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

Ralf Küsters 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, 4, 6, 7, 16, 18, 19, 20, 25]) 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, ii) a joint state theorem which is more general than other formulations and corrects flaws in these formulations, and iii) based on this theorem, joint state realizations and theorems for (replayable) public-key encryption and digital signatures. Unfortunately, all other joint state theorems claimed in the literature for such functionalities with local computation can be shown to be flawed. An overall distinguishing feature of our work is the rigorous treatment, the simplicity of our definitions, and the generality of our results, which is due to the expressivity and simplicity of the model for simulation-based security that we use, the IITM model [20]. 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, 5, 1, 11, 17], 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 [6], within the IITM model we do not need to explicitly define a specific joint state operator. Also, our joint state theorem, unlike the one in the UC model, immediately follows from the composition theorem in the IITM model, no extra proof is needed. In addition to the seamless treatment of the joint state theorem within the IITM model, which exemplifies the simplicity and expressivity of the IITM model, our theorem is even more general than the one in [14, 6] (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 [4]. 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 [6], 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, the joint state theorems claimed for the public-key encryption and digital signature functionalities in [6] are flawed. The same is true for the work by Canetti 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, 23, 12, 24].

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 [25]. 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 [25, 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 [5, 1, 4, 17]; 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 briefly recall the IITM model. The general joint state theorem is presented in Section 3, along with a discussion of the joint state theorem of Canetti and Rabin [14]. In Section 4, we present our formulations of the functionalities for (replayable) public-key encryption and digital signatures. Joint state realizations of these functionalities are presented in Section 5, including a comparison with other formulations in the literature. Full details and more explanations can be found in our technical report [22].

2. The IITM Model

In this section, we briefly recall the IITM model for simulation-based security (see [20, 21] for details). Based on a relatively simple, but expressive general computational model in which IITMs and systems of IITMs are defined, simulation-based security notions are formalized, and a general composition theorem is stated. The IITM model has several features which are important prerequisites for the generality and simplicity of our definitions and results. Among others, these features include i) a generic addressing mechanism which does not fix details of how copies of machines are addressed, e.g., in terms of a specific use of session and party IDs, ii) inexhaustible ITMs (IITMs) which can be activated an unbounded number of times without exhausting their resources and whose runtime may depend on the length of their input, iii) a compact and flexible notation for systems of IITMs, and iv) a general composition theorem.

2.1. The General Computational Model

We first define IITMs and then systems of IITMs. We note that this general computational model is also useful in contexts other than simulation-based security [15].

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 $\mathsf{consuming}$ or $\mathsf{enriching}$ (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 an (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 the IITM (see below).

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 cannot 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, $\mathsf{CheckAddress}$ (deterministic computation) and $\mathsf{Compute}$ (probabilistic computation). The $\mathsf{CheckAddress}$ mode will be used to address different copies of IITMs in a system of IITMs (see below). This is a very generic and flexible addressing mechanism: Details of how an IITM is addressed are not fixed up-front, but are left to the specification of the IITM itself.

Systems of IITMs. A *system* $\mathcal{S}$ of IITMs is of the form $M_1 \,\|\, \dots \,\|\, M_k \,\|\, !M'_1 \,\|\, \dots \,\|\, !M'_{k'}$ (we use the normal form representation here) where the M_i and M'_j are IITMs such that different IITMs have disjoint sets of names of input tapes, and hence, the output tape of one IITM connects to at most one input tape of another IITM. We say that the machines $M'_1, \dots, M'_{k'}$ are in the *scope of a bang*. While machines outside the scope of a bang occur at most once in a run of a system, for those in the scope of a bang arbitrarily many copies (with fresh randomness) can be generated.

More precisely, in a run of $\mathcal{S}$ at any time only one IITM, say a copy of some M in $\mathcal{S}$, is active and all other IITMs wait for new input. 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 an 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), would accept m. If yes, this copy gets to process m. Otherwise, the master IITM in $\mathcal{S}$ is activated. 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 consider only so-called *well-formed* systems [20], which satisfy a simple condition, namely, acyclicity w.r.t enriching tapes, that guarantees polynomial runtime and suffices for applications since it allows to always provide sufficient resources to IITMs via enriching tapes (see Section 2.2).

We write $\mathrm{Prob}[\mathcal{S}(1^\eta, a) \rightsquigarrow 1]$ to denote the probability that the overall output of a run of $\mathcal{S}$ 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* ($\mathcal{P} \equiv \mathcal{Q}$) 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 $f(1^\eta, a) \leq \frac{1}{p(\eta)}$.

Given an IITM M, we will often use its *identifier version* $\underline{M}$ to be able to address multiple copies of M. The identifier version $\underline{M}$ of M is an IITM which simulates M 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 $\underline{M}$ uses 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 $\mathcal{S}$ as above, its *identifier version* $\underline{\mathcal{S}}$ is $\underline{M_1} \parallel \ldots \parallel \underline{M_k} \parallel !\underline{M'_1} \parallel \ldots \parallel !\underline{M'_{k'}}$. 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

To define security notions for simulation-based security based on the general computational model from above, we need some terminology.

For a system $\mathcal{S}$, the input/output tapes of IITMs in $\mathcal{S}$ that do not have a matching output/input tape are called *external*. We group these tapes into *I/O* and *network tapes*. We consider three different types of well-formed systems, modeling real/ideal protocols/functionalities, adversaries/simulators, and environments, respectively: *Protocol systems* are well-formed systems whose IITMs have consuming network and enriching I/O tapes. This is w.l.o.g., as 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 (see below). *Adversarial systems* are well-formed systems whose IITMs may have enriching tapes. *Environmental systems* are well-formed systems whose IITMs have only consuming tapes. They may use the special tapes start and decision, i.e., an environmental system may contain a master IITM and determines the overall output of a system run. It is easy to see that combinations of these systems are again well-formed.

Two systems are called *compatible* if they have the same set of I/O and network tapes, they are called *I/O-compatible* if they have the same set of I/O tapes and disjoint sets of network tapes. A system is called *connectible* for another system if each common external tape has the same type in both (network or I/O) and complementary directions (input or output). A system $\mathcal{S}$ is *adversarially connectible* for $\mathcal{Q}$ if $\mathcal{S}$ is connectible for $\mathcal{Q}$ and $\mathcal{S}$ connects only to network tapes of $\mathcal{Q}$. A system $\mathcal{S}$ is *environmentally connectible* for $\mathcal{Q}$ if $\mathcal{S}$ is connectible for $\mathcal{Q}$ and $\mathcal{S}$ connects only to I/O tapes of $\mathcal{Q}$. Note that this relationship is symmetric. We write $\mathcal{P} \,|\, \mathcal{Q}$, to make sure that $\mathcal{P}$ and $\mathcal{Q}$ communicate only via their external tapes (if necessary internal tapes are renamed).

Definition 1. Let $\mathcal{P}$ and $\mathcal{F}$ be I/O compatible protocol systems, the real and the ideal protocol, respectively. Then, $\mathcal{P}$ *SS-realizes* $\mathcal{F}$ ($\mathcal{P} \leq^{SS} \mathcal{F}$, "SS" stands for *strongly simulates*) iff there exists an adversarial system $\mathcal{S}$ adversarially connectible for $\mathcal{F}$ such that $\mathcal{P}$ and $\mathcal{S} \,|\, \mathcal{F}$ are compatible and for all environmental systems $\mathcal{E}$ which are connectible for $\mathcal{P}$ it holds that $\mathcal{E} \,|\, \mathcal{P} \equiv \mathcal{E} \,|\, \mathcal{S} \,|\, \mathcal{F}$.

In a similar way, other equivalent security notions such as black-box simulatability and (dummy) UC can be defined [20]. 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, as illustrated in Section 4.

Processing arbitrarily many messages of arbitrary length. We note that protocol systems can process and forward arbitrarily many messages of arbitrary length received via the I/O interface (and hence, enriching tapes) because of our definition of polynomial time (see Section 2.1). 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 arbitrarily 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.[1] 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 Section 4), where arbitrarily 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., [19, 20]). Whether or not to use this alternative is a matter of taste.

2.3. Composition Theorems

We restate the composition theorems from [20]. The first composition theorem handles concurrent composition of a fixed number of protocol systems. The second one guarantees secure composition of an unbounded number of copies of a protocol system.

[1]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.

Theorem 1. *([20]) Let $\mathcal{P}_1, \mathcal{P}_2, \mathcal{F}_1, \mathcal{F}_2$ be protocol systems such that $\mathcal{P}_1/\mathcal{F}_1$ is environmentally connectible for $\mathcal{P}_2/\mathcal{F}_2$ (and hence, vice versa), $\mathcal{P}_1 \,|\, \mathcal{P}_2$ and $\mathcal{F}_1 \,|\, \mathcal{F}_2$ are well-formed, and $\mathcal{P}_i \leq^{SS} \mathcal{F}_i$, for $i = 1, 2$. Then, $\mathcal{P}_1 \,|\, \mathcal{P}_2 \leq^{SS} \mathcal{F}_1 \,|\, \mathcal{F}_2$. (Note that an environment may connect to the free I/O tapes of both systems.)*

Theorem 2. *([20]) Let $\mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{P} \leq^{SS} \mathcal{F}$. Then, $!\underline{\mathcal{P}} \leq^{SS} !\underline{\mathcal{F}}$. (Recall that $\underline{\mathcal{P}}/\underline{\mathcal{F}}$ is the session version of $\mathcal{P}/\mathcal{F}$.)*

This theorem cannot only be interpreted as yielding multi-session realizations from single session realizations, but also as providing multi-party realizations from single party realizations (when $\underline{\mathcal{P}}$ and $\underline{\mathcal{F}}$ are considered as multi-party versions).

As an immediate consequence of the above theorems and using that $\leq^{SS}$ is reflexive, we obtain:

Corollary 1. *Let $\mathcal{Q}, \mathcal{P}, \mathcal{F}$ be protocol systems such that $\mathcal{Q} \,|\, !\underline{\mathcal{P}}$ and $\mathcal{Q} \,|\, !\underline{\mathcal{F}}$ are well-formed, $\mathcal{Q}$ is environmentally connectible for $!\underline{\mathcal{P}}$ and $!\underline{\mathcal{F}}$, and $\mathcal{P} \leq^{SS} \mathcal{F}$. Then, $\mathcal{Q} \,|\, !\underline{\mathcal{P}} \leq^{SS} \mathcal{Q} \,|\, !\underline{\mathcal{F}}$, i.e., $\mathcal{Q}$ using an unbounded number of copies of $\underline{\mathcal{P}}$ realizes $\mathcal{Q}$ using an unbounded number of copies of $\underline{\mathcal{F}}$.*

Iterated application of Theorem 1 and 2 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. This may yield simpler and potentially more efficient implementations, e.g., in case of global setups, for which the UC model had to be extended [8].

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 2, 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 $\underline{\underline{\mathcal{P}'}}$ 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 3. *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 1 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 3 immediately follows from Theorem 1 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 1, 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 1, 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 3 in the IITM model (given the composition theorem), the statement that Theorem 3 makes is stronger than that of the joint state theorem in the UC model [14, 6]. Inherited from our composition theorems, and unlike the theorem in the UC model, Theorem 3 does not require that $\mathcal{Q}$ completely shields the sub-protocol from the environment, and hence, from super-protocols on higher levels. This, as mentioned, can lead to simpler systems and more efficient implementations.

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 alone in the original UC model [4] or in the security parameter and the overall length of the input on the I/O interface in the new UC model [6]. As a consequence, by sending many messages to an ITM (on the network interface), the ITM can be forced to stop. Moreover, in general, a single ITM, say M, cannot simulate a concurrent composition of a fixed finite number of ITMs, say $M_1, \ldots, M_n$, or an unbounded number of (copies of) ITMs: By sending many messages to M intended for M_1, say, M will eventually stop, and hence, cannot simulate the other machines anymore, even though, in the actual composition these machines could still take actions.

Now, this causes problems in the joint state theorem of the UC model: Although the ITM $\widehat{\mathcal{F}}$ in that 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 illustrate the problem of realizing $\widehat{\mathcal{F}}$ in the original UC model, i.e., the one presented in [4], on which the work in [14] is based. The corresponding problem for the new version of the UC model [6] is explained in [22].

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.

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 [4] and the new version [6]. 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 (the simulator does not necessarily see how much resources $\mathcal{F}[pid]$ gets from the I/O interface and how much resources $\mathcal{F}[pid]$ has used), and hence, the simulator does not know when to terminate the corresponding copies in the ideal model. So, in these cases again the joint state theorem fails. In fact, just as in the case of realizing $\widehat{\mathcal{F}}$, 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.

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 3, 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 $!\mathcal{P}_{\mathrm{js}} \,|\, \mathcal{F}$ where $!\mathcal{P}_{\mathrm{js}}$ is a very basic protocol in which for every party only one copy of $\mathcal{P}_{\mathrm{js}}$ 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 $!\mathcal{P}_{\mathrm{js}}$ (with one copy per party) such that

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

Note that with $\mathcal{P} \leq^{SS} \mathcal{F}$, the composition theorems together with the transitivity of $\leq^{SS}$ imply that $!\mathcal{P}_{\mathrm{js}} \,|\, \mathcal{P} \leq^{SS} !\underline{\mathcal{F}}$. 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 $!\mathcal{P}_{\mathrm{js}} \,|\, !\underline{\mathcal{P}'} \leq^{SS} !\mathcal{P}_{\mathrm{js}} \,|\, !\underline{\mathcal{F}'} \leq^{SS} !\underline{\mathcal{F}} = !\underline{\underline{\mathcal{F}'}}$, 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: $!\underline{\mathcal{Q}} \,|\, !\underline{\underline{\mathcal{F}}} = !\underline{(\mathcal{Q} \,|\, !\underline{\mathcal{F}})} \geq^{SS} !\underline{(\mathcal{Q} \,|\, (!\mathcal{P}_{\mathrm{js}} \,|\, \mathcal{F}))} = !\underline{\mathcal{Q}} \,|\, !\underline{\mathcal{P}_{\mathrm{js}}} \,|\, !\underline{\mathcal{F}} \geq^{SS} !\underline{\mathcal{Q}} \,|\, !\underline{\mathcal{P}_{\mathrm{js}}} \,|\, !\mathcal{P}_{\mathrm{js}} \,|\, \mathcal{F}$, i.e., $!\underline{\mathcal{P}_{\mathrm{js}}} \,|\, !\mathcal{P}_{\mathrm{js}} \,|\, \mathcal{F}$ 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.

[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.

4. The Public-Key Encryption and Digital Signature Functionalities

In this section, we present our functionalities for (replayable) public-key encryption and digital signatures. We start with the functionality for public-key encryption. First, we introduce some notation and terminology.

Sending and receiving messages. We often formulate an IITM M parameterized by sets $\mathcal{T}$ of entities (e.g., adversary, environment, roles of users) that may write to/read from the IITM via some input and output tapes. We typically write "send m to T" for $T \in \mathcal{T}$ to mean that M writes m on some output tape connected to entity T and then waits for new input. Similarly, we write "If received m from T then …" with the obvious meaning.

Modeling corruption. As mentioned, corruption is not hardwired into the IITM model, but can be specified in a rigorous and flexible way as part of the specification of the protocols/functionalities. This in turn also means that the kind of corruption one would like to model has to be made explicit in the specification. In case the same kind of corruption is used in different protocols/functionalities, it is sometimes convenient to define a "corruption macro" which can be reused in different specifications.

One such macro is depicted in Figure 1. We use this macro in the specification of the functionalities in this paper. 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 (item (b)) and from now on forwards all messages between the network and I/O interface (item (c) and (d)), which is possible by the mechanism discussed in Section 2.2: Right before a message is forwarded from the network interface to the I/O interface (the user), the functionality expects to receive resources from the environment, i.e., a message of the form (Res, r) via the I/O interface (item (e)). The environment can ask whether the IITM is corrupted (item (a)); security notions otherwise would not make sense: a simulator $\mathcal{S}$ could corrupt a functionality at the beginning and then mimic the behavior of the real protocol. In (a) and (b) the 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 variable *corruptible* determines whether or not corruption is possible. 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, e.g., after some initialization, in order to capture non-adaptive corruption. In this paper, we use our macro to model both adaptive and non-adaptive corruption.

We note that forms of corruption other than the ones captured by our macro can also be modeled in the IITM model. For example, we can model that upon corruption the adversary does not take over the entire functionality but only certain parts. Also, the adversary might not learn the whole internal state of the functionality. In this case, it might make sense to first let the functionality compute some output (which the adversary might not be able to do since she does not know the entire internal state of the functionality) and let the adversary manipulate this output before sending it out. 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. Although these forms of corruption can be easily captured in the IITM model, the kinds of corruptions captured by our macro are best suited for the functionalities considered in this paper.

Finally, we note that in a system which consists of several layers of functionalities, corruption is typically specified for every functionality based on the corruption behavior on lower layers.

The public-key encryption functionality. Our public-key encryption functionality $\mathcal{F}_{\mathrm{pke}}$ is in the spirit of the one by Canetti [6] in that ciphertexts are computed and encrypted locally. However, as outlined below, in addition to being specified more rigorously, our formulation has several advantages. Let us first define $\mathcal{F}_{\mathrm{pke}}$. It is of the form

$$\begin{aligned}&\mathcal{F}_{\mathrm{pke}}(L, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)\\ &\quad= F_{\mathrm{dec}}(L, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p) \mid \,!\underline{F_{\mathrm{enc}}(\mathcal{T}_{\mathrm{enc}})}\end{aligned} \tag{2}$$

and specifies a public-key encryption functionality for one decryptor and an unbounded number of encryptors. The decryptor part F_{dec} and the encryptor part F_{enc} are formally specified in Figure 2 and 3, respectively. Let us first explain the parameters of $\mathcal{F}_{\mathrm{pke}}$:

The parameter $L = (L_\eta)_\eta$, called *leakage*, is a family of probabilistic polynomial time algorithms (*leakage algorithms*), where η is the security parameter. Each L_η is associated with an (efficiently decidable) domain $\mathrm{dom}(L_\eta)$. Intuitively, these algorithms capture how much of a plaintext may be leaked. For example, assume that the domain of L_η is the set of all messages of length at least η and on input $m \in \{0,1\}^{\geq \eta}$, L_η returns $0^{|x|}$. Such a leakage L models that the length of a message is leaked. Alternatively, one could define L_η to return a random bit string of length $|x|$ on input x. The advantage of the latter formulation is that it has, what we call, high entropy, a property that can be useful in the formulation of $\mathcal{F}_{\mathrm{pke}}$ (see below): L has *high entropy* if for every $x, y \in \mathrm{dom}(L_\eta)$, the probability that the bit strings returned by L_η on input x and y are the same is negligible (in η). We call L *length preserving* if $\mathrm{Prob}[|L_\eta(x)| = |x|] = 1$ for all η and $x \in \mathrm{dom}(L_\eta)$. The parameters $\mathcal{T}_{\mathrm{dec}}$ and $\mathcal{T}_{\mathrm{enc}}$ in (2) are sets of entities determining which entities may access (via tapes) the decryption

$\texttt{Corr}(\mathit{corrupted} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{corruptible} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{initialized} \in \{\mathsf{true}, \mathsf{false}\}, \mathit{corrMsg}, T_{\text{adv}}, \mathcal{T}_{\text{user}}, T_{\text{env}})$

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 received $(\mathsf{Corrupted?})$ from T_{env}, and $\mathit{initialized}$ do: send $(\mathit{corrupted})$ to T_{env}

(b) *Corruption:* If received $(\mathsf{Corrupt})$ from T_{adv}, $\mathit{corruptible}$, $\mathit{initialized}$, and not $\mathit{corrupted}$ do: $\mathit{corrupted} \leftarrow \mathsf{true}$, send $(\mathsf{Corrupted}, \mathit{corrMsg})$ to T_{adv}

(c) *Forward to A:* If received m from $T \in \mathcal{T}_{\text{user}}$, and $\mathit{corrupted}$ do: $\mathit{res} \leftarrow 0$, send (Recv, m, T) to T_{adv}

(d) *Forward to user:* If received (Send, m, T) from T_{adv} where $T \in \mathcal{T}_{\text{user}}$, $\mathit{corrupted}$, and $0 < |m| \leq \mathit{res}$ do: $\mathit{res} \leftarrow 0$, send m to T

(e) *Resources:* If received (Res, r) from T_{env}, and $\mathit{corrupted}$ do: $\mathit{res} \leftarrow |r|$, send (Res, r) to T_{adv}

Figure 1. Macro for modeling adaptive and non-adaptive corruption behavior.

and the encryption functionalities, respectively. Technically speaking, these parameters determine the set of tapes at the I/O interface of $\mathcal{F}_{\text{pke}}$. An overview of the tapes of $\mathcal{F}_{\text{pke}}$ is given in Figure 4. The I/O tapes of F_{dec} consist of a pair of tapes (one input and one output) for each $T \in \mathcal{T}_{\text{dec}}$, one output tape for each $T \in \mathcal{T}_{\text{enc}}$, one pair of tapes to connect to the environment, referred to as E_{dec}, and one pair of tapes to connect to F_{enc}. Furthermore, F_{dec} has one pair of network tapes, referred to as A_{dec}. Similarly, the tapes of F_{enc} consist of one pair of tapes for each $T \in \mathcal{T}_{\text{enc}}$, the environment ($E_{\text{enc}}$), F_{dec}, and the network (A_{enc}). All I/O tapes are enriching, except for the I/O tape from F_{dec} to F_{enc}, while the network tapes are consuming. The parameter p bounds the size and runtime of the encryption and decryption algorithms e and d provided by the adversary/simulator (see Figure 2, (b)). Since every potential encryption or decryption algorithm has polynomial runtime, p can always be chosen in such a way that the algorithms run as expected.

In (2), $F_{\text{enc}}(\mathcal{T}_{\text{enc}})$ is an IITM, depicted in Figure 3, which models that every encryptor using $\mathcal{F}_{\text{pke}}$ has its own local procedure for encryption, which can be corrupted independently of the decryptors decryption box F_{dec}. Upon corruption, F_{enc} relays all requests to/from the adversary. By writing $!\underline{F_{\text{enc}}(\mathcal{T}_{\text{enc}})}$ we model that an unbounded number of parties may use $\mathcal{F}_{\text{pke}}$ for encryption, where $\underline{F_{\text{enc}}(\mathcal{T}_{\text{enc}})}$ is the party version of $F_{\text{enc}}(\mathcal{T}_{\text{enc}})$. If a message m and a key k' are sent to F_{enc}, then in case F_{enc} is not corrupted the encryption request is forwarded to F_{dec} (see Figure 2, (e) and Figure 3, (c); F_{dec} expects the message from F_{enc} to be prefixed by some PID because it is designed to run together with the party version of F_{enc}), where upon first activation F_{enc} makes sure that F_{dec} is "awake" (see Figure 2, (g) and Figure 3, (a)). If F_{dec} is corrupted or k' is not the correct public key ($k' \neq k$), then m is encrypted by k' using the encryption algorithm e provided by the adversary (if any, Figure 2, (b)). Otherwise, the leakage of m is encrypted and it is checked whether the resulting ciphertext decrypts to the leakage again. In case of success, m and the ciphertext are recorded.

The IITM F_{dec} (see Figure 2) is used by the decryptor to decrypt messages. Before ciphertexts can be decrypted, the decryptor has to initialize F_{dec} (Figure 2, (a)), upon which the adversary is expected to provide encryption and decryption algorithms as well as a public-key (Figure 2, (b)) and then trigger the response to the decryptor, upon which the public key is handed to the decryptor (Figure 2, (c)). The decryptor can decrypt ciphertexts by sending (Dec, c) to F_{dec} (Figure 2, (d)). If (m, c) and (m', c) are recorded for plaintexts $m \neq m'$, the decryption of c is *ambiguous*. Therefore, $\perp$ 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 is sent to the decryptor. Upon corruption of the decryptor (Figure 2, (f)), the set H is revealed to the entity connected to A_{dec}, i.e., the simulator/environment. Since this entity also knows the encryption and decryption algorithms as well as the public-key, 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 5).

Features of $\mathcal{F}_{\text{pke}}$. Let us point out the main features of $\mathcal{F}_{\text{pke}}$. Clearly, in $\mathcal{F}_{\text{pke}}$ ciphertexts are computed locally, with the advantages discussed in the introduction. The most important distinguishing feature of $\mathcal{F}_{\text{pke}}$ compared to other formulations of public-key encryption functionalities with local computation is that $\mathcal{F}_{\text{pke}}$ is suitable for joint state realizations (see Section 5). Another distinguishing feature is that our formulation models the realistic setting that encryptions can be performed even before the decryption box has been invoked for the first time by the decryptor. Our functionality, including corruption, is specified rigorously (which is the reason it might may seem more complex than other formulations). Encryption and decryption boxes can be corrupted independently. The functional-

Functionality $F_{\mathrm{dec}}(L, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)$

Initialization: $\mathit{state} \leftarrow \mathsf{init}$, $\mathit{corrupted} \leftarrow \mathsf{false}$, $\mathit{nokey} \leftarrow \mathsf{true}$, $e, d, k \leftarrow \bot$, $H \leftarrow \emptyset$

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

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

(b) *Key Generation:* If received $(\mathsf{AlgAndKey}, e', d', k')$ from A_{dec}, nokey, and $|e'|, |d'|, |k'| \leq p(\eta)$ do: $(e, d, k) \leftarrow (e', d', k')$, $\mathit{nokey} \leftarrow \mathsf{false}$, send (Ack) to A_{dec}

(c) *Init Response:* If received (Inited) from A_{dec}, not nokey, and $\mathit{state} = (\mathsf{wait}, T)$ do: $\mathit{state} \leftarrow \mathsf{ok}$, send $(\mathsf{PublicKey}, k)$ to T

(d) *Decryption:* If received (Dec, c) from $T \in \mathcal{T}_{\mathrm{dec}}$, $\mathit{state} = \mathsf{ok}$, and not $\mathit{corrupted}$ 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 $(\mathsf{Plaintext}, m)$ to T

(e) *Encryption:* If received $(\mathit{pid}, \mathsf{Enc}, k', m, T)$ from F_{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 $\mathit{corrupted}$ then  $c \leftarrow \mathrm{sim}_{p(|m|+\eta)} e(k', m)$
else $\overline{m} \leftarrow L_\eta(m)$ ; $c \leftarrow \mathrm{sim}_{p(|\overline{m}|+\eta)}\, e(k, \overline{m})$ ; $m' \leftarrow \text{sim-det}_{p(|c|+\eta)}\, d(c)$ ; if $\overline{m} \neq m'$ then $c \leftarrow \bot$ end ;
    if $c \neq \bot$ then $H \leftarrow H \cup \{(m, c)\}$ end
end ;
send $(\mathit{pid}, \mathsf{Ciphertext}, c)$ to $T$
```

(f) *Corruption:* $\mathtt{Corr}(\mathit{corrupted}, \mathsf{true}, \mathit{state} \neq \mathsf{init}, H, A_{\mathrm{dec}}, \mathcal{T}_{\mathrm{dec}}, E_{\mathrm{dec}})$ (adaptive corruption, see Figure 1)

(g) *Wake up:* If received $(\mathit{pid}, \mathsf{WakeUpFromEnc})$ from F_{enc} do: send $(\mathit{pid}, \mathsf{Ack})$ to F_{enc}

If no rule above fires then produce no output

Figure 2. Ideal public-key encryption functionality $\mathcal{F}_{\mathrm{pke}} = F_{\mathrm{dec}} \mid !\underline{F_{\mathrm{enc}}}$, the decryptor's part F_{dec}, where $\mathrm{sim}_n\, A(x)$ means (probabilistic) simulation of A on input x for n steps. Similarly for sim-det, which forces the simulation to be deterministic.

ity $\mathcal{F}_{\mathrm{pke}}$ can be invoked an unbounded number of times, with arbitrarily long messages. Together with high entropy leakage, $\mathcal{F}_{\mathrm{pke}}$ guarantees that ciphertexts can only be guessed with negligible probability. Our functionality does not restrict the kind of algorithms provided by the simulator. This implicit universal quantification over all encryption and decryption algorithms simplifies the reasoning when using the functionality in a larger system and abstracts from details of the algorithms, resulting in simpler, "syntactic" proofs. For example, the proof of the joint state theorem is a purely "syntactic argument". We do not need to reason about probabilities or properties of the algorithms provided by the adversary/simulator. In fact, the joint state theorem even holds in an information theoretic setting, where no computational bounds are put on the environment. 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 $\mathcal{F}_{\mathrm{pke}}$.

In the following theorem, $\mathcal{P}_{\mathrm{pke}}(\Sigma, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}})$ denotes a realization induced by the encryption scheme Σ with domain $(\mathrm{dom}(L_\eta))_\eta$. In this realization, we allow for non-adaptive corruption of the decryptor and adaptive corruption of encryptors. While $\mathcal{F}_{\mathrm{pke}}$ allows for adaptive corruption also of the decryptor, the simulator can enforce non-adaptive corruption by blocking corrupt messages. However, formulating $\mathcal{F}_{\mathrm{pke}}$ in this way only makes this functionality more general. Basically, the following theorem shows that CCA-security is equivalent to realizing $\mathcal{F}_{\mathrm{pke}}$. Our technical report contains a slightly more general formulation of this theorem, which in particular implies that for all length preserving leakages, $\mathcal{F}_{\mathrm{pke}}$ can be realized.

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

Initialization: $state \leftarrow \mathsf{init}$, $corrupted \leftarrow \mathsf{false}$

CheckAddress: Accept all messages on all tapes

Compute: Upon receiving a message check which case matches:

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

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

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

(d) *Corruption:* $\texttt{Corr}(corrupted, \mathsf{true}, state \neq \mathsf{init}, \varepsilon, A_{\text{enc}}, \mathcal{T}_{\text{enc}}, E_{\text{enc}})$ (adaptive corruption, see Figure 1)

If no rule above fires then produce no output

Figure 3. Ideal public-key encryption functionality $\mathcal{F}_{\text{pke}} = F_{\text{dec}} \mid !\underline{F_{\text{enc}}}$, the encryptor's part F_{enc}.

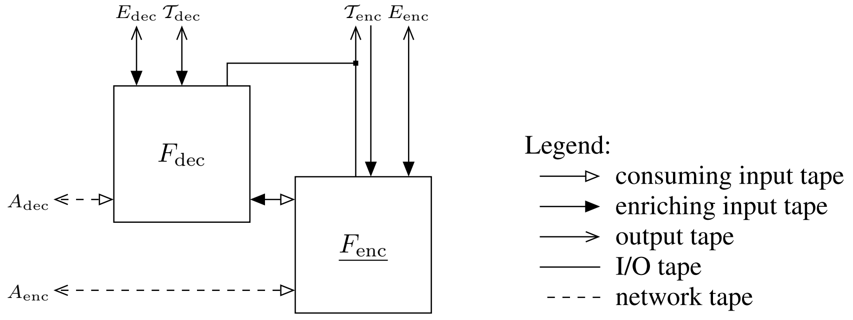


Figure 4. Graphical representation of the ideal public-key encryption functionality $\mathcal{F}_{\text{pke}} = F_{\text{dec}} \mid !\underline{F_{\text{enc}}}$.

Theorem 4. *For disjoint sets $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ of entities, a polynomial p, a public-key encryption scheme Σ whose algorithms run in time bounded by p, and leakage L, where exactly the length of a message is leaked, we obtain: Σ is CCA-secure if and only if*

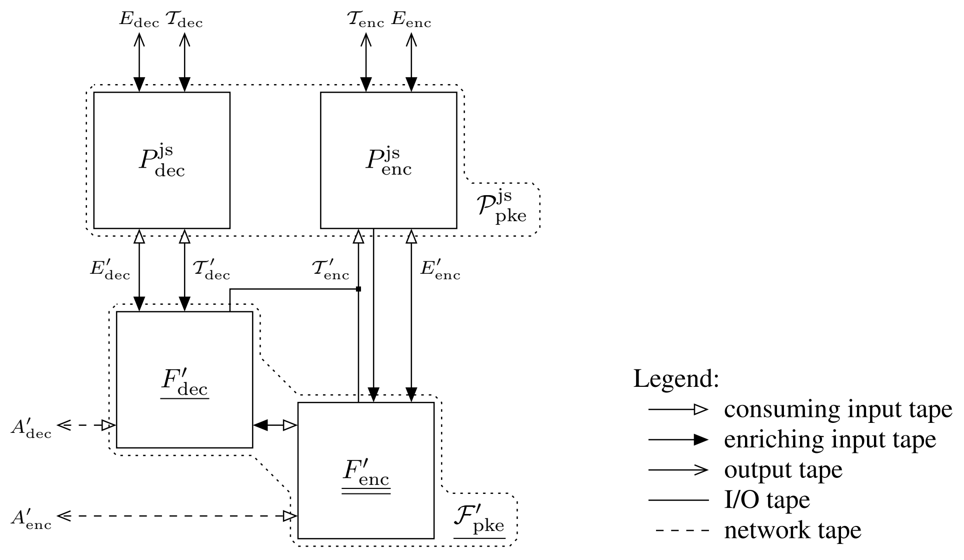
$$\mathcal{P}_{\text{pke}}(\Sigma, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}) \leq^{SS} \mathcal{F}_{\text{pke}}(L, \mathcal{T}_{\text{dec}}, \mathcal{T}_{\text{enc}}, p) \ .$$

Replayable public-key encryption and digital signatures. Recall from the introduction that replayable CCA-security (RCCA-security) is a relaxed form of CCA-security where modifications of the ciphertext that yield the same plaintext are permitted (see [11, 22]). Our functionality $\mathcal{F}_{\text{rpke}}$ for replayable public-key encryption is similar to $\mathcal{F}_{\text{pke}}$. The main difference is that instead of the pair (m, c), the pair $(m, \overline{m})$ is stored, where $\overline{m}$ is the result of the leakage algorithm applied to m. We have proven a theorem similar to Theorem 4, however, the leakage is required to have high entropy. For the specification of our digital signature functionality and our theorem stating the equivalence of this functionality and EU-CMA security, we also refer the reader to [22].

5. The Joint State Realizations

In this section, we present joint state realizations of our (replayable) public-key encryption and digital signatures functionalities. We again start with $\mathcal{F}_{\text{pke}}$. Recall from Section 3 and 4 that $\mathcal{F}_{\text{pke}} = F_{\text{dec}} \mid !\underline{F_{\text{enc}}}$ and that the goal is to obtain a joint state realization of $!\underline{\underline{\mathcal{F}_{\text{pke}}}}$, the multi-session, multi-party version of $\mathcal{F}_{\text{pke}}$, such that for every party only one instance of $\mathcal{F}_{\text{pke}}$ is created.

Our joint state realization of $!\underline{\underline{\mathcal{F}_{\text{pke}}}}$ is of the form $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}'_{\text{pke}}}$ with $\mathcal{P}^{\text{js}}_{\text{pke}} = \mathcal{P}^{\text{js}}_{\text{dec}} \mid \mathcal{P}^{\text{js}}_{\text{enc}}$ for IITMs $\mathcal{P}^{\text{js}}_{\text{dec}}$ and $\mathcal{P}^{\text{js}}_{\text{enc}}$, where $\mathcal{F}'_{\text{pke}}$ is obtained from $\mathcal{F}_{\text{pke}}$ by simply renaming the tapes. The renaming is necessary since the two systems—$!\underline{\underline{\mathcal{F}_{\text{pke}}}}$ and $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}'_{\text{pke}}}$—have to be I/O compatible. (Recall that $!\underline{\mathcal{F}'_{\text{pke}}}$ is the multi-party version of $\mathcal{F}'_{\text{pke}}$.) The connection between $\mathcal{P}^{\text{js}}_{\text{pke}}$ and $\mathcal{F}'_{\text{pke}}$ is pictured in Figure 5. The tapes of $\mathcal{P}^{\text{js}}_{\text{pke}}$ are explained below. The $\mathsf{CheckAddress}$ mode of $\mathcal{P}^{\text{js}}_{\text{dec}}$ is defined in such a way that for a decryptor pid at most one instance of $\mathcal{P}^{\text{js}}_{\text{dec}}$, referred to by $\mathcal{P}^{\text{js}}_{\text{dec}}[pid]$, is created in a run of $!\mathcal{P}^{\text{js}}_{\text{pke}} \mid !\underline{\mathcal{F}_{\text{pke}}}$.



E_{dec} $\mathcal{T}_{\text{dec}}$
$\mathcal{T}_{\text{enc}}$ E_{enc}
$P^{\text{js}}_{\text{dec}}$
$P^{\text{js}}_{\text{enc}}$
$\mathcal{P}^{\text{js}}_{\text{pke}}$
E'_{dec}
$\mathcal{T}'_{\text{dec}}$
$\mathcal{T}'_{\text{enc}}$
E'_{enc}
$\underline{F'_{\text{dec}}}$
A'_{dec}
$\underline{\underline{F'_{\text{enc}}}}$
A'_{enc}
$\underline{\mathcal{F}'_{\text{pke}}}$
Legend:
consuming input tape
enriching input tape
output tape
I/O tape
network tape

Figure 5. 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 $\underline{!\mathcal{F}'_{\text{pke}}}$.

Also, $\mathcal{P}^{\text{js}}_{\text{dec}}$ invokes only one instance of $\underline{F_{\text{dec}}}$, referred to by $\underline{F_{\text{dec}}}[pid]$. Together, $\mathcal{P}^{\text{js}}_{\text{dec}}[pid]$ and $\underline{F_{\text{dec}}}[pid]$ correspond to a local decryption box of party pid which handles all decryption requests of pid in all sessions, where $\mathcal{P}^{\text{js}}_{\text{dec}}[pid]$ serves as a multiplexer. It strips off the SID, say sid, before forwarding a request to $\underline{F_{\text{dec}}}[pid]$. Upon receiving a plaintext from $\underline{F_{\text{dec}}}[pid]$, it checks whether it is of the form (sid', m) with $sid' = sid$. Similarly, the CheckAddress mode of $\mathcal{P}^{\text{js}}_{\text{enc}}$ is defined in such a way that for a pair of PIDs pid and pid' at most one instance of $\mathcal{P}^{\text{js}}_{\text{enc}}$ is created, referred to by $\mathcal{P}^{\text{js}}_{\text{enc}}[pid, pid']$. This instance only invokes one instance of $\underline{\underline{F_{\text{enc}}}}$, referred to by $\underline{\underline{F_{\text{enc}}}}[pid, pid']$. Together $\mathcal{P}^{\text{js}}_{\text{enc}}[pid, pid']$ and $\underline{\underline{F_{\text{enc}}}}[pid, pid']$ correspond to the encryption box of an encryptor pid' for party pid which handles all encryption requests of pid' for pid in all sessions, where $\mathcal{P}^{\text{js}}_{\text{enc}}[pid, pid']$ serves as a multiplexer: When receiving an encryption request, say in session sid, for message m, $\mathcal{P}^{\text{js}}_{\text{enc}}[pid, pid']$ strips off sid from this requests and forwards the message (sid, m) to $\underline{\underline{F_{\text{enc}}}}[pid, pid']$, instead of just m. See [22] for a precise definition of $\mathcal{P}^{\text{js}}_{\text{pke}}$. Although the idea of prefixing messages with SIDs is simple and not new [14, 9], several subtleties have to be taken care of, which were overlooked in other works (see the explanation after Theorem 5).

The multiplexer $\mathcal{P}^{\text{js}}_{\text{pke}}$ (for decryption and encryption) is parameterized by sets of entities $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$ as well as a polynomial p just as $\mathcal{F}_{\text{pke}}$. In addition, another polynomial q is used to bound the length of SIDs. This is necessary for the joint state theorem to hold, but was overlooked in other works (after Theorem 5 other, more important issues are discussed). As depicted in Figure 5, the set of tapes of $\mathcal{P}^{\text{js}}_{\text{pke}}$ consist of all I/O tapes of $\mathcal{F}_{\text{pke}}$ plus the corresponding pairs of tapes for the connection to $\mathcal{F}'_{\text{pke}}$. In particular, $\mathcal{P}^{\text{js}}_{\text{pke}}$ occupies the environment tapes E'_{dec} and E'_{enc} of $\mathcal{F}'_{\text{pke}}$ which are used for corruption and resources (recall Figure 1). These are use by $\mathcal{P}^{\text{js}}_{\text{pke}}$ to forward corruption requests and resources to $\mathcal{F}'_{\text{pke}}$, where, however, the SID is striped off first (see [22] for more details).

We can now state the joint state theorem for $\mathcal{F}_{\text{pke}}$. As mentioned above, the joint state realization $\mathcal{P}^{\text{js}}_{\text{pke}}$ is based on the multi-party version $\underline{!\mathcal{F}'_{\text{pke}}}$ of the ideal functionality $\mathcal{F}'_{\text{pke}}$. Recall that $\mathcal{F}'_{\text{pke}}$ is obtained from $\mathcal{F}_{\text{pke}}$ by renaming tapes. More importantly, $\mathcal{F}'_{\text{pke}}$ will use a leakage L' that in addition to the leakage L in the ideal world $(\underline{\underline{!\mathcal{F}_{\text{pke}}}})$ also leaks the SID of the session in which a message was encrypted. This, in conjunction with the decryption test performed in $\mathcal{F}_{\text{pke}}$ (see Figure 2, (e)), will guarantee that ciphertexts generated in different sessions are different. This is crucial for the joint state theorem to hold (see below). We note that, as explained in Section 3, the following joint state theorem can be applied iteratively to obtain joint state realizations no matter on which sub-protocol layer $\mathcal{F}_{\text{pke}}$ is used. Since $\mathcal{F}_{\text{pke}}$ allows for adaptive corruption of both the decryptor and the encryptors, the theorem holds even in this case. (Implementations of $\mathcal{F}_{\text{pke}}$ will restrict corruption of the decryptor to be non-adaptive, though.) As mentioned, the theorem would even hold for computationally unbounded environments.

Theorem 5. *For all polynomials p and q and disjoint sets of entities $\mathcal{T}_{\text{dec}}$ and $\mathcal{T}_{\text{enc}}$, there exists a polynomial p' such* *that for every leakage L we have*

$$\begin{array}{c} !\mathcal{P}^{\mathrm{js}}_{\mathrm{pke}}(\mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p, q) \mid \underline{!\mathcal{F}'_{\mathrm{pke}}(L', \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p)} \\ \leq^{SS} \\ \underline{\underline{!\mathcal{F}_{\mathrm{pke}}(L, \mathcal{T}_{\mathrm{dec}}, \mathcal{T}_{\mathrm{enc}}, p')}} \end{array} \qquad (3)$$

where $L'((sid, m)) = (sid, L(m))$ if $m \in dom(L)$ for all SIDs sid and messages m.

By our composition theorems, we can replace the ideal functionality on the left-hand side of (3) by its realization as stated in Theorem 4, resulting in an actual joint state realization (without any ideal functionality) . We note that one could have tried to state and prove a joint state realization without resorting to the ideal functionality on the left-hand side of (3). However, this would have been much more complex, because we would have to state and 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 4, 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 joint state realizations are based on ideal functionalities.

The main idea behind the construction of the simulator used to prove Theorem 5 is as follows (see [22] for full details and proofs): If the simulator receives the public-key k, the encryption algorithm e, and the decryption algorithm d, then an instance of an ideal functionality with SID sid obtains the key k, the encryption algorithm $e_{sid}(\cdot,\cdot)$ and the decryption algorithm $d_{sid}(\cdot)$ from the simulator. Before encrypting a message, $e_{sid}(\cdot,\cdot)$ prefixes the message with sid. Conversely, $d_{sid}(\cdot)$ decrypts messages by running $d(\cdot)$ and after successful decryption, $d_{sid}(\cdot)$ checks whether the resulting plaintext is of the form (sid', m) for some m. Only if $sid' = sid$, $d_{sid}(\cdot)$ returns m as plaintext. Otherwise, an error message is returned. Upon corruption of a decryptor or encryptor, the simulator corrupts all recorded sessions of this decryptor or encryptor. In case a decryptor is corrupted, the simulator receives the sets H_{sid} for all sessions sid that are recorded. Then, the simulator returns the set $H = \bigcup_{sid}\{((sid, m), c) \mid (m, c) \in H_{sid}\}$ to the environment. For the simulation to work, it is required that the representation of H does not reveal the order in which elements have been added, e.g., a good representation would be a list of the elements in lexicographical order.

Subtleties in proving the theorem and problems in other joint state theorems. In the proof of this theorem, several subtleties come up which were overlooked in other works, in particular [6, 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 [6] and then for [9]. These explanations motivate and justify the definition of our functionality and the way our joint state theorem is stated (see [22] for more details).

Problems with the joint state realization in [6]. i) The public-key encryption functionality in [6], unlike our functionality, identifies the encryption algorithm e and the public-key. Upon encryption, the environment/user presents some message m and encryption algorithm e'. If e' is equal to the stored encryption algorithm e (which was given to the functionality by the adversary) the ciphertext c is computed as $e(\mu)$ (where μ is some fixed message). Otherwise, c is computed as $e'(m)$. But then, if 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, and hence, $e' \neq e$, the ciphertext is computed as $e'((sid, m))$, in the joint state world. In the ideal world, 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.

ii) In [6], the leakage is fixed to be the length of a message. 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 will fail: 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. However, this is not so if, as is the case in [6], SIDs are not leaked in the joint state world; even if the algorithms provided by the environment/simulator are assumed to be CCA-secure: For the sake of simplicity, let us assume that one tries to prove the joint state theorem in this setting using the following obvious simulator: It works similar to the simulator mentioned above, but in the uncorrupted case, $e_{sid}(k, \cdot)$ encrypts a constant message μ of the given length of the plaintext message plus the length of sid using $e(k, \cdot)$. (Prefixing sid is not an option, because this is not done in the joint state world; sid is not leaked.) Now, consider the following environment: It first asks to encrypt some message m in some session sid with the correct public-key/encryption algorithm. In both worlds, joint state and ideal, a ciphertext c computed by applying $e(k, \cdot)$ on the constant message μ of the length of m and sid is returned. Depending on μ and how pairings are encoded, we may assume that μ has the form of a pair (sid', m'') for some sid', with $sid' \neq sid$, and some plaintext m''. This is, for example, the case if μ is a sequence of 0's and 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. Now, the environment asks to decrypt c in session sid'. In the joint state world, c together with (sid, m) is stored in the public-key encryption functionality. Since $sid' \neq sid$, the decryption will fail. Conversely, in the instance of the public-key encryption functionality corresponding to sid' in the ideal world, c is not known, and hence, the functionality will try to decrypt c with $d_{sid'}(\cdot)$. This succeeds and m'' is returned. This behavior is different from the one in the joint state world, and hence, the environment can distinguish between the two worlds. Generally speaking, the above argument uses that in the joint state world, SIDs are not leaked, and hence, ciphertexts do not contain information about the session in which they were computed. As a result, decryptions in the joint state and ideal world have different observable effects. In [22] a more general argument is presented that applies to wider classes of simulators as well as leakage (including probabilistic leakage) and encodings of pairings. In this sense, the argument is robust. Even if one could define specific leakage and encodings of pairing for which the argument would fail, it would be at least highly 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. For this functionality, a theorem similar to Theorem 5 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 a similar way as prescribed by one of the simulators described above. 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 in the joint state world SIDs are handled outside of the ideal functionality, namely in the joint state realization. Hence, the joint state theorem would be defined with different ideal functionalities in the joint state and ideal world. These issues have 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 [6].

General remark. One general remark for joint state theorems is that specifying corruption precisely, as we do in our work, is vital, because 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. This is why we define corruption in such a way that this order is not revealed.

Replayable public-key encryption and digital signatures. For our replayable public-key encryption functionality and the digital signature functionality we also proved joint state theorems analogous to the one above (see [22]). For these functionalities similar subtleties occur.

Acknowledgment. We thank Ran Canetti for many helpful discussions on the UC model.

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] 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.

[5] 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.

[6] 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.

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

[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] 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.

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

[17] 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.

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

[19] D. Hofheinz, J. Müller-Quade, and D. Unruh. A Simple Model of Polynomial Time UC. One-page abstract of a talk given at the Workshop on Models for Cryptographic Protocols (MCP 2006), 2006.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.