}_{\text{SIG}}$. So, altogether $!\underline{\underline{\mathcal{F}_{\text{SIG}}}}$ contains one copy of $\mathcal{F}_{\text{SIG}}$ per session per party. On the other side, we have only the multi-party version $!\underline{\mathcal{F}'_{\text{SIG}}}$ of $\mathcal{F}'_{\text{SIG}}$ and the "multiplexer" $!\mathcal{P}^{\text{JS}}_{\text{SIG}}$ which in a run will contain one copy of $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ for each party and this copy handles all sessions of this party through one copy of $\mathcal{F}'_{\text{SIG}}$.

The basic idea of $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ is simple and similar to the so called "concatenate and sign" protocol of Canetti and Rabin [14]: when a party in session sid requests to sign a message m then $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ forwards the requests to sign the message (sid, m) to the same copy of $\mathcal{F}_{\text{SIG}}$ for all sessions. Similarly, when a party in session sid requests to verify (m, σ, k) then $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ forwards the requests to verify $((sid, m), \sigma, k)$ to the same copy of $\mathcal{F}_{\text{SIG}}$ for all sessions. However, this simple idea only works given an appropriate formulation of the digital signature functionality (see Section 5.4).

The implementation $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ is parameterized, as $\mathcal{F}_{\text{SIG}}$, by two disjoint sets of names of tapes $\mathcal{T}_{\text{sig}}$ and $\mathcal{T}_{\text{ver}}$, two polynomials p and q, and is given by the composition of two IITMs

$$\mathcal{P}^{\text{JS}}_{\text{SIG}}(\mathcal{T}_{\text{sig}}, \mathcal{T}_{\text{ver}}, p, q) = P^{\text{JS}}_{\text{sig}}(\mathcal{T}_{\text{sig}}, p, q) \,|\, P^{\text{JS}}_{\text{ver}}(\mathcal{T}_{\text{ver}}, p, q)$$

as defined in Figure 10 and 11. A graphical representation and its connection to $\underline{\mathcal{F}'_{\text{SIG}}}$ is pictured in Figure 12.

A problem is that whenever the ideal functionality $\underline{\underline{\mathcal{F}_{\text{SIG}}(q)}}$ has to sign the message m, $\underline{\mathcal{F}'_{\text{SIG}}(p)}$ is requested to sign (sid, m). By the definition of $\mathcal{F}_{\text{SIG}}$, $\underline{\underline{\mathcal{F}_{\text{SIG}}(q)}}$ simulates s on input m at most $q(|m| + \eta)$ steps while $\underline{\mathcal{F}'_{\text{SIG}}(p)}$ simulates s' on input (sid, m) at most $p(|(sid, m)| + \eta)$ steps. A successful simulator has to provide $s(\cdot) = s'((sid, \cdot))$. Still, no matter what polynomial q is taken, an environment could distinguish the ideal world from the joint state world (JS world) by providing very long SIDs and forcing $\mathcal{F}_{\text{SIG}}$ in the ideal world to abort the computation of s' while $\mathcal{F}_{\text{SIG}}$ in the JS world could complete the computation of s. To overcome this problem, $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ restricts the length of SIDs to be polynomially bounded by the security parameter η. We emphasize that this does not restrict the usability and expressiveness of $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ since no protocol, environment, or adversary that is polynomially bounded by η could create exponentially many different sessions. A more complex definition of polynomial time, e.g., the formulation of reactive polynomial time [22], might solve the problem of long SIDs. However, as discussed in Section 4.3, one cannot dispense with parameters altogether.

One technical problem is that of multiple initialization requests where a functionality has to wait for a response from the environment/adversary. For example, if the environment sent an initialization request

Realization $P^{\mathrm{JS}}_{\mathrm{sig}}(\mathcal{T}_{\mathrm{sig}}, p, q)$

I/O-tapes: in: $\mathrm{io}(T, \mathrm{sig})$ for each $T \in \mathcal{T}_{\mathrm{sig}}$, $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig})$ (enriching), $\mathrm{io}(\mathrm{sig}', T)$ for each $T \in \mathcal{T}_{\mathrm{sig}}$, $\mathrm{io}(\mathrm{sig}', E_{\mathrm{sig}})$ (consuming)
out: $\mathrm{io}(\mathrm{sig}, T)$ for each $T \in \mathcal{T}_{\mathrm{sig}}$, $\mathrm{io}(\mathrm{sig}, E_{\mathrm{sig}})$, $\mathrm{io}(T, \mathrm{sig}')$ for each $T \in \mathcal{T}_{\mathrm{sig}}$, $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig}')$

Initialization: $pid \leftarrow \bot$; $Sids \leftarrow \emptyset$; $AllSids \leftarrow \emptyset$; $lastSid \leftarrow \bot$; $res \leftarrow 0$; $k \leftarrow \bot$; $state \leftarrow \mathsf{init}$

CheckAddress: Upon first activation accept only messages of shape (sid, pid', m) on all sig-tapes (i.e. on $\mathrm{io}(T, \mathrm{sig})$, $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig})$). In mode $\mathsf{Compute}$ at first record the PID: $pid \leftarrow pid'$.
On later activations accept only messages of shape (sid, pid', m) on all sig-tapes and of shape (pid', m) on all sig′-tapes where $pid' = pid$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, \mathsf{Init})$ on $\mathrm{io}(T, \mathrm{sig})$ where $T \in \mathcal{T}_{\mathrm{sig}}$, $sid \notin AllSids$ do:
$AllSids \leftarrow AllSids \cup \{sid\}$;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then
  produce no output
else if $state = \mathsf{wait}$ then
  $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
  $state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid, Init) to $\mathrm{io}(T, \mathrm{sig}')$
else
  $Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, \mathsf{PublicKey}, k)$ to $\mathrm{io}(\mathrm{sig}, T)$
end

(b) *Initialization Response:* If recv $(pid, \mathsf{PublicKey}, k')$ on $\mathrm{io}(\mathrm{sig}', T)$ where $T \in \mathcal{T}_{\mathrm{sig}}$, $|k'| \leq p(\eta)$, and $state = \mathsf{wait}$ do: $state \leftarrow \mathsf{ok}$; $k \leftarrow k'$; $Sids \leftarrow \{lastSid\}$; send $(lastSid, pid, \mathsf{PublicKey}, k)$ to $\mathrm{io}(\mathrm{sig}, T)$

(c) *Forward to $\mathcal{F}_{\mathrm{SIG}}$:* If recv (sid, pid, m) on $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig})$ or $\mathrm{io}(T, \mathrm{sig})$ where $T \in \mathcal{T}_{\mathrm{sig}}$, $state = \mathsf{ok}$, $sid \in Sids$, and $m \neq \mathsf{Corrupted}?$ do:

$$m' \leftarrow \begin{cases} (\mathsf{Sign}, (sid, m'')) & \text{if } m = (\mathsf{Sign}, m'') \\ m & \text{otherwise} \end{cases} ;$$

$res \leftarrow res + p(|(sid, pid, m)| + \eta)$; $lastSid \leftarrow sid$; send (pid, m') to $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig}')$ or $\mathrm{io}(T, \mathrm{sig}')$ (resp.)

(d) *Forward from $\mathcal{F}_{\mathrm{SIG}}$:* If recv (pid, m) on $\mathrm{io}(\mathrm{sig}', T)$ where $T \in \mathcal{T}_{\mathrm{sig}}$, $state = \mathsf{ok}$, and $|(lastSid, pid, m)| \leq res$ do: $res \leftarrow res - |(lastSid, pid, m)|$; send $(lastSid, pid, m)$ to $\mathrm{io}(\mathrm{sig}, T)$

(e) *Corruption Request:* If recv $(sid, pid, \mathsf{Corrupted}?)$ on $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig})$ where $sid \in AllSids$ do:
send $(pid, \mathsf{Corrupted}?)$ to $\mathrm{io}(E_{\mathrm{sig}}, \mathrm{sig}')$; wait for recv (pid, b) on $\mathrm{io}(\mathrm{sig}', E_{\mathrm{sig}})$;
send (sid, pid, b) to $\mathrm{io}(\mathrm{sig}, E_{\mathrm{sig}})$

If no rule above fires then produce no output.

Fig. 10. Joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}} = P^{\mathrm{JS}}_{\mathrm{sig}} \,|\, P^{\mathrm{JS}}_{\mathrm{ver}}$ for digital signature schemes, the signer's part $P^{\mathrm{JS}}_{\mathrm{sig}}$.

Realization $P^{\mathrm{JS}}_{\mathrm{ver}}(\mathcal{T}_{\mathrm{ver}}, p, q)$

I/O-tapes: in: $\mathrm{io}(T, \mathrm{sig})$ for each $T \in \mathcal{T}_{\mathrm{ver}}$, $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig})$ (enriching),
$\mathrm{io}(\mathrm{sig}', T)$ for each $T \in \mathcal{T}_{\mathrm{ver}}$, $\mathrm{io}(\mathrm{sig}', E_{\mathrm{ver}})$ (consuming)
out: $\mathrm{io}(\mathrm{sig}, T)$ for each $T \in \mathcal{T}_{\mathrm{ver}}$, $\mathrm{io}(\mathrm{sig}, E_{\mathrm{ver}})$, $\mathrm{io}(T, \mathrm{sig}')$ for each $T \in \mathcal{T}_{\mathrm{ver}}$, $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig}')$

Initialization: $pid, pid' \leftarrow \bot$; $Sids \leftarrow \emptyset$; $AllSids \leftarrow \emptyset$; $lastSid \leftarrow \bot$; $res \leftarrow 0$; $state \leftarrow \mathsf{init}$

CheckAddress: Upon first activation accept msgs of shape (sid, pid_1, pid_2, m) on all sig-tapes (i.e. on $\mathrm{io}(T, \mathrm{sig})$, $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig})$). In mode $\mathsf{Compute}$ at first record the PIDs: $pid \leftarrow pid_1$ and $pid' \leftarrow pid_2$.
On later activations accept msgs of shape (sid, pid_1, pid_2, m) on all sig-tapes and of shape (pid_1, pid_2, m) on all sig′-tapes where $pid_1 = pid$ and $pid_2 = pid'$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, pid', \mathsf{Init})$ on $\mathrm{io}(T, \mathrm{sig})$ where $T \in \mathcal{T}_{\mathrm{ver}}$, $sid \notin AllSids$ do:
$AllSids \leftarrow AllSids \cup \{sid\}$;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then
produce no output
else if $state = \mathsf{wait}$ then
$state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
$state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send $(pid, pid', \mathsf{Init})$ to $\mathrm{io}(T, \mathrm{sig}')$
else
$Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, pid', \mathsf{Inited})$ to $\mathrm{io}(\mathrm{sig}, T)$
end

(b) *Initialization Response:* If recv $(pid, pid', \mathsf{Inited})$ on $\mathrm{io}(\mathrm{sig}', T)$ where $T \in \mathcal{T}_{\mathrm{ver}}$, $state = \mathsf{wait}$ do:
$state \leftarrow \mathsf{ok}$; $Sids \leftarrow \{lastSid\}$; send $(lastSid, pid, pid', \mathsf{Inited})$ to $\mathrm{io}(\mathrm{sig}, T)$

(c) *Forward to* $\mathcal{F}_{\mathrm{SIG}}$*:* If recv (sid, pid, pid', m) on $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig})$ or $\mathrm{io}(T, \mathrm{sig})$ where $T \in \mathcal{T}_{\mathrm{ver}}$, $state = \mathsf{ok}$, $sid \in Sids$, and $m \neq \mathsf{Corrupted}?$ do:
$res \leftarrow res + p(|(sid, pid, pid', m)| + \eta)$; $lastSid \leftarrow sid$;
$m' \leftarrow \begin{cases} (\mathsf{Verify}, (sid, m''), \sigma, k) & \text{if } m = (\mathsf{Verify}, m'', \sigma, k) \\ m & \text{otherwise} \end{cases}$;
send (pid, pid', m') to $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig}')$ or $\mathrm{io}(T, \mathrm{sig}')$ (resp.)

(d) *Forward from* $\mathcal{F}_{\mathrm{SIG}}$*:* If recv $m = (pid, pid', m')$ on $\mathrm{io}(\mathrm{sig}', T)$ where $T \in \mathcal{T}_{\mathrm{ver}}$, $state = \mathsf{ok}$, and $|(lastSid, m)| \leq res$ do: $res \leftarrow res - |(lastSid, m)|$; send $(lastSid, pid, pid', m')$ to $\mathrm{io}(\mathrm{sig}, T)$

(e) *Corruption Request:* If recv $(sid, pid, pid', \mathsf{Corrupted}?)$ on $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig})$ where $sid \in AllSids$ do:
send $(pid, pid', \mathsf{Corrupted}?)$ to $\mathrm{io}(E_{\mathrm{ver}}, \mathrm{sig}')$; wait for recv (pid, pid', b) on $\mathrm{io}(\mathrm{sig}', E_{\mathrm{ver}})$;
send (sid, pid, pid', b) to $\mathrm{io}(\mathrm{sig}, E_{\mathrm{ver}})$

If no rule above fires then produce no output.

Fig. 11. Joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}} = P^{\mathrm{JS}}_{\mathrm{sig}} \,|\, P^{\mathrm{JS}}_{\mathrm{ver}}$ for digital signature schemes, the verifier's part $P^{\mathrm{JS}}_{\mathrm{ver}}$.

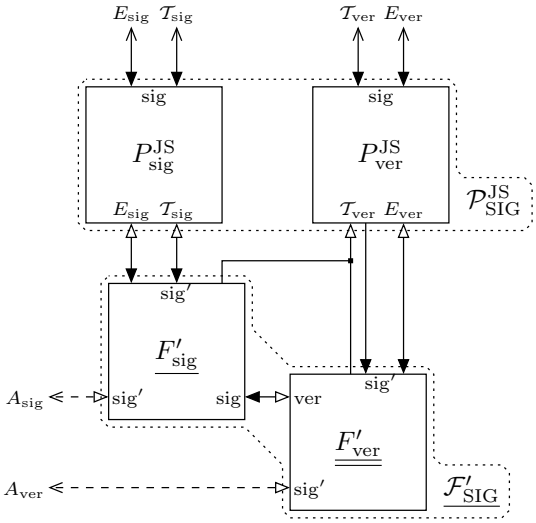

Fig. 12. Graphical representation of the joint state implementation $\mathcal{P}^{\text{JS}}_{\text{SIG}} = P^{\text{JS}}_{\text{sig}} \mid P^{\text{JS}}_{\text{ver}}$ and its connection to $!\underline{\mathcal{F}'_{\text{SIG}}}$. See Figure 4 for a legend.

in session *sid* but has not completed it yet, i.e. the environment/adversary has not yet sent the algorithms and a public key, then, if the environment sends another initialization request in another session sid' for the same party the question is how to deal with this situation. The joint state realization cannot further process this request because it already waits for an answer from the functionality in the first request. (Note that for both requests the same functionality is used in the joint state world). Conversely, the joint state realization also cannot simply ignore this particular request by the environment because a simulator could not reproduce this behavior. Therefore, we define the joint state realization $\mathcal{P}^{\text{JS}}_{\text{PKE}}$ to enter and stay in an error state in which any further messages (except for (Corrupted?) from the environment) are ignored. In other words, the joint state realization stops any further activity. This is not a real limitation as it only forces the environment/adversary to complete initialization requests at once, which in any realistic implementation of the public-key functionality is done anyway because initialization requests are answered immediately—a key pair is generated and the public key is returned right away.

A more detailed description of $\mathcal{P}^{\text{JS}}_{\text{SIG}}$ is given next.

There will be one copy of $P^{\text{JS}}_{\text{sig}} = P^{\text{JS}}_{\text{sig}}(\mathcal{T}_{\text{sig}}, p, q)$ for each signer. We denote the copy of a party (with PID) *pid* by $P^{\text{JS}}_{\text{sig}}[pid]$. Additionally, there will be one copy of $P^{\text{JS}}_{\text{ver}} = P^{\text{JS}}_{\text{ver}}(\mathcal{T}_{\text{ver}}, p, q)$ for each signer *pid* and verifier pid' which we denote by $P^{\text{JS}}_{\text{ver}}[pid, pid']$.

A signer *pid* in session *sid* sends messages of shape (sid, pid, m) to the copy $P^{\text{JS}}_{\text{sig}}[pid]$. A verifier pid' in session *sid* sends messages of shape (sid, pid, pid', m) to the copy $P^{\text{JS}}_{\text{ver}}[pid, pid']$ when he wants to verify messages of party *pid*. No party other than *pid* will communicate with $P^{\text{JS}}_{\text{sig}}[pid]$, so it is like a local procedure and we call party *pid* the *owner* of $P^{\text{JS}}_{\text{sig}}[pid]$. Similarly, no party other than pid' will communicate with $P^{\text{JS}}_{\text{ver}}[pid, pid']$ and party pid' is called the *owner* of $P^{\text{JS}}_{\text{ver}}[pid, pid']$.

A signer *pid* has to register in each session *sid* with $P^{\text{JS}}_{\text{sig}}[pid]$, i.e. send $(sid, pid, \mathsf{Init})$. However, only the first time $P^{\text{JS}}_{\text{sig}}[pid]$ creates a copy of $\underline{F'_{\text{sig}}}$ by stripping off the SID and forwarding the message (pid, Init). To refer to the created copy it is denoted by $F'_{\text{sig}}[pid]$. The polynomial q is used to forbid sessions with *long* SIDs ($sid > q(\eta)$). See Figure 10 (a) and (b). If the same signer *pid* sends a second initialization request before the first one has been completed, $P^{\text{JS}}_{\text{sig}}[pid]$ will enter an error state and ignore all messages send from the signer *pid* or from $F'_{\text{sig}}[pid]$ (except for (Corrupted?) from the environment).

Analogously, a verifier pid' registers with $P^{\text{JS}}_{\text{ver}}[pid, pid']$. Here, only one copy of $\underline{\underline{F'_{\text{ver}}}}$, denoted by $F'_{\text{ver}}[pid, pid']$, is created. See Figure 11 (a) and (b).

After initialization, every message is forwarded by $P^{\text{JS}}_{\text{sig}}[pid]$ and $P^{\text{JS}}_{\text{ver}}[pid, pid']$ to the instances $F'_{\text{sig}}[pid]$ and $F'_{\text{ver}}[pid, pid']$ (respectively) by stripping off the SID. A sign request $(sid, pid, \mathsf{Sign}, m)$

is forwarded as $(pid, \mathsf{Sign}, (sid, m))$. If a verification request $(sid, pid, pid', \mathsf{Verify}, m, \sigma, k)$ is received then the message $(pid, pid', \mathsf{Verify}, (sid, m), \sigma, k)$ is forwarded.

Since the system has to be well-formed the tapes from $\underline{\mathcal{F}'_{\mathrm{SIG}}}$ to $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}$ have to be consuming. Hence, forwarding messages from $\underline{\mathcal{F}'_{\mathrm{SIG}}}$ to a party can not be done arbitrarily. Therefore, $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}$ records the length of the messages sent to $\underline{\mathcal{F}'_{\mathrm{SIG}}}$ and only forwards polynomially many input from $\underline{\mathcal{F}'_{\mathrm{SIG}}}$. However, note that by the definition of $\mathcal{F}_{\mathrm{SIG}}(p)$ every message that is output by $\mathcal{F}_{\mathrm{SIG}}(p)$ has at most length $p(|m|+\eta)$ where m is the input that $\mathcal{F}_{\mathrm{SIG}}(p)$ received before. Thus, when $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}(p,q)$ and $\underline{\mathcal{F}'_{\mathrm{SIG}}(p)}$ are composed, $\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}(p,q)$ will always be able to forward messages from $\underline{\mathcal{F}'_{\mathrm{SIG}}(p)}$.

Next, we state and prove the joint state theorem for digital signatures. Note that since in this theorem, we quantify over all polynomials p, the theorem can be applied iteratively as described in Section 3. Furthermore, the proof reveals that the theorem even holds for unbounded environments and perfect indistinguishability, i.e., there exists a simulator $\mathcal{S}$ such that for all (unbounded) environments $\mathcal{E}$ it holds $\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}} \,|\, !\underline{\mathcal{F}'_{\mathrm{SIG}}}(1^\eta, a) \rightsquigarrow 1] = \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, !\underline{\underline{\mathcal{F}_{\mathrm{SIG}}}}(1^\eta, a) \rightsquigarrow 1]$ for all $\eta \in \mathbb{N}, a \in \{0,1\}^*$.

Theorem 6. *For all polynomials p and q and disjoint sets of names of tapes $\mathcal{T}_{\mathrm{sig}}$ and $\mathcal{T}_{\mathrm{ver}}$ there is a polynomial p' such that*

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p, q) \,|\, !\underline{\mathcal{F}'_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p)} \leq^{SS} !\underline{\underline{\mathcal{F}_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p')}}$$

where $\mathcal{F}'_{\mathrm{SIG}}$ is obtained from $\mathcal{F}_{\mathrm{SIG}}$ by renaming all tapes by replacing sig *by* sig$'$ *in the tape name.*

Proof. Below, we define a simulator $\mathcal{S}$ such that the joint state world (JS world), i.e. $\mathcal{E} \,|\, \mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}} \,|\, !\underline{\mathcal{F}'_{\mathrm{SIG}}}$, is perfectly indistinguishable from the ideal world, i.e. $\mathcal{E} \,|\, \mathcal{S} \,|\, !\underline{\underline{\mathcal{F}_{\mathrm{SIG}}}}$, for every (unbounded) environment $\mathcal{E}$. When the environment presents algorithms s, v and a key k then the simulator $\mathcal{S}$ forwards s_{sid}, v_{sid} and k. The definition of s_{sid} and v_{sid} (which is given below) will yield the definition of p'.

Let $\eta \in \mathbb{N}$, sid be an SID with $|sid| \leq q(\eta)$ and s and v be descriptions of algorithms with $|s| \leq p(\eta)$ and $|v| \leq p(\eta)$. Depending on η, sid, s, v and p, we define the algorithms s_{sid} and v_{sid} as follows:

- Algorithm $s_{sid}(m)$ computes $\sigma \leftarrow s((sid, m))$ and counts the steps needed. If these are at most $p(|(sid, m)| + \eta)$ then return σ else enter an infinite loop.
- Algorithm $v_{sid}(m, \sigma, k)$ computes $b \leftarrow v((sid, m), \sigma, k)$ and counts the steps needed. If these are at most $p(|(sid, m)| + |\sigma| + \eta)$ then return b else enter an infinite loop.

Since the length of the description of s and v and the length of sid is polynomially bounded by η, the length of the description of s_{sid} and v_{sid} is polynomially bounded by η. Moreover, if the runtime of $s((sid, m))$ and $v((sid, m), \sigma, k)$ is bounded by $p(|(sid, m)| + \eta)$ and $p(|(sid, m)| + |\sigma| + \eta)$ (resp.) then the runtime of $s_{sid}(m)$ and $v_{sid}(m, \sigma, k')$ (except when they enter an infinite loop) is polynomial in $|m| + \eta$ and $|m| + |\sigma| + \eta$, respectively. Thus, we find a polynomial p' such that

1. for all $\eta \in \mathbb{N}$ we have that if $|s| \leq p(\eta)$ and $|v| \leq p(\eta)$ then $|s_{sid}| \leq p'(\eta)$ and $|v_{sid}| \leq p'(\eta)$,
2. for all messages m and $\eta \in \mathbb{N}$ we have that the computation of $s((sid, m))$ exceeds $p(|(sid, m)| + \eta)$ steps iff the computation of $s_{sid}(m)$ exceeds $p'(|m| + \eta)$ steps, and
3. for all messages m, signature strings σ, keys k and $\eta \in \mathbb{N}$ we have that the computation of $v((sid, m), \sigma, k)$ exceeds $p(|(sid, m)| + |\sigma| + \eta)$ steps iff the computation of $v_{sid}(m, \sigma, k)$ exceeds $p'(|m| + |\sigma| + \eta)$ steps.

Let $\mathcal{P} = \mathcal{P}^{\mathrm{JS}}_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p, q) \,|\, !\underline{\mathcal{F}'_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p)}$ and $\mathcal{F} = !\underline{\underline{\mathcal{F}_{\mathrm{SIG}}(\mathcal{T}_{\mathrm{sig}}, \mathcal{T}_{\mathrm{ver}}, p')}}$. We define a simulator $!\mathcal{S} \in \mathrm{Sim}^{\mathcal{P}}_{\mathbf{S}}(\mathcal{F})$ such that $\mathcal{P}$ and $!\mathcal{S} \,|\, \mathcal{F}$ are perfectly indistinguishable.

There will be one copy of $\mathcal{S}$ – namely $\mathcal{S}[pid]$ – for each PID pid of a signer. The copy $\mathcal{S}[pid]$ interacts with copies of $\underline{\underline{F_{\mathrm{sig}}}}$ and $\underline{\underline{F_{\mathrm{ver}}}}$ which are denoted by $F_{\mathrm{sig}}[sid, pid]$ and $F_{\mathrm{ver}}[sid, pid, pid']$ (resp.) according to their SID and PIDs.

Because of the initialization messages $\mathcal{S}[pid]$ has full information of which copies $F_{\mathrm{sig}}[sid, pid]$ and $F_{\mathrm{ver}}[sid, pid, pid']$ have been initialized.

Upon the first initialization message of $F_{\mathrm{sig}}[sid, pid]$ where sid is a *short* SID (i.e. $sid \leq q(\eta)$), $\mathcal{S}[pid]$ forwards it to the adversary and waits for receiving algorithms s and v and a key k. Then, $\mathcal{S}[pid]$ provides the algorithms s_{sid} and v_{sid} and key k to all copies $F_{\mathrm{sig}}[sid, pid]$ that have been initialized

with a short SID sid. The algorithms s_{sid} and v_{sid} are defined as above. If $\mathcal{S}[pid]$ receives another initialization message from $F_{\text{sig}}[sid', pid]$ for some other sid' before it received algorithms and completed initialization for the first SID sid then $\mathcal{S}[pid]$ enters the error state, i.e. sets variable *state* to error. From then on, no messages are forwarded by $\mathcal{S}[pid]$ except for corrupt messages. In particular, if $\mathcal{S}[pid]$ is in the error state then no instance $F_{\text{sig}}[sid, pid]$ completes initialization, i.e. enters the state ok. No instance $F_{\text{sig}}[sid, pid]$ with a *long* SID sid (i.e. $sid > q(\eta)$) does complete initialization and is therefore "blocked" throughout the rest of the computation. However, $F_{\text{sig}}[sid, pid]$ will respond to corruption requests of the environment (message (Corrupted?) from E_{sig}). Since the joint state realization forwards the corruption requests even for long SIDs, $\mathcal{S}[pid]$ needs to corrupt all copies $F_{\text{sig}}[sid, pid]$ even the ones with a long SID. But, $\mathcal{S}[pid]$ will not forward any further messages from the adversary to $F_{\text{sig}}[sid, pid]$ and vice versa once it is corrupted since this is exactly what happens with the joint state realization.

Upon corruption of the signer pid, $\mathcal{S}[pid]$ corrupts all (existing) copies $F_{\text{sig}}[sid, pid]$ and from now on, if $F_{\text{sig}}[sid, pid]$ is initialized for a new SID sid, that copy is directly corrupted. Similarly, upon corruption of a verifier pid' that verifies messages of a signer pid, $\mathcal{S}[pid]$ corrupts all (existing) copies $F_{\text{ver}}[sid, pid, pid']$ and from now on, if $F_{\text{ver}}[sid, pid, pid']$ is initialized for a new SID sid, that copy is directly corrupted.

Other messages from $F_{\text{sig}}[sid, pid]$ or $F_{\text{ver}}[sid, pid, pid']$ to the adversary are forwarded by $\mathcal{S}[pid]$ by stripping of the SID. If the message is a forwarded sign or verify request with a message m then it is forwarded with the message (sid, m) instead of m. There is one difficulty with forwarding the response of the adversary to a copy of $\underline{\underline{F_{\text{sig}}}}$ or $\underline{\underline{F_{\text{ver}}}}$. $\mathcal{S}[pid]$ has to know the SID. Therefore, $\mathcal{S}[pid]$ records the last SID that was used when sending messages to the adversary.

A formal definition of the simulator

$$!\mathcal{S} = \mathcal{S}^{\text{JS}}_{\text{SIG}}(p, q) = !S^{\text{JS}}_{\text{sig}}(p, q) \mid !S^{\text{JS}}_{\text{ver}}(q)$$

is given in Figure 13 and 14. A graphical representation of $\mathcal{S}^{\text{JS}}_{\text{SIG}}$ and its connection to $!\underline{\underline{\mathcal{F}_{\text{SIG}}}}$ is pictured in Figure 15.

It remains to show that $\text{Prob}[\mathcal{E} \mid \mathcal{P}^{\text{JS}}_{\text{SIG}}(p, q) \mid !\underline{\mathcal{F}'_{\text{SIG}}(p)}(1^\eta, a) \rightsquigarrow 1] = \text{Prob}[\mathcal{E} \mid !\mathcal{S} \mid !\underline{\underline{\mathcal{F}_{\text{SIG}}(p')}}(1^\eta, a)]$ for every (unbounded) environment $\mathcal{E}$.

At first consider the behavior upon initialization in the JS world compared to the ideal world. Note that $P^{\text{JS}}_{\text{sig}}[pid]$ enters the error state, i.e. sets *state* to the value error, if and only if $S^{\text{JS}}_{\text{sig}}[pid]$ enters the error state. From then on, all messages (except Corrupted? requests) send to $P^{\text{JS}}_{\text{sig}}[pid]$ are ignored and $P^{\text{JS}}_{\text{sig}}[pid]$ produces no output. The same happens in the ideal world. No instance $F_{\text{sig}}[sid, pid]$ has set *state* to ok and will never do so because $S^{\text{JS}}_{\text{sig}}[pid]$ is in the error state and will never send Inited. Hence, upon all messages $F_{\text{sig}}[sid, pid]$ will either produce no output or send some message to $S^{\text{JS}}_{\text{sig}}[pid]$ which then will produce no output because it is in the error state.

Now, consider Corrupted? requests. At first note that the variable *AllSids* in $P^{\text{JS}}_{\text{sig}}[pid]$ always contains the same SIDs as the variable *AllSids* in $S^{\text{JS}}_{\text{sig}}[pid]$. $P^{\text{JS}}_{\text{sig}}[pid]$ forwards all $(sid, pid, \text{Corrupted?})$ messages with $sid \in AllSids$ (even in the error state) whereon $F'_{\text{sig}}[pid]$ will reply with true or false depending on whether it is corrupted or not. This reply is again forwarded by $P^{\text{JS}}_{\text{sig}}[pid]$. In the ideal world, $(sid, pid, \text{Corrupted?})$ is ignored by $F_{\text{sig}}[sid, pid]$ if no prior Init message was sent to $F_{\text{sig}}[sid, pid]$ but this means that $sid \notin AllSids$ of $S^{\text{JS}}_{\text{sig}}[pid]$. In the JS world $F'_{\text{sig}}[pid]$ is corrupted iff for all $sid \in AllSids$ $F_{\text{sig}}[sid, pid]$ is corrupted. Thus, the behavior upon corruption and Corrupted? requests is the same in both worlds.

Similarly, one can show that the behavior of $P^{\text{JS}}_{\text{ver}}[pid, pid']$ and $F'_{\text{ver}}[pid, pid']$ does not differ from the behavior of $F_{\text{ver}}[sid, pid, pid']$ and $S^{\text{JS}}_{\text{ver}}[pid, pid']$ upon initialization, corruption and Corrupted? requests.

By definition, $S^{\text{JS}}_{\text{sig}}[pid]$ accepts s, v, k iff $F'_{\text{sig}}[pid]$ in the JS world accepts s, v, k. Upon key generation, in the JS world the $F'_{\text{sig}}[pid]$ accepts the algorithms s, v and key k if only if $|s| \leq p(\eta)$, $|v| \leq p(\eta)$ and $|k| \leq p(\eta)$. By the definition of p' we have that if $S^{\text{JS}}_{\text{sig}}[pid]$ accepts s, v, k then $F_{\text{sig}}[sid, pid]$ accepts s_{sid}, v_{sid}, k. Therefore, upon key generation, there is exactly the same behavior in the JS world and the ideal world.

We conclude that there is no difference between the JS world and the ideal world according to the behavior of initialization, corruption and key generation.

Upon signature generation in session sid, in the joint state world $F'_{\text{sig}}[pid]$ computes $\sigma \leftarrow s((sid, m))$ with at most $p(|(sid, m)| + \eta)$ steps while in the ideal world $F_{\text{sig}}[sid, pid]$ computes $\sigma \leftarrow s_{sid}(m)$ with at

Simulator $S^{\mathrm{JS}}_{\mathrm{sig}}(p,q)$

net-tapes: in: $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig}')$, $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$, $\mathrm{net}(\mathrm{ver},\mathrm{sig})$ (enriching)
out: $\mathrm{net}(\mathrm{sig}',A_{\mathrm{sig}})$, $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$, $\mathrm{net}(\mathrm{sig},\mathrm{ver})$

Initialization: $pid, lastSid, s, v, k \leftarrow \perp$; $state \leftarrow \mathsf{init}$; $nokey \leftarrow \mathsf{true}$; $corrupted \leftarrow \mathsf{false}$;
$Sids, AllSids, KeySids \leftarrow \emptyset$

CheckAddress: Upon first activation accept messages of shape (sid,pid',m) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$ and of shape (pid',m) on $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig}')$. In mode $\mathsf{Compute}$ at first record the party : $pid \leftarrow pid'$.
On later activations accept messages of shape (sid,pid',m) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$ and of shape (pid',m) on $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig}')$ where $pid' = pid$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (sid,pid,Init) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$ do: $AllSids \leftarrow AllSids \cup \{sid\}$;
if *corrupted* then
send $(sid,pid,\mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$; wait for recv $(sid,pid,\mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$
end ;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then produce no output
else if $state = \mathsf{wait}$ then $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
if *nokey* then $KeySids \leftarrow KeySids \cup \{sid\}$
else send $(sid,pid,\mathsf{AlgorithmsAndKey},s_{sid},v_{sid},k)$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$ [a] ;
wait for recv (sid,pid,Ack) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$
end ;
$state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid,Init) to $\mathrm{net}(\mathrm{sig}',A_{\mathrm{sig}})$
else $Sids \leftarrow Sids \cup \{sid\}$; send $(sid,pid,\mathsf{AlgorithmsAndKey},s_{sid},v_{sid},k)$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$;
wait for recv (sid,pid,Ack) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$; send $(sid,pid,\mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$
end

(b) *Initialization Response:* If recv (pid,Inited) on $\mathrm{net}(A'_{\mathrm{sig}},\mathrm{sig})$, $state = \mathsf{wait}$, and not *nokey* do:
$state \leftarrow \mathsf{ok}$; send $(lastSid,pid,\mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$

(c) *Key Gen.:* If recv $(pid,\mathsf{AlgorithmsAndKey},e',d',k')$ on $\mathrm{net}(A'_{\mathrm{sig}},\mathrm{sig})$, *nokey*, and $|s'|,|v'|,|k'| \le p(\eta)$ do:
$(s,v,k) \leftarrow (s',v',k')$; $nokey \leftarrow \mathsf{false}$;
for all $sid \in KeySids$ do
send $(sid,pid,\mathsf{AlgorithmsAndKey},s_{sid},v_{sid},k)$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$;
wait for recv (sid,pid,Ack) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$
end ;
send (pid,Ack) to $\mathrm{net}(\mathrm{sig}',A_{\mathrm{sig}})$

(d) *Corruption:* If recv $(pid,\mathsf{Corrupt})$ on $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig}')$, $state \neq \mathsf{init}$, and not *corrupted* do:
for all $sid \in AllSids$ do
send $(sid,pid,\mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$; wait for recv $(sid,pid,\mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$
end ;
$corrupted \leftarrow \mathsf{true}$; send $(pid,\mathsf{Corrupted})$ to $\mathrm{net}(\mathrm{sig}',A_{\mathrm{sig}})$

(e) *Forward to A:* If recv (sid,pid,m) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$, $m \neq \mathsf{Init}$, $state \neq \mathsf{error}$, and $sid \in Sids$ do:
$$m' \leftarrow \begin{cases} (\mathsf{Received},(\mathsf{Sign},(sid,m'')),T) & \text{if } m = (\mathsf{Received},(\mathsf{Sign},m''),T) \\ m & \text{otherwise} \end{cases} ;$$
$lastSid \leftarrow sid$; send (pid,m') to $\mathrm{net}(\mathrm{sig}',A_{\mathrm{sig}})$

(f) *Forward from A:* If recv (pid,Send,m,T) on $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig}')$, and $state \neq \mathsf{error}$ do:
send $(lastSid,pid,\mathsf{Send},m,T)$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$

(g) If recv $(pid,pid',\mathsf{SendAlgAndKey},sid)$ on $\mathrm{net}(\mathrm{ver},\mathrm{sig})$ do:
if *nokey* then $KeySids \leftarrow KeySids \cup \{sid\}$
else send $(sid,pid,\mathsf{AlgorithmsAndKey},s_{sid},v_{sid},k)$ to $\mathrm{net}(A_{\mathrm{sig}},\mathrm{sig})$;
wait for recv (sid,pid,Ack) on $\mathrm{net}(\mathrm{sig},A_{\mathrm{sig}})$
end ;
send (pid,pid',Ack) to $\mathrm{net}(\mathrm{sig},\mathrm{ver})$

If no rule above fires then produce no output.

[a] where s_{sid} and v_{sid} are basically defined by $s_{sid}(m)$: return $s((sid,m))$ and $v_{sid}(m,\sigma,k')$: return $v((sid,m),\sigma,k')$ (see formal definitions in the proof of Theorem 6)

Fig. 13. Simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{SIG}} = !S^{\mathrm{JS}}_{\mathrm{sig}} \mid !S^{\mathrm{JS}}_{\mathrm{ver}}$ for the proof of the joint state theorem for digital signatures, the signer's part $S^{\mathrm{JS}}_{\mathrm{sig}}$.

Simulator $S^{\mathrm{JS}}_{\mathrm{ver}}(q)$

net-tapes: in: $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig}')$, $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ (enriching), $\mathrm{net}(\mathrm{sig}, \mathrm{ver})$ (consuming)
out: $\mathrm{net}(\mathrm{sig}', A_{\mathrm{ver}})$, $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$, $\mathrm{net}(\mathrm{ver}, \mathrm{sig})$

Initialization: $pid, pid', lastSid \leftarrow \bot$; $state \leftarrow \mathsf{init}$; $corrupted \leftarrow \mathsf{false}$; $Sids \leftarrow \emptyset$; $AllSids \leftarrow \emptyset$

CheckAddress: Upon first activation accept messages of shape (sid, pid_1, pid_2, m) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ and of shape (pid_1, pid_2, m) on $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig}')$. In mode $\mathsf{Compute}$ at first record PIDs: $pid \leftarrow pid_1$, $pid' \leftarrow pid_2$. On later activations accept messages of shape (sid, pid_1, pid_2, m) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ and of shape (pid_1, pid_2, m) on $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig}')$ where $pid_1 = pid$, $pid_2 = pid'$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, pid', \mathsf{Init})$ on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$ do: $AllSids \leftarrow AllSids \cup \{sid\}$;
if $corrupted$ then
send $(sid, pid, pid', \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$; wait for recv $(sid, pid, pid', \mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$
end ;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then produce no output
else if $state = \mathsf{wait}$ then $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
send $(pid, pid', \mathsf{SendAlgAndKey}, sid)$ to $\mathrm{net}(\mathrm{ver}, \mathrm{sig})$; wait for recv $(pid, pid', \mathsf{Ack})$ on $\mathrm{net}(\mathrm{sig}, \mathrm{ver})$;
$state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid, Init) to $\mathrm{net}(\mathrm{sig}', A_{\mathrm{ver}})$
else send $(pid, pid', \mathsf{SendAlgAndKey}, sid)$ to $\mathrm{net}(\mathrm{ver}, \mathrm{sig})$; wait for recv $(pid, pid', \mathsf{Ack})$ on $\mathrm{net}(\mathrm{sig}, \mathrm{ver})$;
$Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, pid', \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$
end

(b) *Initialization Response:* If recv $(pid, pid', \mathsf{Inited})$ on $\mathrm{net}(A'_{\mathrm{ver}}, \mathrm{sig})$, and $state = \mathsf{wait}$ do:
$state \leftarrow \mathsf{ok}$; send $(lastSid, pid, pid', \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$

(c) *Corruption:* If recv $(pid, pid', \mathsf{Corrupt})$ on $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig}')$, $state \neq \mathsf{init}$, and not $corrupted$ do:
for all $sid \in AllSids$ do
send $(sid, pid, pid', \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$; wait for recv $(sid, pid, pid', \mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$
end ;
$corrupted \leftarrow \mathsf{true}$; send $(pid, pid', \mathsf{Corrupted})$ to $\mathrm{net}(\mathrm{sig}', A_{\mathrm{ver}})$

(d) *Forward to A:* If recv (sid, pid, pid', m) on $\mathrm{net}(\mathrm{sig}, A_{\mathrm{ver}})$, $m \neq \mathsf{Init}$, $state \neq \mathsf{error}$, and $sid \in Sids$ do:

$$m' \leftarrow \begin{cases} (\mathsf{Received}, (\mathsf{Verify}, (sid, m''), \sigma, k'), T) & \text{if } m = (\mathsf{Received}, (\mathsf{Verify}, m'', \sigma, k'), T) \\ m & \text{otherwise ;} \end{cases}$$

$lastSid \leftarrow sid$; send (pid, pid', m') to $\mathrm{net}(\mathrm{sig}', A_{\mathrm{ver}})$

(e) *Forward from A:* If recv $(pid, pid', \mathsf{Send}, m, T)$ on $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig}')$, and $state \neq \mathsf{error}$ do:
send $(lastSid, pid, pid', \mathsf{Send}, m, T)$ to $\mathrm{net}(A_{\mathrm{ver}}, \mathrm{sig})$

If no rule above fires then produce no output.

Fig. 14. Simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{SIG}} = !S^{\mathrm{JS}}_{\mathrm{sig}} \mid !S^{\mathrm{JS}}_{\mathrm{ver}}$ for the proof of the joint state theorem for digital signatures, the verifier's part $S^{\mathrm{JS}}_{\mathrm{ver}}$.

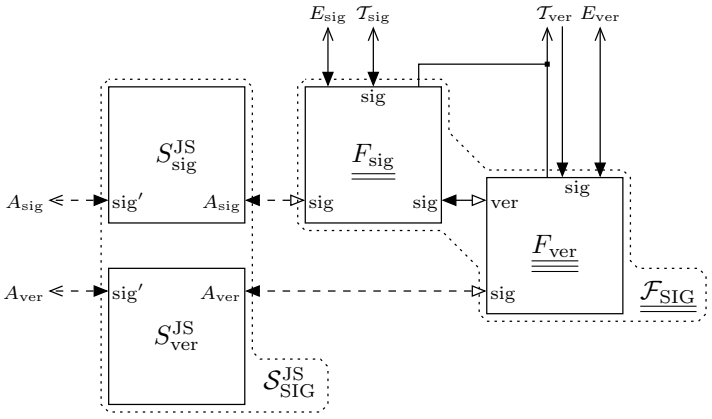

Fig. 15. Graphical representation of the simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{SIG}} = !S^{\mathrm{JS}}_{\mathrm{sig}} \mid !S^{\mathrm{JS}}_{\mathrm{ver}}$ for $!\underline{\underline{\mathcal{F}_{\mathrm{SIG}}}} = !\underline{\underline{F_{\mathrm{sig}}}} \mid !\underline{\underline{F_{\mathrm{ver}}}}$. See Figure 4 for a legend.

most $p'(|m| + \eta)$ steps. By the definition of p' and s_{sid}, the distribution of the output σ in the JS world is equal to the distribution of the output σ in the ideal world. The same argumentation holds for the computation of v and v_{sid}, respectively. Thus, the output upon signature generation in the JS world is indistinguishable from the one in the ideal world.

The same holds true upon verification of (m, σ, k') in session sid, because there is a σ such that $((sid, m), \sigma) \in H_{pid}$ iff there is a σ such that $(m, \sigma) \in H_{sid,pid}$ where H_{pid} is the H in the copy $F'_{\text{sig}}[pid]$ in the JS world and $H_{sid,pid}$ is the H in the copy $F_{\text{sig}}[sid, pid]$ in the ideal world. □

5.4 Comparison with other Formulations

We now compare our formulation of the digital signature functionality to other formulations in the literature.

As mentioned in the introduction, most other formulations of digital signature functionalities are defined in a non-local way [5, 14, 13, 1], 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 [7] 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 EU-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 BPW model [28, 3] which does not explicitly talk about copies of protocols/functionalities.

One problem of the formulation in [7] is that it does not seem to have any reasonable joint state realization, unlike claimed in [7] without providing a joint state realization or a proof: The signature functionality in [7] uses only the signing and verification algorithms s and v, but no key k. It is argued that the key is incorporated in the algorithm v. Thus, to verify a message-signature pair (m, σ) a verification algorithm v' has to be presented and in the functionality it is then checked if $v' = v$. If $v' \neq v$, the algorithm v' is run on (m, σ), and the result of this algorithm is returned. However, as argued next, failing to make the distinction between the verification algorithm and the verification 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 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 s and v. It then requests to verify the message-signature pair (m, σ), where m is not of the form (sid, m'), with the verification algorithm v' where $v' \neq v$ is defined as follows: $v'(m, \sigma)$ outputs 1 if the message m is of shape (sid, m'), it outputs 0 otherwise. If E obtains $(\mathsf{Verified}, 1)$, it outputs 1, and 0 otherwise. It is easy to see that if E communicates with the joint state realization it will always output 1 since this realization forwards (sid, m) to the digital signature functionality. Since $v' \neq v$, the functionality will call $v'((sid, m), \sigma)$ and so 1 is returned. On the other hand, in the ideal world where E communicates directly with a copy of the digital signature functionality, E will always output 0 since this copy runs $v'(m, \sigma)$.

We note that for *non-local* formulations of digital signature functionalities, the above problem does not occur.

Another problem in Canetti's formulation of the digital signature functionality in [7] is that the signing algorithm s is allowed to preserve some state, i.e. the signature values may depend on the messages signed so far, while in our formulation s is stateless. It is easy to prove that with a stateful s, 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 parameterized by certain classes of signing/verification algorithms would be more complex.

Another difference of our formulation of the digital signature functionality compared to other formulations is that in our formulation we model the following realistic scenario: It is possible for a verifier, some party V, to request the functionality that belongs to some other party, say S, to verify a message-signature pair although no key generation request by party S has been sent yet. As illustrated in Section 5.2, this is a natural property that real signature schemes have. (Note that the verifier V has to provide the verification key when invoking the functionality. In particular, V can invoke this functionality even before S generated a key.) The realizations presented in [7, 5, 1, 14, 13] are not very precise on that point. They assume that verification is not possible until the signer has generated its keys.

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 Public-Key Encryption

In this section, we present our ideal public-key encryption functionality $\mathcal{F}_{\text{PKE}}$ with local-computation (Section 6.1), show equivalence to CCA-security (Section 6.2), and provide a joint state implementation 6.3. A comparison with other formulations of public-key encryption is given in Section 6.4.

6.1 Ideal Public-Key Encryption Functionality

We present our ideal functionality $\mathcal{F}_{\text{PKE}}$ for public-key encryption with local computation. This functionality is in the spirit of the one proposed by Canetti in [7] in that other than providing an encryption and decryption algorithm as well as a public-key (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 6.4).

We define the encryption functionality in a general setting that allows the encryption to leak information of the plaintext as specified by a leakage algorithm.

Definition 3. *A family of probabilistic polynomial time (PPT) algorithms $(L_\eta)_{\eta\in\mathbb{N}}$ which take as input a bit string and return a bit string or the special error symbol $\perp$ is called a* family of (probabilistic) leakage algorithms *(or simply* leakage*). We require that for all $\eta \in \mathbb{N}$ if $\text{Prob}[L_\eta(x) = \perp] > 0$ then $\text{Prob}[L_\eta(x) = \perp] = 1$ for all $x \in \{0,1\}^*$. The set of bit strings $\text{dom}(L_\eta) = \{x \in \{0,1\}^* \mid \text{Prob}[L_\eta(x) = \perp] = 0\}$ is called the* domain *of L_η for security parameter $\eta \in \mathbb{N}$.*

Often, it is required or one wants to guarantee that encryption does not leak more than the length of the plaintext, e.g., [29, 18]. This special case is modeled by an appropriate family of leakage algorithms which leaks the length of a plaintext, e.g. one of the following four families of leakage algorithms (i) $(L^0)_{\eta\in\mathbb{N}}$ with $\{L^0(x)\colon \text{return } 0^{|x|}\}$, (ii) $(L^0_\eta)_{\eta\in\mathbb{N}}$ with $\{L^0_\eta(x)\colon \text{ if } |x| < \eta \text{ then return } \perp \text{ else return } 0^{|x|}\}$, (iii) $(L^{|\cdot|})_{\eta\in\mathbb{N}}$ with $\{L^{|\cdot|}(x)\colon \text{return } r \leftarrow^{\text{R}} \{0,1\}^{|x|}\}$, or (iv) $(L^{|\cdot|}_\eta)_{\eta\in\mathbb{N}}$ with $\{L^{|\cdot|}_\eta(x)\colon \text{ if } |x| < \eta \text{ then return } \perp \text{ else return } r \leftarrow^{\text{R}} \{0,1\}^{|x|}\}$. The domain of L^0_η and $L^{|\cdot|}_\eta$ is $\bigcup_{n\geq\eta}\{0,1\}^n$ and the domain of L^0 and L^η is $\{0,1\}^*$. The amount of information leaked by $(L^0)_{\eta\in\mathbb{N}}$, $(L^0_\eta)_{\eta\in\mathbb{N}}$, $(L^{|\cdot|})_{\eta\in\mathbb{N}}$, and $(L^{|\cdot|}_\eta)_{\eta\in\mathbb{N}}$ is the same, namely the length of the message, but $(L^{|\cdot|}_\eta)_{\eta\in\mathbb{N}}$ has another property which we call *high entropy*

and is defined next. Families of leakage algorithms with high entropy have advantages when the ideal functionality is used for reasoning about security properties of protocols (see below, where we summarize properties of $\mathcal{F}_{\text{PKE}}$) and are needed for realizing replayable public-key encryption (see Section 7).

Definition 4. *A family of leakage algorithms* $(L_\eta)_{\eta\in\mathbb{N}}$ *has* high entropy *if* $\text{Prob}[x' \leftarrow L_\eta(x), y' \leftarrow L_\eta(y) : x' = y' \neq \bot]$ *is negligible (as a function in* η*) for all* $x, y \in \{0,1\}^*$.

Most often we consider *length preserving* families of leakage algorithms which have the property that $\text{Prob}[|L_\eta(x)| = |x|] = 1$ for all $\eta \in \mathbb{N}$ and all $x \in \text{dom}(L_\eta)$. For example, $(L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$ as defined above is length preserving and has high-entropy while $(L^0)_{\eta\in\mathbb{N}}$, $(L_\eta^0)_{\eta\in\mathbb{N}}$, and $(L^{|\cdot|})_{\eta\in\mathbb{N}}$ are only length preserving but do not have high-entropy. The leakage $(L^{|\cdot|})_{\eta\in\mathbb{N}}$ does not have high-entropy because there are collisions with non-negligible probability for short messages.

Let $(L_\eta)_{\eta\in\mathbb{N}}$ and $(L'_\eta)_{\eta\in\mathbb{N}}$ be families of leakage algorithms. We say that $(L'_\eta)_{\eta\in\mathbb{N}}$ *leaks at least as much as* $(L_\eta)_{\eta\in\mathbb{N}}$, denoted $(L_\eta)_{\eta\in\mathbb{N}} \leq (L'_\eta)_{\eta\in\mathbb{N}}$, if they have the same domain, i.e. $\text{dom}(L_\eta) = \text{dom}(L'_\eta)$ for all $\eta \in \mathbb{N}$, and it is possible to compute (in polynomial time) $L_\eta(x)$ from $L'_\eta(x)$, i.e. if there exists a family of probabilistic polynomial time algorithms $(T_\eta)_{\eta\in\mathbb{N}}$ such that the distribution of $T_\eta(L'_\eta(x))$ equals the distribution of $L_\eta(x)$ for all $\eta \in \mathbb{N}$ and $x \in \text{dom}(L_\eta)$. For example, it is easy to see that $(L^0)_{\eta\in\mathbb{N}} \leq (L^{|\cdot|})_{\eta\in\mathbb{N}}$, $(L^{|\cdot|})_{\eta\in\mathbb{N}} \leq (L^0)_{\eta\in\mathbb{N}}$, $(L_\eta^0)_{\eta\in\mathbb{N}} \leq (L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$, and $(L_\eta^{|\cdot|})_{\eta\in\mathbb{N}} \leq (L_\eta^0)_{\eta\in\mathbb{N}}$.

In many technical matters the formulation of $\mathcal{F}_{\text{PKE}}$ is similar to $\mathcal{F}_{\text{SIG}}$. We also refer to the conventions given in Section 4.

The functionality $\mathcal{F}_{\text{PKE}}$ with leakage $(L_\eta)_{\eta\in\mathbb{N}}$ is defined by the composition of two IITMs

$$\mathcal{F}_{\text{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p) = F_{\text{dec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p) \mid !\underline{F_{\text{enc}}(\mathcal{T}_{\text{enc}})}$$

where F_{dec} and $\underline{F_{\text{enc}}}$ represent the decryptor's and encryptor's part, respectively.

The IITM $F_{\text{enc}} = F_{\text{enc}}(\mathcal{T}_{\text{enc}})$, as defined in Figure 17, is parameterized by a set of names of tapes $\mathcal{T}_{\text{enc}}$ which specifies the tapes that are used by parties which want to connect to F_{enc} to encrypt messages. Note that one tape might be used by an unbounded number of entities. The party version of F_{enc} is used in $\mathcal{F}_{\text{PKE}}$ to model that every encryptor has its own local procedure which she can query to encrypt messages. Upon initialization, i.e. when the encryptor sends an init message, a message is sent to the IITM F_{dec} to guarantee that an instance of it is created. This instance will later be used by F_{enc} upon receiving an encryption request (see below). Then, the initialization request is forwarded to the adversary who is supposed to answer it whereon the control is given back to the encryptor.

The actual functionality of F_{enc}, i.e. to encrypt a message, is left to F_{dec}. Upon an encryption request, the request is forwarded to F_{dec} (see Figure 17 (c)). The purpose of F_{enc} is only to handle initialization and corruption in a more uniform and simpler way.

The IITM $F_{\text{dec}} = F_{\text{dec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p)$, as defined in Figure 16, is parameterized by the family of leakage functions $(L_\eta)_{\eta\in\mathbb{N}}$ which defines the leakage and the domain of plaintexts, two disjoint sets of names of tapes $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ which are used by the decryptor or the encryptors (resp.) to connect to F_{dec} and by a polynomial p which is used to bound the execution time of the algorithms provided by the environment (see below).

The environment provides algorithms e and d for encryption and decryption of messages and a public key k, see Figure 16 (d). Upon initialization the public key k is sent to the decryptor.

The polynomial p is used to bound the size of d, e and k and the runtime of d and e as described in Section 4.3. Note that the polynomial does not limit the power of the adversary since upon corruption, the adversary is not anymore restricted to the polynomial. Also, every potential encryption or decryption algorithm has polynomial runtime. Therefore, it is possible to choose a polynomial such that $\mathcal{F}_{\text{PKE}}$ executes the algorithms as expected.

If a party sends an encryption request of a message m with key k' to F_{enc} then this request is forwarded to F_{dec}. If $m \notin \text{dom}(L_\eta)$ then an error is returned. Note that it can be checked in polynomial time whether $m \in \text{dom}(L_\eta)$ because L_η is a polynomial time algorithm. If the decryptor is not corrupted and the encryptor provides the correct key, i.e. $k = k'$, then F_{dec} computes the formal ciphertext string c by computing $e(k, m')$ where m' is computed as $L_\eta(m)$ and then verifies that it decrypts properly by checking $d(c) = m'$. If it does, F_{dec} records the pair (m, c) and outputs c to the encryptor. Otherwise, the error symbol $\bot$ is sent to the encryptor. The test $d(c) = m'$ is missing in Canetti's $\mathcal{F}_{\text{PKE}}$ [7] but is crucial for the joint state theorem as we point out in Section 6.3. It captures the fact that the ciphertext c does

Functionality $F_{\mathrm{dec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$

I/O-tapes: in: io(T, pke) for each $T \in \mathcal{T}_{\mathrm{dec}}$, io$(E_{\mathrm{dec}}, \mathrm{pke})$, io$(\mathrm{enc}, \mathrm{dec})$ (enriching)
out: io(pke, T) for each $T \in \mathcal{T}_{\mathrm{dec}} \cup \mathcal{T}_{\mathrm{enc}}$, io$(\mathrm{pke}, E_{\mathrm{dec}})$, io$(\mathrm{dec}, \mathrm{enc})$
net-tapes: in: net$(A_{\mathrm{dec}}, \mathrm{pke})$ (consuming) out: net$(\mathrm{pke}, A_{\mathrm{dec}})$
Initialization: $e, d, k \leftarrow \bot$; $H \leftarrow \emptyset$; *state* $\leftarrow$ init ; *nokey* $\leftarrow$ true ; *corrupted* $\leftarrow$ false
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\mathrm{dec}}$, and *state* = init do: *state* $\leftarrow$ (wait, T) ; send (Init) to A_{dec}

(b) *Initialization Response:* If recv (Inited) from A_{dec}, not *nokey*, and *state* = (wait, T) do:
state $\leftarrow$ ok ; send (PublicKey, k) to T

(c) *Wake up:* If recv (pid, WakeUpFromEnc) from enc do: send (pid, Ack) to enc

(d) *Key Generation:* If recv (AlgorithmsAndKey, e', d', k') from A_{dec}, *nokey*, and $|e'|, |d'|, |k'| \le p(\eta)$ do:
$(e, d, k) \leftarrow (e', d', k')$; *nokey* $\leftarrow$ false ; send (Ack) to A_{dec}

(e) *Decryption:* If recv (Decrypt, c) from $T \in \mathcal{T}_{\mathrm{dec}}$, *state* = ok, and not *corrupted* do:

$$m \leftarrow \begin{cases} \bot & \text{if } \exists m', m'' \colon m' \neq m'', (m', c) \in H, (m'', c) \in H \\ m' & \text{if } \exists! m' \colon (m', c) \in H \\ \text{sim-det}_{p(|c|+\eta)}\, d(c) & \text{otherwise ;} \end{cases}$$

send (Plaintext, m) to T

(f) *Encryption:* If recv (pid, Encrypt, k', m, T) from enc where $T \in \mathcal{T}_{\mathrm{enc}}$, and not *nokey* do:
if $m \notin \mathrm{dom}(L_\eta)$ then $c \leftarrow \bot$
else if $k \neq k'$ or *corrupted* then $c \leftarrow \mathrm{sim}_{p(|m|+\eta)} e(k', m)$
else $m' \leftarrow L_\eta(m)$; $c \leftarrow \mathrm{sim}_{p(|m'|+\eta)}\, e(k, m')$; $m'' \leftarrow \text{sim-det}_{p(|c|+\eta)}\, d(c)$; if $m' \neq m''$ then $c \leftarrow \bot$ end ;
if $c \neq \bot$ then $H \leftarrow H \cup \{(m, c)\}$ end
end ;
send (pid, Ciphertext, c) to T

(g) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $H, A_{\mathrm{dec}}, \mathcal{T}_{\mathrm{dec}}, E_{\mathrm{dec}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 16. Ideal public-key encryption functionality $\mathcal{F}_{\mathrm{PKE}} = F_{\mathrm{dec}} \mid !\underline{F_{\mathrm{enc}}}$, the decryptor's part F_{dec}.

Functionality $F_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}})$

I/O-tapes: in: io(T, pke) for each $T \in \mathcal{T}_{\mathrm{enc}}$, io$(E_{\mathrm{enc}}, \mathrm{pke})$ (enriching), io$(\mathrm{dec}, \mathrm{enc})$ (consuming)
out: io(pke, T) for each $T \in \mathcal{T}_{\mathrm{enc}}$, io$(\mathrm{pke}, E_{\mathrm{enc}})$, io$(\mathrm{enc}, \mathrm{dec})$
net-tapes: in: net$(A_{\mathrm{enc}}, \mathrm{pke})$ (consuming) out: net$(\mathrm{pke}, A_{\mathrm{enc}})$
Initialization: *state* $\leftarrow$ init ; *corrupted* $\leftarrow$ false
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\mathrm{enc}}$, and *state* = init do:
send (WakeUpFromEnc) to dec ; wait for recv (Ack) from dec, *state* $\leftarrow$ (wait, T) ; send (Init) to A_{enc}

(b) *Initialization Response:* If recv (Inited) from A_{enc}, and *state* = (wait, T) do: *state* $\leftarrow$ ok; send (Inited) to T

(c) *Encryption:* If recv (Encrypt, k', m) from $T \in \mathcal{T}_{\mathrm{enc}}$, *state* = ok, and not *corrupted* do:
send (Encrypt, k', m, T) to dec

(d) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $\varepsilon, A_{\mathrm{enc}}, \mathcal{T}_{\mathrm{enc}}, E_{\mathrm{enc}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 17. Ideal public-key encryption functionality $\mathcal{F}_{\mathrm{PKE}} = F_{\mathrm{dec}} \mid !\underline{F_{\mathrm{enc}}}$, the encryptor's part F_{enc}.

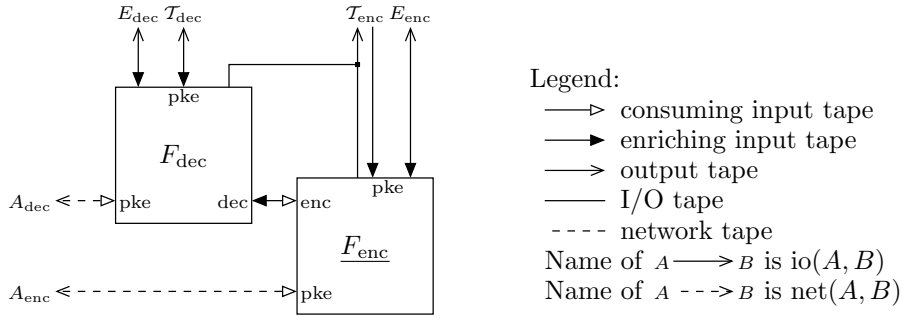

Fig. 18. Graphical representation of the ideal functionality for public-key encryption $\mathcal{F}_{\text{PKE}} = F_{\text{dec}} \mid !\underline{F_{\text{enc}}}$.

not only contain not more than m' but exactly as much information. Since every encryption scheme has to guarantee correct decryption, this is an assumption satisfied for all realistic encryption schemes. If the decryptor is corrupted or the encryptor provides a wrong key, i.e. $k \neq k'$, F_{dec} computes $c \leftarrow e(k', m)$ and sends c to the encryptor. See Figure 16 (f) and Figure 17 (c).

The owner of F_{dec}, i.e. the decryptor, can decrypt ciphertexts by sending $(\mathsf{Decrypt}, c)$ to F_{dec} as soon as he has registered and algorithms are provided. If (m, c) and (m', c) is recorded for plaintexts $m \neq m'$, the decryption of c is *ambiguous* and the error symbol $\perp$ is returned to the decryptor. If there is exactly one recorded pair (m, c) for some plaintext m, F_{dec} returns the plaintext m to the decryptor. Otherwise, the decryption algorithm d is executed on c and the result m is sent to the decryptor. See Figure 16 (e).

Corruption is modeled by the use of the macro `Corr` as defined in Figure 1. Both decryptor and each encryptor can be corrupted adaptively. Upon corruption of the decryptor the set H is given to the environment or simulator, i.e., the entire state of the functionality is revealed. For the joint state realization to work it is required that the representation of H hides the order in which the encryptions have been done, e.g., H could be represented as a list in lexicographical order (see Section 6.3).

The external tapes of $\mathcal{F}_{\text{PKE}}$ and the connection between F_{dec} and F_{enc} are pictured in Figure 18.

Next, we summarize some properties of $\mathcal{F}_{\text{PKE}}$ to show that it specifies an ideal public-key encryption functionality:

- Since the adversary provides the algorithms d and e, no requirements are imposed on these algorithms. When realizing the functionality by a particular encryption scheme, this mechanism allows the simulator to choose e, d, and k at its convenience.
- Assuming that the decryptor is not corrupted, and an encryptor encrypts a message m with the correct encryption key k then:
  $\mathcal{F}_{\text{PKE}}$ *guarantees correct decryption*, i.e. if a message was encrypted, the decryptor is able to decrypt the ciphertext c (if it is not ambiguous), because upon encryption the pair (m, c) was recorded.
  $\mathcal{F}_{\text{PKE}}$ *guarantees security*. Since the generated ciphertext c only depends on $L_\eta(m)$ and not on m, the adversary is not able to learn more from c than $L_\eta(m)$.
- The decryption process is consistent. If a decryptor obtained the plaintext m from $\mathcal{F}_{\text{PKE}}$ upon a decryption request of a ciphertext c then every later decryption request of c will result in the same plaintext m or an error message (as long as the decryptor is not corrupted).
- If the decryptor is corrupted then no security is guaranteed because the ciphertext might depend on the plaintext. For example, the adversary could provide the identity function as algorithms e and d.
- If the encryptor provides a wrong key $k' \neq k$ then no security is guaranteed as if the decryptor is corrupted. This models the fact that the public key is a priori not bound to the identity of the decryptor and no assumptions are made on how the public key is distributed.
- If the functionality is used with a leakage that has high entropy, then it guarantees that unknown ciphertext cannot be guessed. Let us explain: Assume that, e.g., due to nested encryption, a ciphertext c was generated by $\mathcal{F}_{\text{PKE}}$ and that c is not known to the adversary because it was never output to the adversary. If the leakage has high entropy, the following is easy to see: The adversary has only negligible guessing probability for all ciphertexts that are stored in H in $\mathcal{F}_{\text{PKE}}$ and which are formally unknown to the adversary. The proof idea is to exploit that the ciphertext has to contain as much

information as $L_\eta(m)$, because of the decryption test during encryption. Since the leakage has high entropy, $L_\eta(m)$ is sufficiently random and can be guessed only with negligible probability.

It can be shown that a realization of $\mathcal{F}_{\mathrm{PKE}}$ is impossible if the decryptor is adaptive corruptible [26]. However, our formulation of $\mathcal{F}_{\mathrm{PKE}}$ allows for adaptive corruption of the decryptor. This is not a problem but a generalization, as the simulator is able to block corruption requests and therefore "protects" $\mathcal{F}_{\mathrm{PKE}}$ from being corrupted adaptively.

Note that we allow for independent corruption of the decryptor and the encryptors.

6.2 Implementation by a CCA-Secure Encryption Scheme

In this section, it is shown that the ideal public-key encryption functionality $\mathcal{F}_{\mathrm{PKE}}$ that leaks the length of a message is realized by a public-key encryption scheme if and only if the encryption scheme is secure with respect to adaptive chosen-ciphertext attacks (CCA-secure) as defined below.

In terms of [4] a *public-key encryption scheme* $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$ consists of two probabilistic polynomial time algorithms gen, enc and a deterministic polynomial time algorithm dec. Where the key generation algorithm gen takes 1^η as an input (where η is the security parameter) and outputs a pair of keys (k_d, k_e), the secret (or decryption) key k_d and the public (or encryption) key k_e. The encryption algorithm enc expects a public key k_e and a message $m \in \{0,1\}^*$ as input and produces a ciphertext $c = \mathrm{enc}(k_e, m)$. Upon input of a private key k_d and a ciphertext c the decryption algorithm dec outputs a plaintext $m \in \{0,1\}^*$ or the special symbol $\perp$ to indicate that c was invalid. It is required that decryption is correct, i.e. $\mathrm{dec}(k_d, \mathrm{enc}(k_e, m)) = m$ for all $m \in \{0,1\}^*$ and all (k_d, k_e) generated by gen.

Our notion of *CCA-secure* is defined as indistinguishability of encryptions with respect to adaptive chosen-ciphertext attacks (IND-CCA2) as in [4] and is the notion of [29]. IND-CCA2 implies every other security notion discussed in [4] and is equivalent to non-malleability with respect to adaptive chosen-ciphertext attacks (NM-CCA2) which is the notion of [18]. Next, the definition of IND-CCA2 is restated.

An *adversary* A is a pair of probabilistic polynomial time algorithms $A = (A_1, A_2)$. Algorithm A_1 is run on the public key k_e and 1^η as input (where η is the security parameter) and outputs a triple (x_0, x_1, s) where $x_0 \in \{0,1\}^*$ and $x_1 \in \{0,1\}^*$ have to be of the same length. The string $s \in \{0,1\}^*$ can be used by A_1 to pass information to A_2. Algorithm A_2 is run on input $x_0 \in \{0,1\}^*$, $x_1 \in \{0,1\}^*$, $s \in \{0,1\}^*$, $c \in \{0,1\}^*$, and 1^η and outputs a bit $b' \in \{0,1\}$. By $A_1^{O(\cdot)}$ and $A_2^{O(\cdot)}$ we denote that A_1 and A_2 (resp.) have access to the decryption oracle $O(\cdot)$.

Let $A = (A_1, A_2)$ be an adversary and $b \in \{0,1\}$. The corresponding *experiment* is an algorithm $T_{A,\Sigma}$ that runs on input $b \in \{0,1\}$ and $\eta \in \mathbb{N}$.

$$\begin{aligned} T_{A,\Sigma}(b,\eta)\colon\ & (k_d, k_e) \leftarrow \mathrm{gen}(1^\eta);\ (x_0, x_1, s) \leftarrow A_1^{\mathrm{dec}(k_d,\cdot)}(k_e, 1^\eta); \\ & c^* \leftarrow \mathrm{enc}(k_e, x_b);\ b' \leftarrow A_2^{\overline{\mathrm{dec}}(c^*,k_d,\cdot)}(x_0, x_1, s, c^*, 1^\eta);\ \text{return } b'; \end{aligned}$$

where $\overline{\mathrm{dec}}$ is defined as follows:

$$\overline{\mathrm{dec}}(c^*, k_d, c)\colon \text{if } c = c^* \text{ then return } \mathsf{test} \text{ else return } \mathrm{dec}(k_d, c);$$

where the message test is never returned by dec.

The *advantage* of an adversary $A = (A_1, A_2)$ regarding the public-key encryption scheme Σ is defined by

$$\mathrm{Adv}(A, \Sigma, \eta) = |\mathrm{Prob}[T_{A,\Sigma}(1,\eta) = 1] - \mathrm{Prob}[T_{A,\Sigma}(0,\eta) = 1]| \ .$$

Definition 5 (CCA-secure; [4]). *A public-key encryption scheme Σ is called* secure against adaptive chosen-ciphertext attacks (CCA-secure) *if for all adversaries the advantage* $\mathrm{Adv}(A, \Sigma, \eta)$ *is negligible as a function in η.*

Given a public-key encryption scheme $\Sigma = (\mathrm{gen}, \mathrm{enc}, \mathrm{dec})$, the protocol system $\mathcal{P}_{\mathrm{PKE}}(\Sigma)$ (as defined below) models the encryption scheme as a protocol system with non-adaptive corruption behavior of the decryptor and adaptive corruption behavior of the encryptors.

Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be two disjoint sets of names of tapes. Let $(D_\eta)_{\eta\in\mathbb{N}}$ be a family of domains where for every $x \in \{0,1\}^*$ it is possible to verify in polynomial time (in η) whether $x \in D_\eta$ for all $\eta \in \mathbb{N}$. We define

$$\mathcal{P}_{\mathrm{PKE}}(\Sigma, (D_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}) = P_{\mathrm{dec}}(\mathrm{gen}, \mathrm{dec}, \mathcal{T}_{\mathrm{dec}}) \mid \underline{!P_{\mathrm{enc}}(\mathrm{enc}, (D_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{enc}})}$$

Realization $P_{\text{dec}}(\text{gen}, \text{dec}, \mathcal{T}_{\text{dec}})$

I/O-tapes: in: $\text{io}(T, \text{pke})$ for each $T \in \mathcal{T}_{\text{dec}}$, $\text{io}(E_{\text{dec}}, \text{pke})$, $\text{io}(\text{enc}, \text{dec})$ (enriching)
out: $\text{io}(\text{pke}, T)$ for each $T \in \mathcal{T}_{\text{dec}}$, $\text{io}(\text{pke}, E_{\text{dec}})$, $\text{io}(\text{dec}, \text{enc})$
net-tapes: in: $\text{net}(A'_{\text{dec}}, \text{pke})$ (consuming) out: $\text{net}(\text{pke}, A'_{\text{dec}})$
Initialization: *state* $\leftarrow$ init ; $k_d, k_e \leftarrow \perp$; *corrupted* $\leftarrow$ false ; *corruptible* $\leftarrow$ true
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{dec}}$, and *state* = init do:
$(k_d, k_e) \leftarrow \text{gen}(1^\eta)$; *state* $\leftarrow$ ok ; send (PublicKey, k_e) to T
(b) *Decryption:* If recv (Decrypt, c) from $T \in \mathcal{T}_{\text{dec}}$, *state* = ok, and not *corrupted* do:
$m \leftarrow \text{dec}(k_d, c)$; send (Plaintext, m) to T
(c) *Block Corruption:* If recv (*pid*, BlockCorruption) from enc do: *corruptible* $\leftarrow$ false ; send (*pid*, Ack) to enc
(d) *Corruption:* Corr(*corrupted*, *corruptible*, *state* $\neq$ init, (k_d, k_e), $A'_{\text{dec}}, \mathcal{T}_{\text{dec}}, E_{\text{dec}}$) (See Figure 1)

If no rule above fires then produce no output.

Fig. 19. Realization of a public-key encryption scheme $\mathcal{P}_{\text{PKE}} = P_{\text{dec}} \mid !\underline{P_{\text{enc}}}$, the decryptor's part P_{dec}.

Realization $P_{\text{enc}}(\text{enc}, (D_\eta)_{\eta \in \mathbb{N}}, \mathcal{T}_{\text{enc}})$

I/O-tapes: in: $\text{io}(T, \text{pke})$ for each $T \in \mathcal{T}_{\text{enc}}$, $\text{io}(E_{\text{enc}}, \text{pke})$ (enriching), $\text{io}(\text{dec}, \text{enc})$ (consuming)
out: $\text{io}(\text{pke}, T)$ for each $T \in \mathcal{T}_{\text{enc}}$, $\text{io}(\text{pke}, E_{\text{enc}})$, $\text{io}(\text{enc}, \text{dec})$
net-tapes: in: $\text{net}(A'_{\text{enc}}, \text{pke})$ (consuming) out: $\text{net}(\text{pke}, A'_{\text{enc}})$
Initialization: *state* $\leftarrow$ init ; *corrupted* $\leftarrow$ false
CheckAddress: Accept all messages on all tapes
Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\text{enc}}$, and *state* = init do:
state $\leftarrow$ ok ; send (BlockCorruption) to dec ; wait for recv (Ack) from dec ; send (Inited) to T
(b) *Encryption:* If recv (Encrypt, k, m) from $T \in \mathcal{T}_{\text{enc}}$, *state* = ok, and not *corrupted* do:
if $m \in D_\eta$ then $c \leftarrow \text{enc}(k, m)$ else $c \leftarrow \perp$ end ; send (Ciphertext, c) to T
(c) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $\varepsilon, A'_{\text{enc}}, \mathcal{T}_{\text{enc}}, E_{\text{enc}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 20. Realization of a public-key encryption scheme $\mathcal{P}_{\text{PKE}} = P_{\text{dec}} \mid !\underline{P_{\text{enc}}}$, the encryptor's part P_{enc}.

where P_{dec} and P_{enc} are specified in Figure 19 and 20. A graphical representation of $\mathcal{P}_{\text{PKE}}$ and the connection between P_{dec} and $!\underline{P_{\text{enc}}}$ is pictured in Figure 21. The communication between P_{dec} and $\underline{P_{\text{enc}}}$ via tapes $\text{io}(\text{dec}, \text{enc})$ and $\text{io}(\text{enc}, \text{dec})$ is only used to model non-adaptive corruption behavior (see below for more details).

The IITM $P_{\text{dec}} = P_{\text{dec}}(\text{gen}, \text{dec}, \mathcal{T}_{\text{dec}})$ belongs to the decryptor. Upon initialization, a private key k_d and a public key k_e are generated with the key generation algorithm gen. The IITM P_{dec} has non-adaptive corruption behavior, i.e. the decryptor is non-adaptive corruptible which means in this case that it accepts corruption requests until an encryptor has initialized its instance of $\underline{P_{\text{enc}}}$. This is modeled by setting *corruptible* to false once P_{enc} received (Init), see Figure 19 (d) and 20 (a). Upon corruption, P_{dec} reveals the private key k_d and the public key k_e to the adversary.

When the decryptor requests to decrypt a ciphertext c, P_{dec} computes the plaintext m with the decryption algorithm $\text{dec}(k_d, c)$ and returns m to the decryptor.

Each instance of $\underline{P_{\text{enc}}} = \underline{P_{\text{enc}}(\text{enc}, \mathcal{T}_{\text{enc}})}$ belongs to an encryptor. We denote the instance of party with party identifier (PID) *pid* by $\overline{P_{\text{enc}}[pid]}$. Registration requests are directly answered positively. When the encryptor *pid* requests to encrypt a message m with key k, $P_{\text{enc}}[pid]$ computes the ciphertext c with the encryption algorithm $c \leftarrow \text{enc}(k, m)$ and returns c to the encryptor *pid*. If m is not in the domain of plaintexts, i.e. $m \notin D_\eta$ (which can be checked in polynomial time by assumption) then an error message is returned instead. In contrast to P_{dec}, P_{enc} has adaptive corruption behavior.

Next, we show that a public-key encryption scheme Σ realizes the ideal functionality $\mathcal{F}_{\text{PKE}}((L_\eta^{|\cdot|})_{\eta \in \mathbb{N}})$ which does not leak more than the length of the plaintext if and only if Σ is CCA-secure (see Corollary

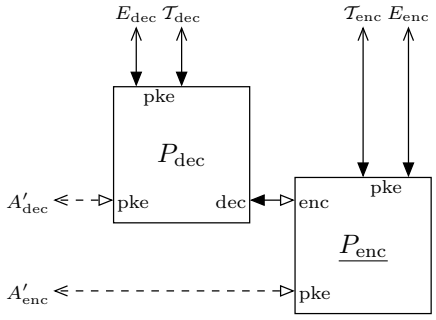

Fig. 21. Graphical representation of the public-key encryption implementation $\mathcal{P}_{\text{PKE}} = P_{\text{dec}} \mid !\underline{P_{\text{enc}}}$. See Figure 18 for a legend.

2). We split the proposition into two parts, Theorem 7 and Lemma 1, to obtain more general results for the case where the leakage is not $(L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$.

A public-key encryption scheme $\Sigma = (\text{gen}, \text{enc}, \text{dec})$ is called *p-bounded* if the runtime of $\text{gen}(1^\eta)$, $\text{dec}(k_d, c)$ and $\text{enc}(k_e, m)$ is bounded by $p(\eta)$, $p(|c|+\eta)$ and $p(|m|+\eta)$ (resp.) for every η, m, c, k_d and k_e and if the length of the description of gen, dec and enc is bounded by $p(\eta)$. Note that for each encryption scheme, there is a polynomial p such that it is p-bounded.

The following theorem shows that a CCA-secure encryption scheme realizes $\mathcal{F}_{\text{PKE}}$ in the context of environments without auxiliary input. One could alternatively define CCA-security by allowing auxiliary input to the adversary. Then, a CCA-secure encryption scheme would realizes $\mathcal{F}_{\text{PKE}}$ in the context of environments with auxiliary input.

Theorem 7. *Let $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ be some disjoint sets of names of tapes, Σ a CCA-secure p-bounded public-key encryption scheme. Then,*

$$\mathcal{P}_{\text{PKE}}(\Sigma, (\text{dom}(L_\eta))_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\text{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p)$$

for every length preserving leakage $(L_\eta)_{\eta\in\mathbb{N}}$ (see above for the definition of length preserving).

Proof. We prove the theorem by contradiction. Let Σ be a p-bounded public-key encryption scheme and assume that $\mathcal{P}_{\text{PKE}}(\Sigma, (\text{dom}(L_\eta))_{\eta\in\mathbb{N}})$ does not SS-realize $\mathcal{F}_{\text{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, p)$. We will show that Σ is not CCA-secure by constructing an adversary $A = (A_1, A_2)$ with non-negligible advantage from the environment $\mathcal{E}$ that distinguishes $\mathcal{P}_{\text{PKE}}$ from $\mathcal{S}_{\text{PKE}} \mid \mathcal{F}_{\text{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, p)$ with the simulator $\mathcal{S}_{\text{PKE}} = \mathcal{S}_{\text{PKE}}(\Sigma)$ given in Figure 22. A graphical representation of $\mathcal{S}_{\text{PKE}}$ and its connection to $\mathcal{F}_{\text{PKE}}$ is pictured in Figure 23. One easily verifies that $\mathcal{S}_{\text{PKE}}$ is a simulator as required in Definition 1.

The runtime of $\mathcal{E}$ in a run of $\mathcal{E} \mid \mathcal{P}_{\text{PKE}}(1^\eta, \varepsilon)$ is polynomially bounded in η. Therefore, the total number of messages of type $(pid, \mathsf{Encrypt}, k', m)$ that are send by $\mathcal{E}$ is bounded by $q(\eta)$ for some polynomial q.

The description of algorithm $A = (A_1^{O(\cdot)}(k, 1^\eta), A_2^{O(\cdot)}(x_0, x_1, s, c, 1^\eta))$ follows. At first algorithm $A_1^{O(\cdot)}(k, 1^\eta)$ initializes a counter $i \leftarrow 0$ that indicates how many messages where already encrypted by $\mathcal{E}$ with key k. Then, A_1 chooses $h \leftarrow^{\text{R}} \{1, \ldots, q(\eta)\}$ (uniformly at random) and simulates the run of $\mathcal{E} \mid \mathcal{P}_{\text{PKE}}$ on input ε with the security parameter η as defined how to run a system in Section 2 with the following exceptions:

(a) When P_{dec} wants to simulate $\text{gen}(1^\eta)$ then continue the simulation as if $\text{gen}(1^\eta)$ returned the decryption key 0 and the encryption key k.
(b) When some instance of $\underline{P_{\text{enc}}}$ wants to simulate $\text{enc}(k', m)$ (Figure 20 (b)) then:
- If $k = k'$ and $i + 1 < h$ then A_1 computes $i \leftarrow i + 1$, $m_i \leftarrow m$, $c_i \leftarrow \text{enc}(k, m_i)$ and continues the simulation with c_i.
- If $k = k'$ and $i + 1 = h$ then A_1 computes $m_h \leftarrow m$ and stores all information needed for the simulation of $\mathcal{E}$ plus all m_j and c_j $(j \leq h)$ plus h plus the key k in the string s. Then A_1 halts with output $(m_h, L_\eta(m_h), s)$ (Note that $|m_h| = |L_\eta(m_h)|$).
- If $k \neq k'$ then A_1 continues the simulation normally, i.e. as defined in P_{enc}.

Simulator $S_{\mathrm{PKE}}(\Sigma)$

net-tapes: in: $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$, $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$, $\mathrm{net}(A'_{\mathrm{dec}}, \mathrm{pke})$, $\mathrm{net}(A'_{\mathrm{enc}}, \mathrm{pke})$ (enriching)
out: $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$, $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$, $\mathrm{net}(\mathrm{pke}, A'_{\mathrm{dec}})$, $\mathrm{net}(\mathrm{pke}, A'_{\mathrm{enc}})$

Initialization: *corruptible* $\leftarrow$ true ; *state* $\leftarrow$ init ; *nokey* $\leftarrow$ true ; $(k_d, k_e) \leftarrow \mathrm{gen}(1^\eta)$

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

(a) *Initialization (Dec):* If recv (Init) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$ do:
if *nokey* then
send $(\mathsf{AlgorithmsAndKey}, \mathrm{dec}(k_d, \cdot), \mathrm{enc}(\cdot, \cdot), k_e)$ to $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$;
wait for recv (Ack) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$; *nokey* $\leftarrow$ false
end ;
state $\leftarrow$ ok ; send (Inited) to $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$

(b) *Corruption (Dec):* If recv (Corrupt) on $\mathrm{net}(A'_{\mathrm{dec}}, \mathrm{pke})$, *state* $\neq$ init, and *corruptible* do:
send (Corrupt) to $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$

(c) *Corrupted (Dec):* If recv $(\mathsf{Corrupted}, \emptyset)$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$ do: send $(\mathsf{Corrupted}, (k_d, k_e))$ to $\mathrm{net}(\mathrm{pke}, A'_{\mathrm{dec}})$

(d) *Forward from A (Dec):* If recv $m = (\mathsf{Send}, m', T)$ on $\mathrm{net}(A'_{\mathrm{dec}}, \mathrm{pke})$ do: send m to $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$

(e) *Forward to A (Dec):* If recv $m = (\mathsf{Received}, m', T)$ or $m = (\mathsf{ReceivedResources}, r)$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$ do:
send m to $\mathrm{net}(\mathrm{pke}, A'_{\mathrm{dec}})$

(f) *Initialization (Enc):* If recv (pid, Init) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ do:
if *nokey* then
send $(\mathsf{AlgorithmsAndKey}, \mathrm{dec}(k_d, \cdot), \mathrm{enc}(\cdot, \cdot), k_e)$ to $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$;
wait for recv (Ack) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$; *nokey* $\leftarrow$ false
end ;
corruptible $\leftarrow$ false, send (pid, Inited) to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$

(g) *Forward from A (Enc):* If recv m on $\mathrm{net}(A'_{\mathrm{enc}}, \mathrm{pke})$ do: send m to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$

(h) *Forward to A (Enc):* If recv $m \neq (pid, \mathsf{Init})$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ do: send m to $\mathrm{net}(\mathrm{pke}, A'_{\mathrm{enc}})$

If no rule above fires then produce no output.

Fig. 22. Simulator S_{PKE} for the proof of $\mathcal{P}_{\mathrm{PKE}}$ SS-realizes $\mathcal{F}_{\mathrm{PKE}}$.

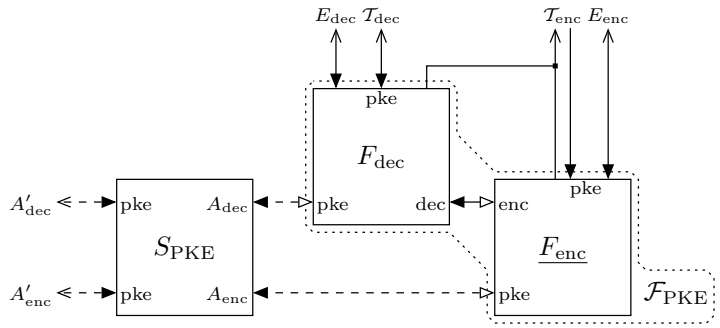

Fig. 23. Graphical representation of the simulator S_{PKE} for $\mathcal{F}_{\mathrm{PKE}} = F_{\mathrm{dec}} \mid !\underline{F_{\mathrm{enc}}}$. See Figure 18 for a legend.

(c) When P_{dec} wants to simulate $\mathrm{dec}(k_d, c)$ (Figure 19 (b)) then A_1 checks if $c = c_j = c_{j'}$ for some $j, j' \leq i$ with $m_j \neq m_{j'}$. If it does, A_1 continues the simulation as if P_{dec} send an error message. Otherwise, A_1 checks if $c = c_j$ for some $j \leq i$. If it does, A_1 continues the simulation with m_j. Otherwise, A_1 computes $m \leftarrow O(c)$ with its decryption oracle and continues the simulation with m.
(d) When $\mathcal{E}$ outputs f on tape decision then A_1 halts with output $(1, 1, f)$.

Algorithm $A_2^{O(\cdot)}(x_0, x_1, s, c, 1^\eta)$ restores the information encoded in s and sets $i \leftarrow h$ and $c_h \leftarrow c$. If $x_0 = x_1 = 1$ then A_2 halts with output $s = f$. Otherwise, A_2 continues the simulation of $\mathcal{E}$ as if the simulation of $\mathrm{enc}(k', m)$ in $\underline{P_{\mathrm{enc}}}$ returned c_h with the following exceptions:

(a) When some instance of $\underline{P_{\mathrm{enc}}}$ wants to simulate $\mathrm{enc}(k', m)$ (Figure 20 (b)) then:
- If $k = k'$ then A_2 computes $i \leftarrow i + 1$, $m_i \leftarrow m$, $c_i \leftarrow \mathrm{enc}(k, L_\eta(m_i))$ and continues the simulation with c_i.
- If $k \neq k'$ then A_2 continues the simulation normally.

(b) When P_{dec} wants to simulate $\mathrm{dec}(k_d, c)$ (Figure 19 (b)) then A_2 checks if $c = c_j = c_{j'}$ for some $j, j' \leq i$ with $m_j \neq m_{j'}$. If it does, A_2 continues the simulation as if P_{dec} sent an error message. Otherwise, A_2 checks if $c = c_j$ for some $j \leq i$. If it does, A_2 continues the simulation with m_j. Otherwise, A_2 computes $m \leftarrow O(c)$ with its decryption oracle and continues the simulation with m.
(c) When $\mathcal{E}$ outputs f on tape decision then A_2 halts with output f.

Note that A_2 will never ask its oracle $O(\cdot)$ to decrypt the challenge c_h, because A_2 checks if the ciphertext c equals c_h and does not ask $O(\cdot)$ in that case.

To analyze the advantage of A, as [7], we define a random variable H_j for all $j \in \{0, \ldots, q(\eta)\}$ which denotes the output of $\mathcal{E}$ in an interaction with $\mathcal{S}_{\mathrm{PKE}} \,|\, \mathcal{F}_{\mathrm{PKE}}$ with security parameter η except that the first j times where F_{dec} receives $(pid, \mathsf{Encrypt}, k', m, T)$, $k = k'$ and the decryptor is not corrupted the ciphertext c is computed by the encryption of m, i.e. $c \leftarrow e(k, m)$, instead of the encryption of $L_\eta(m)$, i.e. $c \leftarrow e(k, L_\eta(m))$, and it is not tested whether $d(c) = L_\eta(m)$ or not. In other words, "$m' \leftarrow L_\eta(m)$, $c \leftarrow e(k, m')$, $m'' \leftarrow d(c)$, if $m' \neq m''$ then $c \leftarrow \perp$, if $c \neq \perp$ then $H \leftarrow H \cup \{(m, c)\}$" in Figure 16 (e) is replaced by "$c \leftarrow e(k, m)$, $H \leftarrow H \cup \{(m, c)\}$".

By the assumption that $\mathcal{E}$ distinguishes $\mathcal{P}_{\mathrm{PKE}}$ from $\mathcal{S}_{\mathrm{PKE}} \,|\, \mathcal{F}_{\mathrm{PKE}}$ we conclude that $\mathcal{E}$ does not corrupt the decryptor. Otherwise, the view of $\mathcal{E}$ in the real world would be the same as in the ideal world.

The view of $\mathcal{E}$ as simulated by H_0 would only differ from the view in the ideal world if F_{dec} sends the error symbol $\perp$ because $m' \neq m''$. Since Σ is an encryption scheme, $d(\mathrm{enc}(k, m')) = m'$ for all m' which are generated by $L(m)$, so,

$$\mathrm{Prob}[H_0 = 1] = \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\mathrm{PKE}} \,|\, \mathcal{F}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 1] \ .$$

We show that

$$\mathrm{Prob}[H_{q(\eta)} = 1] = \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 1] \ .$$

The random variable $H_{q(\eta)}$ denotes the output of $\mathcal{E}$ in an interaction with $\mathcal{S}_{\mathrm{PKE}} \,|\, \mathcal{F}_{\mathrm{PKE}}$ except that upon encryption, always $e(k, m)$ is computed instead of $e(k, L_\eta(m))$. Thus, the output of $\mathcal{E}$ differs from $H_{q(\eta)}$ only upon decryption of a ciphertext c:

(a) If c is ambiguous, i.e. if there are two recorded pairs (m, c) and (m', c) with $m \neq m'$, because then in the ideal world an error message is produced, or
(b) If there is a recorded pair (m, c) and $\mathrm{dec}(k_d, c) \neq m$.

If case (a) would occur, then $\mathrm{enc}(k, m)$ and $\mathrm{enc}(k, m')$ with $m \neq m'$ produced the same ciphertext c. This can not happen, because Σ is an encryption scheme and has the property that decryption is correct, i.e. $\mathrm{dec}(k_d, \mathrm{enc}(k_e, m)) = m$ for all $m \in \{0, 1\}^*$ and all (k_d, k_e) generated by gen. Since otherwise, $m = \mathrm{dec}(k_d, \mathrm{enc}(k, m)) = \mathrm{dec}(k_d, c) = \mathrm{dec}(k_d, \mathrm{enc}(k, m')) = m'$. A contradiction to $m \neq m'$.

If (m, c) was recorded then c was computed by $\mathrm{enc}(k, m)$ which implies $m = \mathrm{dec}(k_d, c)$ because Σ is an encryption scheme. Thus, case (b) does not occur either.

Because $\mathcal{E}$ does not corrupt the decryptor, decryption is correct, and $e(k, L_\eta(m)) = \mathrm{enc}(k, L_\eta(m))$ and by the definition of A_1's behavior (c) and A_2's behavior (b) we have for all $j \in \{1, \ldots, q'(\eta)\}$

$$\mathrm{Prob}[H_{j-1} = 1] = \mathrm{Prob}[T_{A,\Sigma}(1, \eta) = 1 \mid h = j] \text{ and}$$
$$\mathrm{Prob}[H_j = 1] = \mathrm{Prob}[T_{A,\Sigma}(0, \eta) = 1 \mid h = j] \ .$$

The advantage of A is

$$\begin{aligned}
&\mathrm{Adv}(A,\Sigma,\eta) = |\mathrm{Prob}[T_{A,\Sigma}(1,\eta)=1] - \mathrm{Prob}[T_{A,\Sigma}(0,\eta)=1]| \\
&= \left|\sum_{j=1}^{q(\eta)} \mathrm{Prob}[T_{A,\Sigma}(1,\eta)=1, h=j] - \mathrm{Prob}[T_{A,\Sigma}(0,\eta)=1, h=j]\right| \\
&= \left|\sum_{j=1}^{q(\eta)} \frac{1}{q(\eta)}\mathrm{Prob}[T_{A,\Sigma}(1,\eta)=1 \mid h=j] - \frac{1}{q(\eta)}\mathrm{Prob}[T_{A,\Sigma}(0,\eta)=1 \mid h=j]\right| \\
&= \frac{1}{q(\eta)}\left|\sum_{j=1}^{q(\eta)} \mathrm{Prob}[T_{A,\Sigma}(1,\eta)=1 \mid h=j] - \mathrm{Prob}[T_{A,\Sigma}(0,\eta)=1 \mid h=j]\right| \\
&= \frac{1}{q(\eta)}\left|\sum_{j=1}^{q(\eta)} \mathrm{Prob}[H_{j-1}=1] - \mathrm{Prob}[H_j=1]\right| \\
&= \frac{1}{q(\eta)}|\mathrm{Prob}[H_0=1] - \mathrm{Prob}[H_{q(\eta)}=1]| \\
&= \frac{1}{q(\eta)}|\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S}_{\mathrm{PKE}} \,|\, \mathcal{F}_{\mathrm{PKE}}(1^\eta,\varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta,\varepsilon) \rightsquigarrow 1]| \\
&> \frac{1}{q(\eta)} \cdot \frac{1}{p(\eta)}
\end{aligned}$$

for infinitely many η. Thus, the advantage of A is non-negligible. □

We say that a family of leakage algorithms $(L_\eta)_{\eta\in\mathbb{N}}$ *leaks at most the length of a message* if $(L_\eta)_{\eta\in\mathbb{N}} \leq (L'_\eta)_{\eta\in\mathbb{N}}$ where $\{L'_\eta(x)\colon$ if $x \in \mathrm{dom}(L_\eta)$ then return $0^{|x|}$ else return $\bot\}$. Note that $(L^0_\eta)_{\eta\in\mathbb{N}}$ and $(L^{|\cdot|}_\eta)_{\eta\in\mathbb{N}}$ both leak at most the length of a message.

Lemma 1. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes, $\Sigma = (\mathrm{gen}, \mathrm{dec}, \mathrm{enc})$ a p-bounded public-key encryption scheme such that $\mathrm{Prob}[\mathrm{enc}(k,m)=\bot] = \mathrm{Prob}[L_\eta(m)=\bot]$ for all k generated by $\mathrm{gen}(1^\eta)$ and all $m \in \{0,1\}^*$ and*

$$\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$$

where $(L_\eta)_{\eta\in\mathbb{N}}$ is a leakage which leaks at most the length of a message. Then, Σ is CCA-secure.

Proof. We prove the lemma by contraposition. Assume that Σ is not CCA-secure, i.e. there is an adversary $A = (A_1, A_2)$ with non-negligible advantage. We will construct an environment $\mathcal{E}$ that distinguishes $\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}})$ ($\mathcal{P}_{\mathrm{PKE}}$ for short) from $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{PKE}}$ for each simulator $\mathcal{S}$.

Let $T \in \mathcal{T}_{\mathrm{dec}}$ and $T' \in \mathcal{T}_{\mathrm{enc}}$. We define $\mathcal{E}$ to be a master IITM (an IITM with a tape named start) with an output tape named decision and tapes to connect to $\mathcal{P}_{\mathrm{PKE}}$. In mode CheckAddress $\mathcal{E}$ accepts every incoming message and in mode Compute it operates as follows:

(a) Upon first activation (on tape start) output (Init) on tape $\mathrm{io}(T, \mathrm{pke})$.
(b) Upon receiving $(\mathsf{PublicKey}, k)$ on $\mathrm{io}(\mathrm{pke}, T)$.
(c) Simulate the adversary A_1 with input k as the public key.
- When A_1 asks its decryption oracle to decrypt a message c then output $(\mathsf{Decrypt}, c)$ on $\mathrm{io}(T, \mathrm{pke})$ and wait for receiving $(\mathsf{Plaintext}, m)$ on $\mathrm{io}(\mathrm{pke}, T)$. Then continue simulating A_1 as if the oracle returned m (note that possibly $m = \bot$).
- When A_1 terminates and outputs two test messages x_0, x_1 with $|x_0| = |x_1|$ and state information s, then choose $b \leftarrow^{\mathrm{R}} \{0,1\}$ (uniformly at random), send message $(pid, \mathsf{Encrypt}, k, x_b)$ to $\mathrm{io}(T', \mathrm{pke})$, and wait for receiving $(pid, \mathsf{Ciphertext}, c^*)$ on $\mathrm{io}(\mathrm{pke}, T')$.

(d) Then, start the simulation of A_2 with input x_0, x_1, s and c^* with the following exceptions:
- When A_2 asks its decryption oracle to decrypt a message c then i) if $c \neq c^*$ output $(\mathsf{Decrypt}, c)$ on $\mathrm{io}(T, \mathrm{pke})$ and wait for receiving $(\mathsf{Plaintext}, m)$ on $\mathrm{io}(\mathrm{pke}, T)$. Then continue simulating A_2 as if the oracle returned m ii) otherwise if $c = c^*$ then continue simulating A_2 as if the oracle returned test.

- When A_2 terminates and outputs bit $b' \in \{0,1\}$ then check if the decryptor is corrupted or if *pid* is a corrupted encryptor, i.e.:
  - Send (Corrupted?) to $\mathrm{io}(\mathrm{pke}, T)$
  - Upon receiving (true) on $\mathrm{io}(T, \mathrm{pke})$ choose $b'' \leftarrow^{\mathrm{R}} \{0,1\}$ and output b'' on tape decision and halt else send $(pid, \mathsf{Corrupted}?)$ to $\mathrm{io}(\mathrm{pke}, T')$
  - Upon receiving (true) on $\mathrm{io}(T', \mathrm{pke})$ choose $b'' \leftarrow^{\mathrm{R}} \{0,1\}$ and output b'' on tape decision and halt else output $b \oplus b'$ on tape decision and halt.

If at some point above $\mathcal{E}$ waits for a message to receive and the input is not as expected or on an unexpected tape then $\mathcal{E}$ chooses $b'' \leftarrow^{\mathrm{R}} \{0,1\}$, outputs b'' on tape decision, and halts.

One easily verifies that $\mathcal{E}$ meets the requirements of environments as stipulated in Definition 1.

In the real world, i.e. in a run of $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, a)$, $\mathcal{E}$ will always receive what it expects because of the definition of $\mathcal{P}_{\mathrm{PKE}}$ and no party gets corrupted. The simulation of A_1 and A_2 is exactly like in the experiment $T_{A,\Sigma}$ of the definition of CCA-security. Thus, we have

$$\begin{aligned}
&\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 0] \\
&= \frac{1}{2}\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 0 \mid b = 0] \\
&\quad + \frac{1}{2}\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 0 \mid b = 1] \\
&= \frac{1}{2}\mathrm{Prob}[T_{A,\Sigma}(0,\eta) \rightsquigarrow 0] + \frac{1}{2}\mathrm{Prob}[T_{A,\Sigma}(1,\eta) \rightsquigarrow 1] \ .
\end{aligned} \tag{3}$$

On the other hand, in the ideal world, i.e. in a run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{PKE}}(1^\eta, \varepsilon)$, $\mathcal{E}$ outputs 1 with probability exactly one half. If $\mathcal{E}$ receives some unexpected input or if the decryptor or the encryptor with PID *pid* gets corrupted then this is clear by the definition of $\mathcal{E}$. Otherwise, $\mathcal{E}$ always receives what it expects and neither the decryptor nor the encryptor *pid* are corrupted. Thus, $\mathcal{E}$ outputs 1 iff $b \neq b'$. The ciphertext c^* only depends on $L_\eta(x_b)$. To see that it is statistically independent of b, we have to verify that the distributions of $L_\eta(x_0)$ and $L_\eta(x_1)$ are the same. Because $(L_\eta)_{\eta\in\mathbb{N}}$ leaks at most the length of a message, we can conclude that this is the case. Because c^* is never decrypted by $\mathcal{F}_{\mathrm{PKE}}$ and c^* is statistically independent of b, the probability that $b \neq b'$ is exactly one half:

$$\begin{aligned}
&\mathrm{Prob}[b \xleftarrow{\mathrm{R}} \{0,1\}, F \text{ outputs } 1-b] \\
&= \mathrm{Prob}[b \xleftarrow{\mathrm{R}} \{0,1\}, F \text{ outputs } 1, b = 0] + \mathrm{Prob}[b \xleftarrow{\mathrm{R}} \{0,1\}, F \text{ outputs } 0, b = 1] \\
&= \mathrm{Prob}[b \xleftarrow{\mathrm{R}} \{0,1\}, b = 0] \cdot \mathrm{Prob}[F \text{ outputs } 1] \\
&\quad + \mathrm{Prob}[b \xleftarrow{\mathrm{R}} \{0,1\}, b = 1] \cdot \mathrm{Prob}[F \text{ outputs } 0] \\
&= \frac{1}{2}(\mathrm{Prob}[F \text{ outputs } 1] + \mathrm{Prob}[F \text{ outputs } 0]) = \frac{1}{2} \ .
\end{aligned}$$

Therefore, we have

$$\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 1] = \frac{1}{2} \ . \tag{4}$$

By (3) and (4), we conclude that

$$\begin{aligned}
&|\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{PKE}}(1^\eta, \varepsilon) \rightsquigarrow 1]| \\
&= \left|\frac{1}{2}\mathrm{Prob}[T_{A,\Sigma}(0,\eta) \rightsquigarrow 0] + \frac{1}{2}\mathrm{Prob}[T_{A,\Sigma}(1,\eta) \rightsquigarrow 1] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}(1 - \mathrm{Prob}[T_{A,\Sigma}(0,\eta) \rightsquigarrow 1]) + \frac{1}{2}\mathrm{Prob}[T_{A,\Sigma}(1,\eta) \rightsquigarrow 1] - \frac{1}{2}\right| \\
&= \frac{1}{2}|\mathrm{Prob}[T_{A,\Sigma}(0,\eta) \rightsquigarrow 1] - \mathrm{Prob}[T_{A,\Sigma}(1,\eta) \rightsquigarrow 1]| = \frac{1}{2}\mathrm{Adv}(A, \Sigma, \eta)
\end{aligned}$$

which is by assumption non-negligible. This proves $\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}} \not\equiv^{noaux} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{PKE}}$. □

Note that the above theorem and lemma hold in particular for $(L_\eta)_{\eta\in\mathbb{N}} = (L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$, so, we obtain the following corollary.

Corollary 2. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes and let $\Sigma = (\mathrm{gen}, \mathrm{dec}, \mathrm{enc})$ be a p-bounded public-key encryption scheme with $\mathrm{Prob}[\mathrm{enc}(k, m) = \bot] = \mathrm{Prob}[L_\eta^{|\cdot|}(m) = \bot]$ for all k generated by $\mathrm{gen}(1^\eta)$ and all $m \in \{0,1\}^*$. Then, Σ is CCA-secure if and only if*

$$\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{PKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$$

where $(L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$ is the above defined leakage. Note that $\mathrm{dom}(L_\eta^{|\cdot|}) = \bigcup_{n\geq\eta}\{0,1\}^n$.

6.3 Joint State for Public-Key Encryption

In this section we present a protocol system $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ that realizes the multi-session multi-party version of $\mathcal{F}_{\mathrm{PKE}}$ by using only one copy of $\mathcal{F}_{\mathrm{PKE}}$ per party instead of one copy of $\mathcal{F}_{\mathrm{PKE}}$ per party *per session*.

The basic idea in realizing joint state for public-key encryption appears in [9] but no details or proof are given there (see 6.4). It is similar to the one for digital signatures: The session identifier (SID) *sid* is used as a prefix to the plaintext m prior to encryption, i.e. instead of encrypting m in session *sid* with a separate key for this session, (sid, m) is encrypted with the same key for each session. Upon decryption of a ciphertext c in session *sid* one has to check whether c decrypts to (sid, m) with the correct SID *sid*. While the main idea is simple, it only works given an appropriate formulation of the public-key encryption functionality (see also Section 6.4).

Next, we present our joint state realization of $\mathcal{F}_{\mathrm{PKE}}$, i.e. the formulation of the protocol system $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$. Recall that it uses $!\underline{\mathcal{F}_{\mathrm{PKE}}}$. More precisely, one copy of $\underline{\mathcal{F}_{\mathrm{PKE}}}$ per party is generated. The joint state theorem for public-key encryption is similar to the one for digital signatures, it basically says that for all leakage $(L_\eta)_{\eta\in\mathbb{N}}$

$$!\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \mid !\underline{\mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}})} \leq^{SS} \underline{!(\,!\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}})})}$$

where $\mathcal{F}'_{\mathrm{PKE}}$ is identical to $\mathcal{F}_{\mathrm{PKE}}$ except that tapes have been renamed and $(L'_\eta)_{\eta\in\mathbb{N}}$ leaks as much as $(L_\eta)_{\eta\in\mathbb{N}}$ but additionally reveals the SID of the session in which the message was encrypted. More formal, $(L'_\eta)_{\eta\in\mathbb{N}}$ is defined by $\{L'_\eta(x)\colon$ if x is of shape (sid, x') for some SID *sid* and bit string x and $x' \in \mathrm{dom}(L_\eta)$ then return $(sid, L_\eta(x'))$ else return $\bot\}$. It is easy to see that $\mathrm{dom}(L'_\eta) = \{(sid, x) \mid sid \in \{0,1\}^*$ and $x \in \mathrm{dom}(L_\eta)\}$. Note that leakage of the SID is not a security issue as in the ideal world $\underline{!(\,!\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}})})}$ may also reveal the SID in the ciphertexts because the adversary is able to provide different encryption algorithms for each session. As described in Section 3, on the right-hand side we have a multi-session multi-party version, i.e. the inner part $!\underline{\mathcal{F}_{\mathrm{PKE}}}$ is the multi-party version of $\mathcal{F}_{\mathrm{PKE}}$ where we have one copy of $\underline{\mathcal{F}_{\mathrm{PKE}}}$ per party and the outer part is the multi-session version of the multi-party version of $\mathcal{F}_{\mathrm{PKE}}$. Thus, we have one copy of $\mathcal{F}_{\mathrm{PKE}}$ per session per party. Note that $\underline{!(\,!\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}})})}$ behaves exactly like $!\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}})}$. On the other side, we have only the multi-party version $!\underline{\mathcal{F}'_{\mathrm{PKE}}}$ and the protocol $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ where there will be one copy of $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ for each party which is the protocol that the party runs and through which its multi-party version of $\mathcal{F}'_{\mathrm{PKE}}$ is accessible. Note that on the left-hand side, we have one copy of $\mathcal{F}'_{\mathrm{PKE}}$ per party which is used in every session the party is involved in.

The realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ is parameterized, just like $\mathcal{F}_{\mathrm{PKE}}$, by two disjoint sets of names of tapes $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$, and two polynomials p and q and is defined as the composition of two IITMs:

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(\mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p, q) = P^{\mathrm{JS}}_{\mathrm{dec}}(\mathcal{T}_{\mathrm{dec}}, p, q) \mid P^{\mathrm{JS}}_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}}, p, q)$$

where the IITMs are defined in Figure 24 and 25. A graphical representation and its connection to $\underline{\mathcal{F}'_{\mathrm{PKE}}}$ is pictured in Figure 26.

The problem that occurs with long SIDs in case of digital signatures also occurs here. Thus, as described in Section 5.3, $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ restricts SIDs to be not longer than $q(\eta)$ where η is the security parameter. As argued, this does not restrict the usability and expressiveness of $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ since no protocol, environment, or adversary that is polynomially bounded by η could create exponentially many different sessions. As already noted for digital signatures (Section 5.3), a more complex definition of polynomial time might solve this particular inconvenience. However, one cannot dispense with parameters altogether.

Also similar to the case of digital signatures, one technical problem is that of multiple initialization requests where a functionality has to wait for a response from the environment/adversary. For example, if the environment sent an initialization request in session *sid* but has not completed it yet, i.e. the environment/adversary has not yet sent the algorithms and a public key, then, if the environment sends

Realization $P^{\mathrm{JS}}_{\mathrm{dec}}(\mathcal{T}_{\mathrm{dec}}, p, q)$

I/O-tapes: in: $\mathrm{io}(T, \mathrm{pke})$ for each $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke})$ (enriching), $\mathrm{io}(\mathrm{pke}', T)$ for each $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathrm{io}(\mathrm{pke}', E_{\mathrm{dec}})$ (consuming)
out: $\mathrm{io}(\mathrm{pke}, T)$ for each $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathrm{io}(\mathrm{pke}, E_{\mathrm{dec}})$, $\mathrm{io}(T, \mathrm{pke}')$ for each $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke}')$

Initialization: $pid \leftarrow \bot$; $Sids \leftarrow \emptyset$; $AllSids \leftarrow \emptyset$; $lastSid \leftarrow \bot$; $res \leftarrow 0$; $k \leftarrow \bot$; $state \leftarrow \mathsf{init}$

CheckAddress: Upon first activation accept only messages of shape (sid, pid', m) on all pke-tapes (i.e. on $\mathrm{io}(T, \mathrm{pke})$, $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke})$). In mode $\mathsf{Compute}$ at first record the PID: $pid \leftarrow pid'$.
On later activations accept only messages of shape (sid, pid', m) on all pke-tapes and of shape (pid', m) on all pke′-tapes where $pid' = pid$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, \mathsf{Init})$ on $\mathrm{io}(T, \mathrm{pke})$ where $T \in \mathcal{T}_{\mathrm{dec}}$, $sid \notin AllSids$ do:
$AllSids \leftarrow AllSids \cup \{sid\}$;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then
  produce no output
else if $state = \mathsf{wait}$ then
  $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
  $state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid, Init) to $\mathrm{io}(T, \mathrm{pke}')$
else
  $Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, \mathsf{PublicKey}, k)$ to $\mathrm{io}(\mathrm{pke}, T)$
end

(b) *Initialization Response:* If recv $(pid, \mathsf{PublicKey}, k')$ on $\mathrm{io}(\mathrm{pke}', T)$ where $T \in \mathcal{T}_{\mathrm{dec}}$, $|k'| \leq p(\eta)$, and $state = \mathsf{wait}$ do: $state \leftarrow \mathsf{ok}$; $k \leftarrow k'$; $Sids \leftarrow \{lastSid\}$; send $(lastSid, pid, \mathsf{PublicKey}, k)$ to $\mathrm{io}(\mathrm{pke}, T)$

(c) *Forward to* pke*:* If recv (sid, pid, m) on $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke})$ or $\mathrm{io}(T, \mathrm{pke})$ where $T \in \mathcal{T}_{\mathrm{dec}}$, $state = \mathsf{ok}$, $sid \in Sids$, and $m \neq \mathsf{Corrupted}?$ do:
$res \leftarrow res + p(|(sid, pid, m)| + \eta)$; $lastSid \leftarrow sid$; send (pid, m) to $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke}')$ or $\mathrm{io}(T, \mathrm{pke}')$ (resp.)

(d) *Forward from* pke*:* If recv (pid, m) on $\mathrm{io}(\mathrm{pke}', T)$ where $T \in \mathcal{T}_{\mathrm{dec}}$, $state = \mathsf{ok}$, $|(lastSid, pid, m)| \leq res$ do:

$$m' \leftarrow \begin{cases} (\mathsf{Plaintext}, m'') & \text{if } m = (\mathsf{Plaintext}, (lastSid, m'')) \\ (\mathsf{Plaintext}, \bot) & \text{if } m = (\mathsf{Plaintext}, (sid, m'')) \text{ with } sid \neq lastSid \\ m & \text{otherwise ;} \end{cases}$$

$res \leftarrow res - |(lastSid, pid, m)|$; send $(lastSid, pid, m')$ to $\mathrm{io}(\mathrm{pke}, T)$

(e) *Corruption Request:* If recv $(sid, pid, \mathsf{Corrupted}?)$ on $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke})$ where $sid \in AllSids$ do:
send $(pid, \mathsf{Corrupted}?)$ to $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke}')$; wait for recv (pid, b) on $\mathrm{io}(\mathrm{pke}', E_{\mathrm{dec}})$;
send (sid, pid, b) to $\mathrm{io}(\mathrm{pke}, E_{\mathrm{dec}})$

If no rule above fires then produce no output.

Fig. 24. Joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} = P^{\mathrm{JS}}_{\mathrm{dec}} \,|\, P^{\mathrm{JS}}_{\mathrm{enc}}$ for public-key encryption, the decryptor's part $P^{\mathrm{JS}}_{\mathrm{dec}}$.

Realization $P^{\mathrm{JS}}_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}}, p, q)$

I/O-tapes: in: $\mathrm{io}(T, \mathrm{pke})$ for each $T \in \mathcal{T}_{\mathrm{enc}}$, $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke})$ (enriching), $\mathrm{io}(\mathrm{pke}', T)$ for each $T \in \mathcal{T}_{\mathrm{enc}}$, $\mathrm{io}(\mathrm{pke}', E_{\mathrm{enc}})$ (consuming)
out: $\mathrm{io}(\mathrm{pke}, T)$ for each $T \in \mathcal{T}_{\mathrm{enc}}$, $\mathrm{io}(\mathrm{pke}, E_{\mathrm{enc}})$, $\mathrm{io}(T, \mathrm{pke}')$ for each $T \in \mathcal{T}_{\mathrm{enc}}$, $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke}')$

Initialization: $pid, pid' \leftarrow \bot$; $Sids \leftarrow \emptyset$; $AllSids \leftarrow \emptyset$; $lastSid \leftarrow \bot$; $res \leftarrow 0$; $state \leftarrow \mathsf{init}$

CheckAddress: Upon first activation accept msgs of shape (sid, pid_1, pid_2, m) on all pke-tapes (i.e. on $\mathrm{io}(T, \mathrm{pke})$, $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke})$). In mode $\mathsf{Compute}$ at first record the PIDs: $pid \leftarrow pid_1$ and $pid' \leftarrow pid_2$. On later activations accept msgs of shape (sid, pid_1, pid_2, m) on all pke-tapes and of shape (pid_1, pid_2, m) on all pke′-tapes where $pid_1 = pid$ and $pid_2 = pid'$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, pid', \mathsf{Init})$ on $\mathrm{io}(T, \mathrm{pke})$ where $T \in \mathcal{T}_{\mathrm{enc}}$, $sid \notin AllSids$ do:
$AllSids \leftarrow AllSids \cup \{sid\}$;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then
  produce no output
else if $state = \mathsf{wait}$ then
  $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
  $state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send $(pid, pid', \mathsf{Init})$ to $\mathrm{io}(T, \mathrm{pke}')$
else
  $Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, pid', \mathsf{Inited})$ to $\mathrm{io}(\mathrm{pke}, T)$
end

(b) *Initialization Response:* If recv $(pid, pid', \mathsf{Inited})$ on $\mathrm{io}(\mathrm{pke}', T)$ where $T \in \mathcal{T}_{\mathrm{enc}}$, and $state = \mathsf{wait}$ do:
$state \leftarrow \mathsf{ok}$; $Sids \leftarrow \{lastSid\}$; send $(lastSid, pid, pid', \mathsf{Inited})$ to $\mathrm{io}(\mathrm{pke}, T)$

(c) *Forward to* pke*:* If recv (sid, pid, pid', m) on $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke})$ or $\mathrm{io}(T, \mathrm{pke})$ where $T \in \mathcal{T}_{\mathrm{enc}}$, $state = \mathsf{ok}$, $sid \in Sids$, and $m \neq \mathsf{Corrupted}?$ do:
$res \leftarrow res + p(|(sid, pid, pid', m)| + \eta)$; $lastSid \leftarrow sid$;
$$m' \leftarrow \begin{cases} (\mathsf{Encrypt}, k, (sid, m'')) & \text{if } m = (\mathsf{Encrypt}, k, m'') \\ m & \text{otherwise} \end{cases} ;$$
send (pid, pid', m') to $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke}')$ or $\mathrm{io}(T, \mathrm{pke}')$ (resp.)

(d) *Forward from* pke*:* If recv $m = (pid, pid', m')$ on $\mathrm{io}(\mathrm{pke}', T)$ where $T \in \mathcal{T}_{\mathrm{enc}}$, $state = \mathsf{ok}$, and $|(lastSid, m)| \leq res$ do: $res \leftarrow res - |(lastSid, m)|$; send $(lastSid, pid, pid', m')$ to $\mathrm{io}(\mathrm{pke}, T)$

(e) *Corruption Request:* If recv $(sid, pid, pid', \mathsf{Corrupted}?)$ on $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke})$ where $sid \in AllSids$ do:
send $(pid, pid', \mathsf{Corrupted}?)$ to $\mathrm{io}(E_{\mathrm{enc}}, \mathrm{pke}')$; wait for recv (pid, pid', b) on $\mathrm{io}(\mathrm{pke}', E_{\mathrm{enc}})$;
send (sid, pid, pid', b) to $\mathrm{io}(\mathrm{pke}, E_{\mathrm{enc}})$

If no rule above fires then produce no output.

Fig. 25. Joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} = P^{\mathrm{JS}}_{\mathrm{dec}} \,|\, P^{\mathrm{JS}}_{\mathrm{enc}}$ for public-key encryption, the encryptor's part $P^{\mathrm{JS}}_{\mathrm{enc}}$.

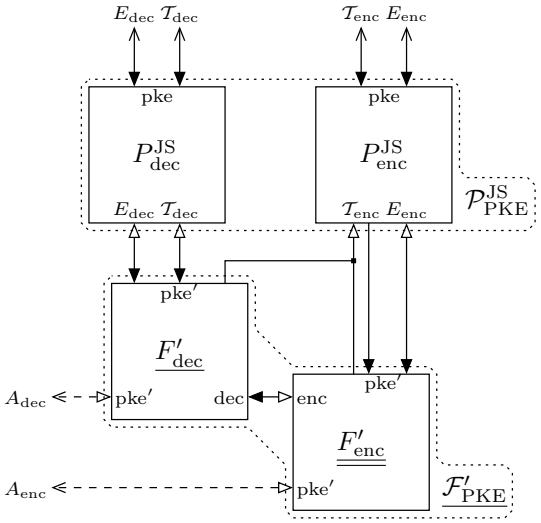

Fig. 26. Graphical representation of the joint state realization $\mathcal{P}^{\text{JS}}_{\text{PKE}} = P^{\text{JS}}_{\text{dec}} \mid P^{\text{JS}}_{\text{enc}}$ and its connection to $!\underline{\mathcal{F}'_{\text{PKE}}}$. See Figure 18 for a legend.

another initialization request in another session sid' for the same party the question is how to deal with this situation. The joint state realization cannot further process this request because it already waits for an answer from the functionality in the first request. (Note that for both requests the same functionality is used in the joint state world). Conversely, the joint state realization also cannot simply ignore this particular request by the environment because a simulator could not reproduce this behavior. Therefore, we define the joint state realization $\mathcal{P}^{\text{JS}}_{\text{PKE}}$ to enter and stay in an error state in which any further messages (except for (Corrupted?) from the environment) are ignored. In other words, the joint state realization stops any further activity. This is not a real limitation as it only forces the environment/adversary to complete initialization requests at once, which in any realistic implementation of the public-key functionality is done anyway because initialization requests are answered immediately—a key pair is generated and the public key is returned right away.

A more detailed description of $\mathcal{P}^{\text{JS}}_{\text{PKE}}$ is given next.

There will be one instance of $P^{\text{JS}}_{\text{dec}} = P^{\text{JS}}_{\text{dec}}(\mathcal{T}_{\text{dec}}, p, q)$ for each decryptor. We denote the instance of a party (with PID) pid by $P^{\text{JS}}_{\text{dec}}[pid]$. Additionally, there will be one instance of $P^{\text{JS}}_{\text{enc}} = P^{\text{JS}}_{\text{enc}}(\mathcal{T}_{\text{enc}}, p, q)$ for each encryptor pid' who encrypts messages for the decryptor pid which we denote by $P^{\text{JS}}_{\text{enc}}[pid, pid']$.

A decryptor pid in session sid sends messages of shape (sid, pid, m) to the instance $P^{\text{JS}}_{\text{dec}}[pid]$. An encryptor pid' in session sid sends messages of shape (sid, pid, pid', m) to the instance $P^{\text{JS}}_{\text{enc}}[pid, pid']$ when he wants to encrypt messages for party pid. No party other than pid will communicate with $P^{\text{JS}}_{\text{dec}}[pid]$, so it is like a local procedure and we call party pid the *owner* of $P^{\text{JS}}_{\text{dec}}[pid]$. Similarly, no party other than pid' will communicate with $P^{\text{JS}}_{\text{enc}}[pid, pid']$ and party pid' is called the *owner* of $P^{\text{JS}}_{\text{enc}}[pid, pid']$.

A decryptor pid has to register in each session sid with $P^{\text{JS}}_{\text{dec}}[pid]$, i.e. send $(sid, pid, \mathsf{Init})$. However, only the first time $P^{\text{JS}}_{\text{dec}}[pid]$ creates an instance of $\underline{F'_{\text{dec}}}$ by stripping off the SID and forwarding the message (pid, Init). To refer to the created instance it is denoted by $F'_{\text{dec}}[pid]$. The polynomial q is used to forbid sessions with *long* SIDs ($sid > q(\eta)$). See Figure 24 (a) and (b). If the same decryptor pid sends a second initialization request before the first one has been completed, $P^{\text{JS}}_{\text{dec}}[pid]$ will enter an error state and ignore all messages send from the signer pid or from $F'_{\text{dec}}[pid]$ (except for (Corrupted?) from the environment).

Analogously, an encryptor pid' registers with $P^{\text{JS}}_{\text{enc}}[pid, pid']$. Here, one instance of $\underline{\underline{F'_{\text{enc}}}}$, denoted by $F'_{\text{enc}}[pid, pid']$, is created. See Figure 25 (a) and (b).

After initialization, every message is forwarded by $P^{\text{JS}}_{\text{dec}}[pid]$ and $P^{\text{JS}}_{\text{enc}}[pid, pid']$ to $F'_{\text{dec}}[pid]$ and $F'_{\text{enc}}[pid, pid']$ (respectively) by stripping off the SID. An encryption request $(sid, pid, pid', \mathsf{Encrypt}, k, m)$ is forwarded as $(pid, pid', \mathsf{Encrypt}, k, (sid, m))$. If upon decryption a plaintext is received, i.e. $P^{\text{JS}}_{\text{dec}}[pid]$ receives the message $(pid, \mathsf{Plaintext}, m)$ on tape $\text{io}(\text{pke}', T)$, then the message $(sid, pid, \mathsf{Plaintext}, m')$ is

forwarded if m is of shape (sid, m') and the message $(sid, pid, \mathsf{Plaintext}, \bot)$ is forwarded otherwise where sid is the recorded SID with that the decryption request was made.

Since the system has to be well-formed the tapes from $\underline{\mathcal{F}'_{\mathrm{PKE}}}$ to $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ have to be consuming. Hence, forwarding messages from $\underline{\mathcal{F}'_{\mathrm{PKE}}}$ to a party can not be done arbitrarily. Therefore, $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$ records the length of the messages sent to $\underline{\mathcal{F}'_{\mathrm{PKE}}}$ and only forwards polynomially many input from $\underline{\mathcal{F}'_{\mathrm{PKE}}}$. However, note that by the definition of $\mathcal{F}_{\mathrm{PKE}}(p)$ every message that is output by $\mathcal{F}_{\mathrm{PKE}}(p)$ has at most length $p(|m|+\eta)$ where m is the input that $\mathcal{F}_{\mathrm{PKE}}(p)$ received before. This holds as well if $\mathcal{F}_{\mathrm{PKE}}(p)$ is corrupted because the resources are passed through $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}$. Thus, when $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p,q)$ and $\underline{\mathcal{F}'_{\mathrm{PKE}}(p)}$ are composed, $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p,q)$ will always be able to forward messages from $\underline{\mathcal{F}'_{\mathrm{PKE}}(p)}$.

Next, we state and prove the joint state theorem for public-key encryption. Note that $\mathcal{F}_{\mathrm{PKE}}$ is defined such that the decryptor is adaptive corruptible, so, the joint state theorem holds as well for adaptively corruptible decryptors. Furthermore, the proof reveals that the theorem even holds for unbounded environments and perfect indistinguishability, i.e., there exists a simulator $\mathcal{S}$ such that for all (unbounded) environments $\mathcal{E}$ it holds $\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \,|\, !\underline{\mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}})}(1^\eta, a) \rightsquigarrow 1] = \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}})}}(1^\eta, a) \rightsquigarrow 1]$ for all $\eta \in \mathbb{N}, a \in \{0,1\}^*$.

Theorem 8. *For all polynomials p and q and disjoint sets of names of tapes $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ there exists a polynomial p' such that for all leakage $(L_\eta)_{\eta\in\mathbb{N}}$ we have*

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(\mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p, q) \,|\, !\underline{\mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)} \leq^{SS} !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p')}}$$

where $(L'_\eta)_{\eta\in\mathbb{N}}$ is defined by

$$L'_\eta(x)\colon \textit{if } \exists sid \in \{0,1\}^*, x' \in \mathrm{dom}(L_\eta)\colon x \textit{ is of shape } (sid, x') \textit{ then return } (sid, L_\eta(x'))$$
$$\textit{else return } \bot$$

and $\mathcal{F}'_{\mathrm{PKE}}$ is obtained from $\mathcal{F}_{\mathrm{PKE}}$ by replacing pke *by* pke$'$ *in all tape names.*

Proof. Below, we define a simulator $\mathcal{S}$ such that the joint state world (JS world), i.e., $\mathcal{E} \,|\, \mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \,|\, !\underline{\mathcal{F}'_{\mathrm{PKE}}}$, is perfectly indistinguishable from the ideal world, i.e. $\mathcal{E} \,|\, \mathcal{S} \,|\, !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}}}$, for every environment $\mathcal{E}$. When the environment presents algorithms d, e and a key k then the simulator $\mathcal{S}$ forwards d_{sid}, e_{sid} and k. The definition of d_{sid} and e_{sid} (which is given below) will yield the definition of p'. Note that p' will be independent from the leakage $(L_\eta)_{\eta\in\mathbb{N}}$.

Let $\eta \in \mathbb{N}$, sid be an SID with $|sid| \leq q(\eta)$, and d and e be descriptions of algorithms with $|d| \leq p(\eta)$ and $|e| \leq p(\eta)$. Depending on η, sid, d, e and p, we define the algorithms d_{sid} and e_{sid} as follows:

- Algorithm $d_{sid}(c)$ computes $m \leftarrow d(c)$ and counts the steps needed. If these are more than $p(|c|+\eta)$ then enter an infinite loop else if $m = (sid, m')$ for some $m' \in \{0,1\}^*$ then return m' else return $\bot$.
- Algorithm $e_{sid}(k, m)$ computes $c \leftarrow e(k, (sid, m))$ counting the steps needed. If these are at most $p(|(sid, m)| + \eta)$ return c else enter an infinite loop.

Since the length of the description of d and e and the length of sid is polynomially bounded by η, the length of the description of d_{sid} and e_{sid} is polynomially bounded by η. Moreover, if the runtime of $d(c)$ and $e(k, (sid, m))$ is bounded by $p(|c|+\eta)$ and $p(|(sid, m)| + \eta)$ (resp.) then the runtime of $d_{sid}(c)$ and $e_{sid}(k, m)$ (except when they enter an infinite loop) is polynomial in $|c|+\eta$ and $|m|+\eta$, respectively. Thus, we find a polynomial p' such that

1. for all $\eta \in \mathbb{N}$ we have that if $|d| \leq p(\eta)$ and $|e| \leq p(\eta)$ then $|d_{sid}| \leq p'(\eta)$ and $|e_{sid}| \leq p'(\eta)$,
2. for all ciphertexts c and $\eta \in \mathbb{N}$ we have that the computation of $d(c)$ exceeds $p(|c|+\eta)$ steps iff the computation of $d_{sid}(c)$ exceeds $p'(|c|+\eta)$ steps, and
3. for all plaintexts $m \in \{0,1\}^*$, keys k and $\eta \in \mathbb{N}$ we have that the computation of $e(k, (sid, m))$ exceeds $p(|(sid, m)| + \eta)$ steps iff the computation of $e_{sid}(k, m)$ exceeds $p'(|m| + \eta)$ steps.

A formal definition of the simulator

$$!\mathcal{S} = \mathcal{S}^{\mathrm{JS}}_{\mathrm{PKE}}(p, q) = !S^{\mathrm{JS}}_{\mathrm{dec}}(p, q) \,|\, !S^{\mathrm{JS}}_{\mathrm{enc}}(q)$$

is given in Figure 27 and 28. A graphical representation of $\mathcal{S}^{\text{JS}}_{\text{PKE}}$ and its connection to $!\underline{\underline{\mathcal{F}_{\text{PKE}}}}$ is pictured in Figure 29. The technical part is similar to the simulator $\mathcal{S}^{\text{JS}}_{\text{SIG}}$ in the proof of the joint state theorem for digital signatures (Theorem 6). The only difference is that instead of the algorithms s_{sid} and v_{sid} the algorithms e_{sid} and d_{sid} (as defined above) are used. So, we refer to Section 5.3 for a description of the simulator.

We abbreviate the systems of IITMs $\mathcal{P} = \mathcal{P}^{\text{JS}}_{\text{PKE}}(\mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p, q) \mid !\underline{\mathcal{F}'_{\text{PKE}}((L'_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p)}$ and $\mathcal{F} = !\underline{\underline{\mathcal{F}_{\text{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p')}}$. It is easy to see that $!\mathcal{S}$ is adversarially connectible for $\mathcal{F}$ and that $\mathcal{P}$ and $!\underline{\underline{\mathcal{S} \mid \mathcal{F}}}$ are compatible. It remains to show that $\mathcal{E} \mid \mathcal{P}$ and $\mathcal{E} \mid !\mathcal{S} \mid !\mathcal{F}$ are perfectly indistinguishable. The proof shows that the two systems behave exactly the same, in particular no reasoning about probabilities is involved. The main part is to show that collisions of ciphertexts between different sessions do not occur in the JS world (see below).

By the same arguments as in the proof of the joint state theorem for digital signatures (Theorem 6) we conclude that there is no difference between the JS world and the ideal world according to the behavior of initialization, corruption and key generation. For corruption we note that if a decryptor with PID pid gets corrupted then in the JS world H_{pid} is returned where the elements of this set look like $((sid, m), c)$. On the other hand in the ideal world, the simulator receives the sets $H_{sid,pid}$ for all recorded SIDs sid where the elements do not include the SID, i.e., look like (m, c). The simulator returns $H_{pid} = \bigcup_{sid}\{((sid, m), c) \mid (m, c) \in H_{sid,pid}\}$ and because the representation of H_{pid} does not reveal the order in which elements have been added, e.g., a possible representation would be a list of the elements in lexicographical order, the environment can not distinguish from JS world and ideal world upon corruption.

Consider encryption of m with key k' for a decryptor pid in session sid. If $k' \neq k$ then $F'_{\text{dec}}[pid]$ in the JS world computes $c \leftarrow e(k', (sid, m))$ while $F_{\text{dec}}[sid, pid]$ in the ideal world computes $c \leftarrow e_{sid}(k', m)$. By the definition of p' and e_{sid}, the output in the JS world has the same distribution as the output in the ideal world.

On the other hand if $k' = k$, $F'_{\text{dec}}[pid]$ in the JS world computes $m'' \leftarrow L'_\eta((sid, m)); c \leftarrow e(k, m'')$ and tests $d(c) = m''$ while $F_{\text{dec}}[sid, pid]$ computes $m' \leftarrow L_\eta(m); c \leftarrow e_{sid}(k, m')$ and tests $d_{sid}(c) = m'$ in the ideal world. Since the distribution of m'' equals the one of m' if we would prefix m' by sid (by definition of L'_η), the computed ciphertext c in the JS world has the same distribution as c in the ideal world. By the definition of d_{sid}, we have $d(c) = m''$ if and only if $d_{sid}(c) = m'$. Thus, the output in the JS world has the same distribution as the output in the ideal world. Another consequence is that

$$\text{Prob}[((sid, m), c) \in H_{pid}] = \text{Prob}[(m, c) \in H_{sid,pid}] \tag{5}$$

for all sid, m and c where H_{pid} is the set H in the copy $F'_{\text{PKE}}[pid]$ in the JS world and $H_{sid,pid}$ is the set H in the copy $F_{\text{PKE}}[sid, pid]$ in the ideal world.

Now, we consider decryption of a ciphertext c by a decryptor pid in session sid.

If $d(c)$ is not of shape (sid', m) for all SIDs sid' and m then $d_{sid}(c) = \bot$ and in the ideal world the error symbol $\bot$ is returned just as in the JS world.

If $d(c) = (sid', m)$ for some plaintext m but a different SID $sid' \neq sid$ then again $d_{sid}(c) = \bot$ and in the ideal world the error symbol $\bot$ is returned. We analyze the behavior in the JS world and show that here $\bot$ is returned as well. Assume that in the JS world $\bot$ is not returned. Then, c is not ambiguous in H_{pid} and there exists a plaintext m' such that $((sid, m'), c) \in H_{pid}$. Because of the decryption test upon encryption, $d(c)$ has to be prefixed by sid which contradicts with $sid \neq sid'$.

Now consider the last case, i.e. that $d(c) = (sid, m)$ for some plaintext m. We have $d_{sid}(c) = m$, so, in the ideal world the error symbol $\bot$ is returned iff c is ambiguous in $H_{sid,pid}$. Otherwise, m or the message recorded in $H_{sid,pid}$ is returned. Assume that $((sid', m'), c) \in H_{pid}$ for some plaintext m' and $sid' \neq sid$. Then, again $d(c)$ has to be prefixed by sid which contradicts with $sid \neq sid'$. Therefore and because $d(c) = (sid, m)$, in the JS world $\bot$ is returned iff c is ambiguous in H_{pid}. Otherwise, if $((sid, m'), c) \in H_{pid}$ then m' is returned else m is returned.

Assume that c is ambiguous in the JS world. Then, there are messages m_1, m_2 and SIDs sid_1, sid_2 such that $(sid_1, m) \neq (sid_2, m_2)$ and $((sid_1, m_1), c)$, $((sid_2, m_2), c) \in H_{pid}$ and upon encryption the decryption tests where both positive. Thus, $d(c)$ is prefixed by some sid. Since d is deterministic, $sid = sid_1 = sid_2$ and $m_1 \neq m_2$. In other words, ciphertexts recorded in different sessions can not be equal. Therefore, the probability that c is ambiguous in H_{pid} is equal to the probability that c is ambiguous in $H_{sid',pid}$ where sid' is the SID that is revealed by $d(c)$.

Simulator $S^{\mathrm{JS}}_{\mathrm{dec}}(p,q)$

net-tapes: in: $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke}')$, $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$, $\mathrm{net}(\mathrm{enc},\mathrm{dec})$ (enriching)
out: $\mathrm{net}(\mathrm{pke}',A_{\mathrm{dec}})$, $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$, $\mathrm{net}(\mathrm{dec},\mathrm{enc})$

Initialization: $pid, lastSid, e, d, k \leftarrow \perp$; $state \leftarrow \mathsf{init}$; $nokey \leftarrow \mathsf{true}$; $corrupted \leftarrow \mathsf{false}$;
$Sids, AllSids, KeySids \leftarrow \emptyset$

CheckAddress: Upon first activation accept only messages of shape (sid, pid', m) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$ and of shape (pid', m) on $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke}')$. In mode $\mathsf{Compute}$ at first record the PID: $pid \leftarrow pid'$.
On later activations accept only messages of shape (sid, pid', m) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$ and of shape (pid', m) on $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke}')$ where $pid' = pid$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, \mathsf{Init})$ on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$ do: $AllSids \leftarrow AllSids \cup \{sid\}$;
if *corrupted* then
send $(sid, pid, \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$; wait for recv $(sid, pid, \mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$
end ;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then produce no output
else if $state = \mathsf{wait}$ then $state \leftarrow \mathsf{error}$; produce no output ;
else if $state = \mathsf{init}$ then
if *nokey* then $KeySids \leftarrow KeySids \cup \{sid\}$
else send $(sid, pid, \mathsf{AlgorithmsAndKey}, e_{sid}, d_{sid}, k)$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$ [a] ;
wait for recv (sid, pid, Ack) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$
end ;
$state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid, Inited) to $\mathrm{net}(\mathrm{pke}',A_{\mathrm{dec}})$
else $Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, \mathsf{AlgorithmsAndKey}, e_{sid}, d_{sid}, k)$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$;
wait for recv (sid, pid, Ack) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$; send $(sid, pid, \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$
end

(b) *Initialization Response:* If recv (pid, Inited) on $\mathrm{net}(A'_{\mathrm{dec}},\mathrm{pke})$, $state = \mathsf{wait}$, and not *nokey* do:
$state \leftarrow \mathsf{ok}$; send $(lastSid, pid, \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$

(c) *Key Gen.:* If recv $(pid, \mathsf{AlgorithmsAndKey}, e', d', k')$ on $\mathrm{net}(A'_{\mathrm{dec}},\mathrm{pke})$, *nokey*, and $|e'|, |d'|, |k'| \le p(\eta)$ do:
$(e, d, k) \leftarrow (e', d', k')$; $nokey \leftarrow \mathsf{false}$;
for all $sid \in KeySids$ do
send $(sid, pid, \mathsf{AlgorithmsAndKey}, e_{sid}, d_{sid}, k)$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$;
wait for recv (sid, pid, Ack) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$
end ;
send (pid, Ack) to $\mathrm{net}(\mathrm{pke}',A_{\mathrm{dec}})$

(d) *Corruption:* If recv $(pid, \mathsf{Corrupt})$ on $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke}')$, $state \neq \mathsf{init}$, and not *corrupted* do: $H \leftarrow \emptyset$;
for all $sid \in AllSids$ do
send $(sid, pid, \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$; wait for recv $(sid, pid, \mathsf{Corrupted}, H')$ on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$;
$H \leftarrow H \cup \{((sid, m), c) \mid (m, c) \in H'\}$
end ;
$corrupted \leftarrow \mathsf{true}$; send $(pid, \mathsf{Corrupted}, H)$ to $\mathrm{net}(\mathrm{pke}',A_{\mathrm{dec}})$

(e) *Forward to A:* If recv (sid, pid, m) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$, $m \neq \mathsf{Init}$, $state \neq \mathsf{error}$, and $sid \in Sids$ do:
$lastSid \leftarrow sid$; send (pid, m) to $\mathrm{net}(\mathrm{pke}',A_{\mathrm{dec}})$

(f) *Forward from A:* If recv $(pid, \mathsf{Send}, m, T)$ on $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke}')$, and $state \neq \mathsf{error}$ do:

$$m' \leftarrow \begin{cases} (\mathsf{Plaintext}, m'') & \text{if } m = (\mathsf{Plaintext}, (lastSid, m'')) \\ (\mathsf{Plaintext}, \perp) & \text{if } m = (\mathsf{Plaintext}, (sid, m'')) \text{ with } sid \neq lastSid \\ m & \text{otherwise ;} \end{cases}$$

send $(lastSid, pid, \mathsf{Send}, m', T)$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$

(g) If recv $(pid, pid', \mathsf{SendAlgAndKey}, sid)$ on $\mathrm{net}(\mathrm{enc},\mathrm{dec})$ do:
if *nokey* then $KeySids \leftarrow KeySids \cup \{sid\}$
else send $(sid, pid, \mathsf{AlgorithmsAndKey}, e_{sid}, d_{sid}, k)$ to $\mathrm{net}(A_{\mathrm{dec}},\mathrm{pke})$;
wait for recv (sid, pid, Ack) on $\mathrm{net}(\mathrm{pke},A_{\mathrm{dec}})$
end ; send $(pid, pid', \mathsf{Ack})$ to $\mathrm{net}(\mathrm{dec},\mathrm{enc})$

If no rule above fires then produce no output.

[a] where d_{sid} and e_{sid} are basically defined by $e_{sid}(k', m)$: return $e(k', (sid, m))$ and $d_{sid}(c)$: if $d(c) = (sid, m')$ then return m' else return $\perp$ (see formal definitions in the proof of Theorem 8)

Fig. 27. Simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{PKE}} = !S^{\mathrm{JS}}_{\mathrm{dec}} \mid !S^{\mathrm{JS}}_{\mathrm{enc}}$ for the proof of the joint state theorem for public-key encryption, the decryptor's part $S^{\mathrm{JS}}_{\mathrm{dec}}$.

Simulator $S^{\mathrm{JS}}_{\mathrm{enc}}(q)$

net-tapes: in: $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke}')$, $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ (enriching), $\mathrm{net}(\mathrm{dec}, \mathrm{enc})$ (consuming)
out: $\mathrm{net}(\mathrm{pke}', A_{\mathrm{enc}})$, $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$, $\mathrm{net}(\mathrm{enc}, \mathrm{dec})$

Initialization: $pid, pid', lastSid \leftarrow \bot$; $state \leftarrow \mathsf{init}$; $corrupted \leftarrow \mathsf{false}$; $Sids, AllSids \leftarrow \emptyset$

CheckAddress: Upon first activation accept msgs of shape (sid, pid_1, pid_2, m) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ and of shape (pid_1, pid_2, m) on $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke}')$. In mode Compute at first record PIDs: $pid \leftarrow pid_1$, $pid' \leftarrow pid_2$. On later activations accept msgs of shape (sid, pid_1, pid_2, m) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ and of shape (pid_1, pid_2, m) on $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke}')$ where $pid_1 = pid$, $pid_2 = pid'$.

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv $(sid, pid, pid', \mathsf{Init})$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$ do: $AllSids \leftarrow AllSids \cup \{sid\}$;
if *corrupted* then
send $(sid, pid, pid', \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$; wait for recv $(sid, pid, pid', \mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$
end ;
if $state = \mathsf{error}$ or $|sid| > q(\eta)$ then produce no output
else if $state = \mathsf{wait}$ then $state \leftarrow \mathsf{error}$; produce no output
else if $state = \mathsf{init}$ then
send $(pid, pid', \mathsf{SendAlgAndKey}, sid)$ to $\mathrm{net}(\mathrm{enc}, \mathrm{dec})$; wait for recv $(pid, pid', \mathsf{Ack})$ on $\mathrm{net}(\mathrm{dec}, \mathrm{enc})$; $state \leftarrow \mathsf{wait}$; $lastSid \leftarrow sid$; send (pid, Inited) to $\mathrm{net}(\mathrm{pke}', A_{\mathrm{enc}})$
else send $(pid, pid', \mathsf{SendAlgAndKey}, sid)$ to $\mathrm{net}(\mathrm{enc}, \mathrm{dec})$; wait for recv $(pid, pid', \mathsf{Ack})$ on $\mathrm{net}(\mathrm{dec}, \mathrm{enc})$; $Sids \leftarrow Sids \cup \{sid\}$; send $(sid, pid, pid', \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$
end

(b) *Initialization Response:* If recv $(pid, pid', \mathsf{Inited})$ on $\mathrm{net}(A'_{\mathrm{enc}}, \mathrm{pke})$, and $state = \mathsf{wait}$ do:
$state \leftarrow \mathsf{ok}$; send $(lastSid, pid, pid', \mathsf{Inited})$ to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$

(c) *Corruption:* If recv $(pid, pid', \mathsf{Corrupt})$ on $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke}')$, $state \neq \mathsf{init}$, and not *corrupted* do:
for all $sid \in AllSids$ do
send $(sid, pid, pid', \mathsf{Corrupt})$ to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$; wait for recv $(sid, pid, pid', \mathsf{Corrupted})$ on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$
end ;
$corrupted \leftarrow \mathsf{true}$; send $(pid, pid', \mathsf{Corrupted})$ to $\mathrm{net}(\mathrm{pke}', A_{\mathrm{enc}})$

(d) *Forward to A:* If recv (sid, pid, pid', m) on $\mathrm{net}(\mathrm{pke}, A_{\mathrm{enc}})$, $m \neq \mathsf{Init}$, $state \neq \mathsf{error}$, and $sid \in Sids$ do:

$$m' \leftarrow \begin{cases} (\mathsf{Received}, (\mathsf{Encrypt}, k', (sid, m'')), T) & \text{if } m = (\mathsf{Received}, (\mathsf{Encrypt}, k', m''), T) \\ m & \text{otherwise ;} \end{cases}$$

$lastSid \leftarrow sid$; send (pid, pid', m') to $\mathrm{net}(\mathrm{pke}', A_{\mathrm{enc}})$

(e) *Forward from A:* If recv $(pid, pid', \mathsf{Send}, m, T)$ on $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke}')$, and $state \neq \mathsf{error}$ do:
send $(lastSid, pid, pid', \mathsf{Send}, m, T)$ to $\mathrm{net}(A_{\mathrm{enc}}, \mathrm{pke})$

If no rule above fires then produce no output.

Fig. 28. Simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{PKE}} = !S^{\mathrm{JS}}_{\mathrm{enc}} \mid !S^{\mathrm{JS}}_{\mathrm{enc}}$ for the proof of the joint state theorem for public-key encryption, the encryptor's part $S^{\mathrm{JS}}_{\mathrm{enc}}$.

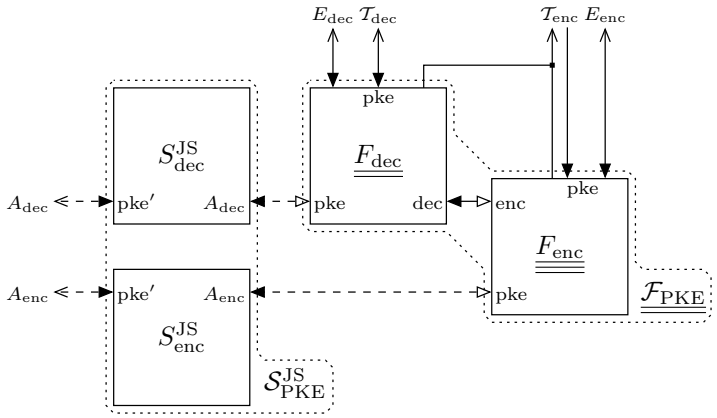

Fig. 29. Graphical representation of the simulator $\mathcal{S}^{\mathrm{JS}}_{\mathrm{PKE}} = !S^{\mathrm{JS}}_{\mathrm{dec}} \mid !S^{\mathrm{JS}}_{\mathrm{enc}}$ for $!\underline{\underline{\mathcal{F}_{\mathrm{PKE}}}} = !\underline{\underline{F_{\mathrm{dec}}}} \mid !\underline{\underline{F_{\mathrm{enc}}}}$. See Figure 18 for a legend.

Because of (5), the output in the JS world has the same distribution as the output in the ideal world. □

Note that because of the quantification over all polynomials p and q and all leakage $(L_\eta)_{\eta\in\mathbb{N}}$, the above theorem can be applied iteratively as described in Section 3. Also, the theorem implies that we obtain joint state realizations for all realizations of $\mathcal{F}_{\mathrm{PKE}}$. In particular, we obtain that the joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta))_{\eta\in\mathbb{N}})}$ of a CCA-secure public-key encryption scheme Σ realizes the multi-party multi-session version of $\mathcal{F}_{\mathrm{PKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}})$. More general, we obtain the following.

Corollary 3. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes and Σ a p-bounded CCA-secure public-key encryption scheme then for all polynomials q there is a polynomial p' such that for all length preserving leakage $(L_\eta)_{\eta\in\mathbb{N}}$ we have*

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(\mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p, q) \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}})} \leq^{SS\text{-}noaux} !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p')}}$$

where $\mathcal{P}'_{\mathrm{PKE}}$ is obtained from $\mathcal{P}_{\mathrm{PKE}}$ by renaming all tapes by replacing pke *by* pke′ *in the tape name.*

Proof. Let q be a polynomial and let p' be the polynomial from the joint state theorem (Theorem 8) which depends only on p and q. Furthermore, let $(L_\eta)_{\eta\in\mathbb{N}}$ be a length preserving leakage and define the leakage function $(L'_\eta)_{\eta\in\mathbb{N}}$ as in Theorem 8. Then $(L'_\eta)_{\eta\in\mathbb{N}}$ is length preserving, too.

Theorem 7 implies

$$\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L'_\eta))_{\eta\in\mathbb{N}}) \leq^{SS\text{-}noaux} \mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}}, p) \ .$$

By the composition theorems (Theorem 2 and 3) we obtain

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p, q) \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L'_\eta))_{\eta\in\mathbb{N}})} \leq^{SS\text{-}noaux} \mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p, q) \mid !\underline{\mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}}, p)}$$

and by the joint state theorem (Theorem 8) we obtain

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p, q) \mid !\underline{\mathcal{F}'_{\mathrm{PKE}}((L'_\eta)_{\eta\in\mathbb{N}}, p)} \leq^{SS} !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, p')}} \ .$$

Since strong simulatability without auxiliary input ($\leq^{SS\text{-}noaux}$) is transitive and is implied by strong simulatability ($\leq^{SS}$), we conclude

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(p, q) \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L'_\eta))_{\eta\in\mathbb{N}})} \leq^{SS\text{-}noaux} !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}((L_\eta)_{\eta\in\mathbb{N}}, p')}}$$

which completes the proof. □

We note that one could have tried to prove this corollary which yields a joint state realization without resorting to the ideal functionality on the left-hand side. However, this would have been much more complex, because we would have to prove a realization (with joint state) for the multi-party, multi-session version of $\mathcal{F}_{\mathrm{PKE}}$ directly. Also, results on the realization of the single-party, single-session version of $\mathcal{F}_{\mathrm{PKE}}$, such as Theorem 7, would then be completely useless. In other words, one would *not* take advantage of the main feature of the simulation-based approach: composability. Finally, using the ideal functionality yields joint state realizations for any realization of the ideal functionality. Altogether, this is why our and also all other joint state realizations in the literature are based on ideal functionalities.

6.4 Comparison with other Formulations

In the proof of the joint state theorem for public key encryption, several subtleties come up which were overlooked in other works, in particular [7, 9]. In these works, joint state theorems, similar to the one above, 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 [7] and then for [9]. 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 [7]. i) The public-key encryption functionality in [7], unlike our functionality, identifies the encryption algorithm e and the public-key. If the environment wishes to encrypt a message m, it is supposed to also present an encryption *algorithm* e' (not just a key, as in our functionality). If $e \neq e'$, i.e., e' is different from the algorithm associated with the functionality, then the ciphertext returned is $e'(m)$. Now, assume that the environment asks to encrypt some message m with e' in session sid, where, say e' coincides with e except that e' uses a different public-key. In the joint state world (i.e., in an interaction with $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \mid !\underline{\mathcal{F}'_{\mathrm{PKE}}}$), the ciphertext is computed as $e'((sid, m))$. In the ideal world (i.e., in an interaction with the simulator and $!\underline{\underline{\mathcal{F}_{\mathrm{PKE}}}}$), the ciphertext is computed as $e'(m)$. 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.

ii) In [7], the leakage is fixed to be the length of a message, i.e., instead of a message m a fixed message $\mu_{|m|}$ of length $|m|$ is encrypted, e.g. $\mu_{|m|} = 1^{|m|}$. 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 μ is encrypted. In the ideal world, the two ciphertext 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 this 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 ciphertext are negligible solves the problem. Let us call such an encryption scheme an *allowed* encryption scheme. However, this is not so if, as in [7], SIDs are not leaked in the joint state world; even if the algorithms provided by the environment/simulator are assumed to be CCA-secure.

Upon encryption of some message m_0 with the proper key k in some session sid_0 in the joint state world the ciphertext c is computed as $e(k, \mu_{|(sid_0,m_0)|})$. Depending on μ_n and how pairings are encoded, we have that

$$\mu_{|(sid_0,m_0)|} = (sid_1, m_1) \tag{6}$$

for some SID sid_1 and some plaintext m_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 μ (see below). Note that the environment can even try to choose m_0 and sid_0 in order to make (6) 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 $e(\cdot,\cdot)$, $d(\cdot)$ and key k from the environment, then it provides the algorithms $e_{sid}(\cdot,\cdot)$ and $d_{sid}(\cdot)$ and the key k to the instance of $\mathcal{F}_{\mathrm{PKE}}$ with SID sid where

$$e_{sid}(k', m)\text{: if } k = k' \text{ and not corrupted then return } e(k, \mu_{|(sid,m)|}) \text{ else return } e(k', (sid, m)) \ [4]$$

and

$$d_{sid}(c)\text{: } m \leftarrow d(c);\ \text{if } m = (sid', m') \text{ for some } m' \text{ and } sid' = sid \text{ then return } m' \text{ else return } \perp$$

for all SIDs sid. This is the only reasonable simulator because in the joint state world a ciphertext for a message m in session sid is computed as $e(k, \mu_{(sid,m)})$ and the plaintext of a ciphertext c that was not output by the functionality is computed as $m = d(c)$ and the joint state realization checks if $m = (sid', m')$ and outputs m' if $sid' = sid$ and $\perp$ otherwise. Hence, the algorithms $e'_{sid}(\cdot,\cdot)$ and $d'_{sid}(\cdot)$ provided by any successful simulator should have a distribution that is computational indistinguishable from the algorithms presented above. But in order to be more general, we do not only consider the standard simulator but a class of simulators, which in particular includes the standard simulator. This

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

class is defined as follows: We assume that the algorithm $d'_{sid_1}(\cdot)$ does not distinguish between ciphertexts generated by $e'_{sid_0}(k, \mu_{|m_0|})$ and $e(k, \mu_{|(sid_0,m_0)|})$ (for sid_0 and sid_1 as in (6)), i.e.,

$$\text{Prob}[c \leftarrow e'_{sid_0}(k, \mu_{|m_0|}) \colon d'_{sid_1}(c) = m] = \text{Prob}[c \leftarrow e(k, \mu_{|(sid_0,m_0)|}) \colon d'_{sid_1}(c) = m] \tag{7}$$

for all $m \in \{0,1\}^* \cup \{\bot\}$. One could try to construct simulators for which (7) is not satisfied but these do not seem to be the kind of simulators that one would use to prove the joint state theorem. At least such simulators have not been considered in the literature so far.

Now, we provide an environment $\mathcal{E}$ that distinguishes between the joint state and the ideal world for any simulator $\mathcal{S}$ that satisfies (7) which in particular includes the standard simulator.

In the joint state world $\mathcal{E}$ will not corrupt any parties, so, we may assume that 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.

At first $\mathcal{E}$ initializes an encryptor (with PID) pid' who encrypts messages for (the party with PID) pid in session (with SID) sid_0 and a decryptor pid in session sid_1. Furthermore, $\mathcal{E}$ provides algorithms $e(\cdot,\cdot), d(\cdot)$ and a public key k where e, d and k originate from an allowed encryption scheme, e.g. e, d, k could belong to a CCA secure encryption scheme, and chooses a random bit $b \in^R \{0,1\}$.

i) If $b = 0$, then $\mathcal{E}$ (locally) computes $c \leftarrow e(k, \mu_{|(sid_0,m_0)|})$ (note that $\mathcal{E}$ knows e and k) and sends a decryption request of c from party pid in session sid_1. Then, $\mathcal{E}$ outputs "joint state" (or 0) if the result is the plaintext m_1, and "ideal" (or 1) otherwise.

ii) If $b = 1$, then $\mathcal{E}$ requests to encrypt the plaintext m_0 by party pid' for party pid in session sid_0 and stores the ciphertext c that is returned. If $d(c) \neq \mu_{|(sid_0,m_0)|}$, $\mathcal{E}$ outputs "ideal". Finally, $\mathcal{E}$ sends a decryption request of c from party pid in session sid_1 and outputs "joint state" if the returned plaintext is $\bot$, and "ideal" otherwise.

Let us analyze the advantage of $\mathcal{E}$. When running in the joint state world $\mathcal{E}$ will always output "joint state" because of (6) and e, d, k belong to an encryption scheme and therefore $d(e(k, \mu_{|(sid_0,m_0)|})) = \mu_{|(sid_0,m_0)|}$. On the other hand, in the ideal world as the simulator does not know b and because of (7), the output of $d'_{sid_1}(c)$ does not depend on b either. But for the simulation to work it has to hold that $d'_{sid_1}(c) = m_1$ if $b = 0$ and $d'_{sid_1}(c) = \bot$ otherwise. Hence, the simulator can do no better than guessing b, i.e., $\mathcal{E}$ outputs "ideal" with probability at least $\frac{1}{2}$. Which proves that $\mathcal{E}$ is a distinguishing environment for any simulator that fulfills (7).

The above argument is robust in terms of the exact definition of $\mathcal{F}_{\text{PKE}}$ and the encoding of pairings. For example if $\mathcal{F}_{\text{PKE}}$ would ideally not encrypt constant messages but probabilistically chosen ones, e.g. $\mu \in^R \{0,1\}^{|m|}$ for plaintext m, this does not help either because $\mathcal{E}$ can decrypt the ciphertext c and check if it is of the form (sid, m) for some SID sid and message m and then proceed as above. For a short (w.r.t. η) SID sid and a short message m, the probability that $d(c) = (sid', m')$ where c is generated by $e(k, \mu)$ and $\mu \in^R \{0,1\}^*$ for some sid' and m' would be non-negligible. For other encodings of pairings or other choices of μ the argument above may fail (even though it has not been proven). In any case, it would be at least unsatisfactory if the correctness of the theorem would depend on such details.

Problems with the joint state realization in [9]. In [9], a (certified) public-key encryption functionality with local computation is proposed which is parameterized 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 8 is stated only informally and without 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 8. So, the ideal functionality has already to mimic the joint state realization. However, the ideal functionality in the joint state world should be defined differently: It should ignore SIDs, because on the left-hand side 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 [9]. 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 [7]. 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 8 and our proof of this theorem.

Alternatively, instead of parameterizing 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 CCA schemes have in general. Putting this aside (even though it is a serious problem), then the environment cannot decrypt ciphertexts a priori. However, if the algorithms are parameterized with the algorithms the standard simulator provides, the environment can dispense with the decryption because it knows what the ciphertexts will decrypt to (namely $\mu_{|m|}$: it knows the plaintext if fixed leakage is used as in [9]). For other parameterizations satisfying (7) or probabilistic leakage one has to argue that the environment does not gain information about plaintexts (in particular the SIDs in them). Hence, one would have to assume that the encryption scheme used is secure in some sense. But then there does not seem to be a point in using the ideal functionality at all in the joint state theorem. Instead one can just as well try to prove the joint state theorem directly without using an ideal functionality on the left-hand side. Consequently, realizations shown for the ideal functionality itself would be of no use.

Remarks on other Functionalities in the Literature. As mentioned in the introduction, other formulations of public-key encryption functionalities, e.g., those in [5, 20], 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 [28, 2].

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 CCA-secure scheme, but requires a specific stronger construction.

In [28], formulations of public-key encryption functionalities with local computation are proposed which are parameterized by specific encryption and decryption algorithms. With the drawbacks (concerning joint state) mentioned for [9].

We note that in [2, 28] joint state realizations of their 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 5.4)

Furthermore, we note that, analogously to the case of digital signatures (see Section 5.4) and unlike other formulations, our formulation models the realistic situation that encryptions can be performed even before the decryption box has been invoked for the first time by the decryptor. Corruptions are specified rigorously, and encryption and decryption boxes can be corrupted independently. Our functionality $\mathcal{F}_{\mathrm{PKE}}$ can be invoked an unbounded number of times, with arbitrarily long messages. We note that $\mathcal{F}_{\mathrm{PKE}}$ specifies the public-key encryption functionality for a *single* party (more precisely, *one* decryptor, with an unbounded number of encryptors). If $\mathcal{P}_{\mathrm{PKE}}$ realizes $\mathcal{F}_{\mathrm{PKE}}$, i.e., $\mathcal{P}_{\mathrm{PKE}} \leq^{SS} \mathcal{F}_{\mathrm{PKE}}$, then our composition theorem immediately implies that the multi-party version $!\underline{\mathcal{P}_{\mathrm{PKE}}}$ of $\mathcal{P}_{\mathrm{PKE}}$ realizes the multi-party version $!\underline{\mathcal{F}_{\mathrm{PKE}}}$ of $\mathcal{F}_{\mathrm{PKE}}$, i.e., $!\underline{\mathcal{P}_{\mathrm{PKE}}} \leq^{SS} !\underline{\mathcal{F}_{\mathrm{PKE}}}$. Applying the composition theorem again yields that $!\underline{\underline{\mathcal{P}_{\mathrm{PKE}}}} \leq^{SS} !\underline{\underline{\mathcal{F}_{\mathrm{PKE}}}}$, i.e., the multi-party, multi-session version of $\mathcal{P}_{\mathrm{PKE}}$ realizes the multi-party, multi-session version of $\mathcal{F}_{\mathrm{PKE}}$.

7 Replayable Public-Key Encryption

In this section, we propose a public-key encryption functionality with local computation which is designed to capture IND-RCCA-secure, i.e., replayable IND-RCCA-secure, encryption schemes. Recall that

Functionality $F_{\mathrm{rdec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$

I/O-tapes: in: $\mathrm{io}(T, \mathrm{pke})$ for each $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathrm{io}(E_{\mathrm{dec}}, \mathrm{pke})$, $\mathrm{io}(\mathrm{enc}, \mathrm{dec})$ (enriching)
out: $\mathrm{io}(\mathrm{pke}, T)$ for each $T \in \mathcal{T}_{\mathrm{dec}} \cup \mathcal{T}_{\mathrm{enc}}$, $\mathrm{io}(\mathrm{pke}, E_{\mathrm{dec}})$, $\mathrm{io}(\mathrm{dec}, \mathrm{enc})$
net-tapes: in: $\mathrm{net}(A_{\mathrm{dec}}, \mathrm{pke})$ (consuming) out: $\mathrm{net}(\mathrm{pke}, A_{\mathrm{dec}})$

Initialization: $e, d, k \leftarrow \bot$; $H \leftarrow \emptyset$; *state* $\leftarrow$ init ; *nokey* $\leftarrow$ true ; *corrupted* $\leftarrow$ false

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

(a) *Initialization:* If recv (Init) from $T \in \mathcal{T}_{\mathrm{dec}}$, and *state* = init do: *state* $\leftarrow$ (wait, T) ; send (Init) to A_{dec}

(b) *Initialization Response:* If recv (Inited) from A_{dec}, not *nokey*, and *state* = (wait, T) do:
state $\leftarrow$ ok ; send (PublicKey, k) to T

(c) *Wake up:* If recv (pid, WakeUpFromEnc) on $\mathrm{io}(\mathrm{enc}, \mathrm{dec})$ do: send (pid, Ack) to $\mathrm{io}(\mathrm{dec}, \mathrm{enc})$

(d) *Key Generation:* If recv (AlgorithmsAndKey, e', d', k') from A_{dec}, *nokey*, and $|e'|, |d'|, |k'| \leq p(\eta)$ do:
$(e, d, k) \leftarrow (e', d', k')$; *nokey* $\leftarrow$ false ; send (Ack) to A_{dec}

(e) *Decryption:* If recv (Decrypt, c) from $T \in \mathcal{T}_{\mathrm{dec}}$, *state* = ok, and not *corrupted* do:
$\overline{m} \leftarrow \text{sim-det}_{p(|c|+\eta)}\, d(c)$;

$$m \leftarrow \begin{cases} \bot & \text{if } \exists m', m'' \colon m' \neq m'', (m', \overline{m}) \in H, (m'', \overline{m}) \in H \\ m' & \text{if } \exists! m' \colon (m', \overline{m}) \in H \\ \overline{m} & \text{otherwise ;} \end{cases}$$

send (Plaintext, m) to T

(f) *Encryption:* If recv (pid, Encrypt, k', m, T) from enc where $T \in \mathcal{T}_{\mathrm{enc}}$, and not *nokey* do:
if $m \notin \mathrm{dom}(L_\eta)$ then $c \leftarrow \bot$
else if $k \neq k'$ or *corrupted* then $c \leftarrow \mathrm{sim}_{p(|m|+\eta)}\, e(k', m)$
else $\overline{m} \leftarrow L_\eta(m)$; $c \leftarrow \mathrm{sim}_{p(|\overline{m}|+\eta)}\, e(k, \overline{m})$; if $c \neq \bot$ then $H \leftarrow H \cup \{(m, \overline{m})\}$ end ;
end ;
send (pid, Ciphertext, c) to T

(g) *Corruption:* Corr(*corrupted*, true, *state* $\neq$ init, $H, A_{\mathrm{dec}}, \mathcal{T}_{\mathrm{dec}}, E_{\mathrm{dec}}$) (See Figure 1 for definition of Corr)

If no rule above fires then produce no output.

Fig. 30. Ideal replayable public-key encryption func. $\mathcal{F}_{\mathrm{RPKE}} = F_{\mathrm{rdec}} \mid !\underline{F_{\mathrm{enc}}}$, the decryptor's part F_{rdec}.

IND-RCCA-security is a relaxed form of CCA-security where modifications of the ciphertext that yield the same plaintext are permitted (see [11]). As explained in [11], RCCA-security suffices in many applications where CCA-security is used. The equivalence of IND-RCCA-secure encryption schemes and our functionality for replayable public-key encryption is established in Section 7.2, with a joint state realization presented in Section 7.3.

7.1 Ideal Replayable Public-Key Encryption Functionality

We now present our ideal functionality $\mathcal{F}_{\mathrm{RPKE}}$ with local computation for replayable public-key encryption.

In many technical matters the formulation of $\mathcal{F}_{\mathrm{RPKE}}$ is similar to $\mathcal{F}_{\mathrm{PKE}}$. We also refer to the conventions given in Section 4. Recall the definition of leakage (Definition 3).

The functionality $\mathcal{F}_{\mathrm{RPKE}}$ with leakage $(L_\eta)_{\eta\in\mathbb{N}}$ is defined as follows

$$\mathcal{F}_{\mathrm{RPKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p) = F_{\mathrm{rdec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p) \mid !\underline{F_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}})}$$

where the IITMs F_{rdec} and $\underline{F_{\mathrm{enc}}}$ represent the decryptor's and encryptor's part, respectively. The IITM $F_{\mathrm{enc}} = F_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}})$, is defined as in Section 6.1 in Figure 17. The IITM $F_{\mathrm{rdec}} = F_{\mathrm{rdec}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$, as defined in Figure 30, is parameterized by the leakage $(L_\eta)_{\eta\in\mathbb{N}}$, two disjoint sets of names of tapes $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ which are used by the decryptor or the encryptors (resp.) to connect to F_{rdec} and by a polynomial p.

The major difference between $\mathcal{F}_{\mathrm{RPKE}}$ and $\mathcal{F}_{\mathrm{PKE}}$ is that upon encryption, the pair $(m, \overline{m})$ is stored instead of (m, c) and that upon decryption it is not looked for the ciphertext c but for the decryption $d(c)$ of the ciphertext. Hence, it might be possible to produce a ciphertext $c' \neq c$ such that the decryption m of c' is the same as the one of c without knowing m. This models replayable encryption. Another difference is that the decryption test can be omitted.

7.2 Realization by an RCCA-Secure Encryption Scheme

In this section, it is shown that the ideal replayable public-key encryption functionality $\mathcal{F}_{\text{RPKE}}$ that leaks the length of a message is realized by a public-key encryption scheme if and only if the encryption scheme is secure with respect to adaptive replayable chosen-ciphertext attacks (IND-RCCA-secure) as defined by [11]. We first restate the definition of RCCA.

Let $A = (A_1, A_2)$ be an adversary, as defined in Section 6.2, and $b \in \{0,1\}$. The corresponding *RCCA-experiment* is a probabilistic algorithm $T_{A,\Sigma}$ that runs on input $b \in \{0,1\}$ and $\eta \in \mathbb{N}$.

$$T_{A,\Sigma}(b,\eta)\colon (k_d,k_e) \leftarrow \text{gen}(1^\eta);\ (x_0,x_1,s) \leftarrow A_1^{\text{dec}(k_d,\cdot)}(k_e,1^\eta);$$
$$c^* \leftarrow \text{enc}(k_e,x_b);\ b' \leftarrow A_2^{\overline{\text{dec}}(x_0,x_1,k_d,\cdot)}(x_0,x_1,s,c^*,1^\eta);\ \text{return } b';$$

where $\overline{\text{dec}}(x_0,x_1,k_d,\cdot)$ is defined as follows:

$$\overline{\text{dec}}(x_0,x_1,k_d,c)\colon x \leftarrow \text{dec}(k_d,c); \text{if } x \in \{x_0,x_1\} \text{ then return } \mathsf{test} \text{ else return } x;$$

where test is a special message that is never returned by dec.

The *advantage* of an adversary $A = (A_1, A_2)$ regarding the public-key encryption scheme Σ is defined by

$$\text{Adv}(A,\Sigma,\eta) = |\text{Prob}[T_{A,\Sigma}(1,\eta)=1] - \text{Prob}[T_{A,\Sigma}(0,\eta)=1]| \ .$$

Definition 6 ([11]). *A public-key encryption scheme Σ is called* secure against replayable adaptive chosen-ciphertext attacks (IND-RCCA-secure) *if for all adversaries A the advantage* $\text{Adv}(A,\Sigma,\eta)$ *is negligible as a function in η.*

The following theorem shows that an RCCA-secure encryption scheme realizes $\mathcal{F}_{\text{RPKE}}$ in the context of environments without auxiliary input. One could alternatively define RCCA-security by allowing auxiliary input to the adversary. Then, an RCCA-secure encryption scheme would realizes $\mathcal{F}_{\text{RPKE}}$ in the context of environments with auxiliary input. Note that length preserving leakage with high entropy (recall Definition 4) implies that the domain of plaintexts contains only long messages.

Theorem 9. *Let $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ be some disjoint sets of names of tapes, Σ an IND-RCCA-secure p-bounded public-key encryption scheme. Then,*

$$\mathcal{P}_{\text{PKE}}(\Sigma,(\text{dom}(L_\eta))_{\eta\in\mathbb{N}},\mathcal{T}_{\text{dec}},\mathcal{T}_{\text{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},\mathcal{T}_{\text{dec}},\mathcal{T}_{\text{enc}},p)$$

for every length preserving leakage $(L_\eta)_{\eta\in\mathbb{N}}$ that has high entropy.

Proof. We prove the theorem by contradiction.

Assuming $\mathcal{P}_{\text{PKE}}(\Sigma,(\text{dom}(L_\eta))_{\eta\in\mathbb{N}}) \not\leq^{SS\text{-}noaux} \mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},p)$ we show that Σ is not IND-RCCA-secure by constructing a successful adversary. Let $\mathcal{S} = \mathcal{S}_{\text{PKE}}$ be the simulator from Section 6.2 and $\mathcal{E} \in \text{Con}_{\mathbf{E}}(\mathcal{P}_{\text{PKE}}(\Sigma,(\text{dom}(L_\eta))_{\eta\in\mathbb{N}}))$ such that $\mathcal{E} \,|\, \mathcal{P}_{\text{PKE}}(\Sigma,(\text{dom}(L_\eta))_{\eta\in\mathbb{N}}) \not\equiv^{noaux} \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},p)$.

Let B denote the event that in an run of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},p)(1^\eta,\varepsilon)$ the environment $\mathcal{E}$ asks for encryption of a plaintext m after m was chosen by L_η in Step (f) of F_{rdec}. We show that if Σ is IND-RCCA-secure then B occurs only with negligible probability. Intuitively, this is clear because L has high entropy, so, guessing is only successful with negligible probability and if $\mathcal{E}$ could learn m from the encryption of it then Σ is not IND-RCCA-secure. More formally, assume that Σ is IND-RCCA-secure and that B occurs with non-negligible probability. We define the adversary $A = (A_1, A_2)$ as follows. Let p be a polynomial such that the number of messages that $\mathcal{E}$ encrypts is bounded by $p(\eta)$.

$A_1(k_e,1^\eta)$ chooses $h \leftarrow^{\text{R}} \{1,\ldots,p(\eta)\}$ and simulates $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},p)(1^\eta,\varepsilon)$ except that the public key that $\mathcal{S}$ hands to $\mathcal{F}_{\text{RPKE}}$ is replaced by k_e and upon decryption not $d(c)$ is computed but the decryption oracle $\text{dec}(k_d,c)$ is called. If $\mathcal{E}$ asks $\mathcal{F}_{\text{RPKE}}$ to encrypt the h-th messages m_h with the proper key k_e then A_1 computes $x_0 \leftarrow L_\eta(m_h)$ and $x_1 \leftarrow L_\eta(m_h)$ and outputs (x_0,x_1,s) where s contains all information about the simulation of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{RPKE}}$, the public key k_e and the message m_h. Note that the length of s is polynomial in the security parameter.

$A_2(x_0,x_1,(k_e,m_h),c^*,1^\eta)$ parses s and continues the simulation of $\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\text{RPKE}}$ but $\mathcal{F}_{\text{RPKE}}$ does not store m_h but returns c^* as the encryption of m_h to $\mathcal{E}$. Upon decryption of c not $d(c)$ is computed but $\overline{\text{dec}}(x_0,x_1,k_d,c)$. If the decryption oracle returns test then $\mathcal{F}_{\text{RPKE}}$ returns m_h to $\mathcal{E}$ as the plaintext. If

at some point $\mathcal{E}$ asks to encrypt x_0 (resp., x_1) then A_2 outputs 0 (resp., 1). Otherwise, if the run stops and $\mathcal{E}$ has never output x_0 or x_1 then A_2 outputs $b \leftarrow^{\mathrm{R}} \{0,1\}$.

Next, we analyze the advantage of A. Note that the view of $\mathcal{E}$ in the simulation of A is identical to its view in an interaction with $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{RPKE}}$. Hence, if $b = 0$ then the run is independent of the value of x_1. Hence, the probability that $\mathcal{E}$ asks for encryption of x_1 is at most the probability of guessing x_1, i.e., $\mathrm{Prob}[T_{A,\Sigma}(0,\eta) = 1] \leq \frac{1}{2} + p(\eta) \cdot f(\eta)$ where f is a negligible function. On the other hand, if $b = 1$ then the probability that $\mathcal{E}$ asks for encryption of x_1 is $\frac{1}{p(\eta)} \cdot \mathrm{Prob}[B]$. Since $\mathcal{E}$ might ask for encryption of x_0 with small probability, we have that $\mathrm{Prob}[T_{A,\Sigma}(1,\eta) = 1] \geq \frac{1}{2} + \frac{1}{p(\eta)} \cdot \mathrm{Prob}[B] - p(\eta) \cdot f(\eta)$. We conclude

$$\mathrm{Adv}(A, \Sigma, \eta) \geq \frac{1}{p(\eta)} \cdot \mathrm{Prob}[B] - 2 \cdot p(\eta) \cdot f(\eta)$$

which is non-negligible if $\mathrm{Prob}[B]$ is non-negligible because p is a polynomial and f is negligible. This proves that $\mathrm{Prob}[B]$ is negligible if Σ is IND-RCCA-secure.

Now, we give a successful adversary $A = (A_1, A_2)$ on Σ. A is defined as the adversary in the proof of Theorem 7 except that if the decryption oracle $\overline{\mathrm{dec}}(x_0, x_1, k_d, c)$ returns test then A_2 continues as if it returned m_h. (The motivation is that c either decrypts to m_h or to $L_\eta(m_h)$, but $L_\eta(m_h)$ was requested to be encrypted only if event B occurs. However, the probability for B to happen is negligible.)

The proof proceeds via a hybrid argument as the proof of Theorem 7. We conclude that Σ is not IND-RCCA-secure which proves the theorem. □

Lemma 2. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes, $\Sigma = (\mathrm{gen}, \mathrm{dec}, \mathrm{enc})$ a p-bounded public-key encryption scheme such that $\mathrm{Prob}[\mathrm{enc}(k,m) = \bot] = \mathrm{Prob}[L_\eta^{|\cdot|}(m) = \bot]$ for all k generated by $\mathrm{gen}(1^\eta)$ and all $m \in \{0,1\}^*$ and*

$$\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p) \ .$$

Then, Σ is IND-RCCA-secure.

Proof. The proof is similar to the proof of Lemma 1. Assuming Σ is not IND-RCCA-secure, we use a successful adversary $A = (A_1, A_2)$ to construct an environment $\mathcal{E}$ that distinguishes $\mathcal{P}_{\mathrm{PKE}}$ from $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{RPKE}}$ for each simulator $\mathcal{S}$. The environment $\mathcal{E}$ is defined as in the proof of Lemma 1 except that if A_2 asks its decryption oracle to decrypt a message c then $\mathcal{E}$ asks $\mathcal{F}_{\mathrm{RPKE}}$ to decrypt c. If $\mathcal{F}_{\mathrm{RPKE}}$ returns one of the messages x_0 or x_1 (which A claims to distinguish) as a decryption then $\mathcal{E}$ continues the simulation of A_2 as if test is returned.

One can show that if $\mathcal{E}$ interacts with the real world, i.e. $\mathcal{P}_{\mathrm{PKE}}$, then the simulation of A is exactly like in the experiment $T_{A,\Sigma}$ and thus $\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}}) \rightsquigarrow 1]$ is non-negligible less than one half.

On the other hand, one can prove that if $\mathcal{E}$ interacts with the ideal world, i.e. with $\mathcal{S} \,|\, \mathcal{F}_{\mathrm{RPKE}}$, then b' is independent of b and thus $\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}})(1^\eta, \varepsilon) \rightsquigarrow 1]$ is exactly one half.

We conclude that

$$|\mathrm{Prob}[\mathcal{E} \,|\, \mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}})(1^\eta, \varepsilon) \rightsquigarrow 1] - \mathrm{Prob}[\mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}})(1^\eta, \varepsilon) \rightsquigarrow 1]|$$

is non-negligible for each simulator $\mathcal{S}$ and thus $\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}}) \not\leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}})$. □

With $(L_\eta)_{\eta\in\mathbb{N}} = (L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}$ in Theorem 9, we obtain:

Corollary 4. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes and let Σ be a p-bounded public-key encryption scheme where $\mathrm{Prob}[\mathrm{enc}(k,m) = \bot] = \mathrm{Prob}[L_\eta^{|\cdot|}(m) = \bot]$ for all k generated by $\mathrm{gen}(1^\eta)$ and all $m \in \{0,1\}^*$. Then, Σ is IND-RCCA-secure if and only if*

$$\mathcal{P}_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}) \leq^{SS\text{-}noaux} \mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p) \ .$$

7.3 Joint State for Replayable Public-Key Encryption

In this section, we show that the protocol system $\mathcal{P}^{\text{JS}}_{\text{PKE}}$ (see Section 6.3) is not only a joint state realization of $\mathcal{F}_{\text{PKE}}$ but also of $\mathcal{F}_{\text{RPKE}}$.

Theorem 10. *For all polynomials p and q and disjoint sets of names of tapes $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ exists a polynomial p' such that for every leakage $(L_\eta)_{\eta\in\mathbb{N}}$ we have*

$$\mathcal{P}^{\text{JS}}_{\text{PKE}}(\mathcal{T}_{\text{dec}},\mathcal{T}_{\text{enc}},p,q) \mid \underline{!\mathcal{F}'_{\text{RPKE}}((L'_\eta)_{\eta\in\mathbb{N}},\mathcal{T}_{\text{dec}},\mathcal{T}_{\text{enc}},p)} \leq^{SS} \underline{\underline{!\mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},\mathcal{T}_{\text{dec}},\mathcal{T}_{\text{enc}},p')}}$$

where $(L'_\eta)_{\eta\in\mathbb{N}}$ is defined by

$$L'_\eta(x)\colon \textit{if } \exists sid\in\{0,1\}^*, x'\in\text{dom}(L_\eta)\colon x \textit{ is of shape } (sid,x') \textit{ then return } (sid,L_\eta(x')) \textit{ else return } \bot$$

and $\mathcal{F}'_{\text{RPKE}}$ is obtained from $\mathcal{F}_{\text{RPKE}}$ by renaming all tapes by replacing pke *by* pke′ *in the tape name.*

Proof. The proof is similar to the proof of the joint state theorem of $\mathcal{F}_{\text{PKE}}$, in particular, we use the same simulator $\mathcal{S}=\mathcal{S}^{\text{JS}}_{\text{PKE}}$ (see Figure 27 and 28) and polynomial p' to prove

$$\mathcal{E}\mid\mathcal{P}^{\text{JS}}_{\text{PKE}}(p,q)\mid \underline{!\mathcal{F}'_{\text{RPKE}}((L'_\eta)_{\eta\in\mathbb{N}},p)} \equiv \mathcal{E}\mid\mathcal{S}\mid \underline{\underline{!\mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}},p')}}$$

for all environments $\mathcal{E}\in\text{Con}_{\mathbf{E}}(\mathcal{P}^{\text{JS}}_{\text{PKE}}\mid \underline{!\mathcal{F}'_{\text{RPKE}}((L'_\eta)_{\eta\in\mathbb{N}})})$.

At first note that upon initialization, key generation and corruption the behavior of $\mathcal{P}^{\text{JS}}_{\text{PKE}}\mid \underline{!\mathcal{F}'_{\text{RPKE}}}$ does not differ from the behavior of $\underline{\underline{!\mathcal{F}_{\text{RPKE}}}}$.

Let H_{pid} refer to H in the copy $\mathcal{F}'_{\text{RPKE}}((L'_\eta)_{\eta\in\mathbb{N}})[pid]$ in the joint state world (JS world) while $H_{sid,pid}$ refers to H in the copy $\mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}})[sid,pid]$ in the ideal world. Because of the definition of the leakage $(L'_\eta)_{\eta\in\mathbb{N}}$ we have that if $((sid,m),(sid',\overline{m}))\in H_{pid}$ then $sid=sid'$. We can prove

$$\text{Prob}[((sid,m),(sid,\overline{m}))\in H_{pid}] = \text{Prob}[(m,\overline{m})\in H_{sid,pid}] \tag{8}$$

for all SIDs sid and messages $m,\overline{m}\in\{0,1\}^*$.

The output of $\mathcal{P}^{\text{JS}}_{\text{PKE}}\mid \underline{!\mathcal{F}'_{\text{RPKE}}((L'_\eta)_{\eta\in\mathbb{N}})}$ upon an encryption request has the same distribution as the output of $\underline{\underline{!\mathcal{F}_{\text{RPKE}}((L_\eta)_{\eta\in\mathbb{N}})}}$ because of the definition of e_{sid} and the choice of p'.

Now, consider decryption of a ciphertext c in session sid by a decryptor with PID pid.

First, we show that $\bot$ is returned in the JS world if and only if it is returned in the ideal world.

Note that if a collision occurs in the JS world, i.e. there exist $(sid,\overline{m})$, (sid',m') and (sid'',m'') such that $((sid',m'),(sid,\overline{m}))\in H_{pid}$ and $((sid'',m''),(sid,\overline{m}))\in H_{pid}$ then $sid=sid'=sid''$. Hence, collisions in the JS world do not occur across different sessions. By the definition of $(L'_\eta)_{\eta\in\mathbb{N}}$, for each ciphertext the probability that a collision occurs in the JS world in session sid is equal to the probability that a collision occurs in the ideal world in $H_{sid,pid}$.

If no collisions occur, the error symbol $\bot$ is returned in the JS world if and only if one of the following four cases apply:

(a) $d(c)=\bot$.
(b) $d(c)=x$ and there is no entry in H_{pid} with second component x but x is not of shape (sid,m) for some message m.
(c) $d(c)=x$ and $\exists!x'\colon (x',x)\in H_{pid}$ and x' is not of shape (sid,m) for some message m.

If (a) then $d_{sid}(c)=\bot$, so, $\bot$ is returned in the ideal world as well. If (b) then $d_{sid}(c)=\bot$, so, $\bot$ is returned in the ideal world. If (c) then $x=(sid',m)$ and $x'=(sid',m')$ for some sid', m and m' but $sid'\neq sid$. Thus, $d_{sid}(c)=\bot$ and $\bot$ is returned in the ideal world.

Vice versa, if no collisions occur, $\bot$ is returned in the ideal world if and only if $d_{sid}(c)=\bot$. Then $x=d(c)$ is not of shape (sid,m) for some message m. If $((sid',m'),x)\in H_{pid}$ for some sid' and m' then $sid'\neq sid$. Thus, $\bot$ is returned in the JS world.

We conclude that the probability that $\bot$ is returned in the JS world is equal to the probability that $\bot$ is returned in the ideal world.

Now, assume that decryption of the ciphertext c in session sid returned a plaintext $m \neq \perp$ in the JS world.

Then either i) $d(c) = (sid, m)$ and (sid, m) does not occur in H_{pid} in the second component, or ii) $d(c) = (sid, \overline{m})$ and $((sid, m), (sid, \overline{m})) \in H_{pid}$ (and no collision occurs). If i) then $d_{sid}(c) = m$ and if ii) then $d_{sid}(c) = \overline{m}$.

On the other hand, if the decryption of the ciphertext c in session sid returns a plaintext $m \neq \perp$ in the ideal world then either i) $d_{sid}(c) = m$ and m does not occur in $H_{sid,pid}$ in the second component, or ii) $d_{sid}(c) = \overline{m}$ and $(m, \overline{m}) \in H_{pid}$ (and no collision occurs). If i) then $d(c) = (sid, m)$ and if ii) then $d(c) = (sid, \overline{m})$.

Because of (8) the output upon decryption in the JS world does not differ from the output in the ideal world. □

As in Section 6.3, we can note that because of the quantification over all polynomials p and q and all leakage $(L_\eta)_{\eta\in\mathbb{N}}$, the above theorem can be applied iteratively as described in Section 3. Also, the theorem implies that we obtain joint state realizations for all realizations of $\mathcal{F}_{\mathrm{PKE}}$. In particular, we obtain that the joint state realization $\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}} \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta^{|\cdot|}))_{\eta\in\mathbb{N}})}$ of an IND-RCCA-secure public-key encryption scheme Σ realizes the multi-party multi-session version of $\mathcal{F}_{\mathrm{RPKE}}((L_\eta^{|\cdot|})_{\eta\in\mathbb{N}})$. More general, we obtain the following.

Corollary 5. *Let $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ be disjoint sets of names of tapes and Σ a p-bounded IND-RCCA-secure public-key encryption scheme. Then for all polynomials q there is a polynomial p' such that for every length preserving leakage $(L_\eta)_{\eta\in\mathbb{N}}$ that has high entropy we have*

$$\mathcal{P}^{\mathrm{JS}}_{\mathrm{PKE}}(\mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p, q) \mid !\underline{\mathcal{P}'_{\mathrm{PKE}}(\Sigma, (\mathrm{dom}(L_\eta))_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}})} \leq^{SS\text{-}noaux} !\underline{\underline{\mathcal{F}_{\mathrm{RPKE}}((L_\eta)_{\eta\in\mathbb{N}}, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p')}}$$

where $\mathcal{P}'_{\mathrm{PKE}}$ is obtained from $\mathcal{P}_{\mathrm{PKE}}$ by renaming all tapes by replacing pke *by* pke′ *in the tape name.*

As for non-replayable public key encryption (Section 6), we note that one could have tried to prove this corollary which yields a joint state realization without resorting to the ideal functionality on the left-hand side. However, this has several disadvantages as discussed in Section 6.3.

7.4 Comparison with other Formulations

In [11], Canetti et al. define and motivate RCCA-secure encryption schemes and propose a public-key functionality with *non-local* computation that captures RCCA-security. In [7], Canetti sketches in a few lines how his public-key encryption functionality with local computation should be modified to obtain a functionality that mimics 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 this functionality with RCCA-security is not provided. Also, neither in [11] nor in [7] the issue of joint state is touched in the context of 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 in the context of RCCA-security.

The general remarks in Sections 5.4 and 6.4 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.

References

[1] M. Backes and D. Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. In K. Zhang and Y. Zheng, editors, *Proceedings of the 7th International Conference on Information Security (ISC 2004)*, 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. Backes, B. Pfitzmann, and M. Waidner. Secure Asynchronous Reactive Systems. Technical Report 2004/082, Cryptology ePrint Archive, 2004. Available at http://eprint.iacr.org/2004/082.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-Party and Multi-party Secure Computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 494–503. ACM, 2002.

[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 D. Boneh, editor, *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] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto 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.

[16] V. Cortier, R. Küsters, and B. Warinschi. A Cryptographic Model for Branching Time Security Properties – the Case of Contract Signing Protocol. In J. Biskup and J. Lopez, editors, *Proceedings of the 12th European Symposium on Research in Computer Security (ESORICS 2007)*, volume 4734 of *Lecture Notes in Computer Science*, pages 422–437. Springer, 2007. Full version available at http://eprint.iacr.org/2007/251.

[17] A. Datta, R. Küsters, J.C. 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-Verlag, 2005. Full version to appear in the Journal of Cryptology.

[18] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

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

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

[21] 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.

[22] 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.

[23] 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.

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

[25] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 514–523. ACM, 2002.

[26] 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.

[27] B. Pfitzmann, M. Schunter, and M. Waidner. Reactively Simulatable Certified Mail. Technical Report 2006/041, Cryptology ePrint Archive, 2006. Available at http://eprint.iacr.org/2006/041.

[28] 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 184–200. IEEE Computer Society, 2001.

[29] C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Advances in Cryptology, 11th Annual International Cryptology Conference (CRYPTO 1991)*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1991.