# Universal Composition with Joint State*

Ran Canetti† Tal Rabin†

November 17, 2003

**Abstract**

Cryptographic systems often involve running multiple concurrent instances of some protocol, where the instances have some amount of joint state and randomness. (Examples include systems where multiple protocol instances use the same public-key infrastructure, or the same common reference string.) Rather than attempting to analyze the entire system as a single unit, we would like to be able to analyze each such protocol instance as stand-alone, and then use a general composition theorem to deduce the security of the entire system. However, no known composition theorem applies in this setting, since they all assume that the composed protocol instances have disjoint internal states, and that the internal random choices in the various instances are independent.

We propose a new composition operation that can handle the case where different components have some amount of joint state and randomness, and demonstrate sufficient conditions for when the new operation preserves security. The new operation, which is called *universal composition with joint state* (and is based on the recently proposed universal composition operation), turns out to be very useful in a number of quite different scenarios such as those mentioned above.

Keywords: cryptographic protocols, protocol composition, security analysis

*An extended abstract of this work appears in the proceedings of Crypto 2003.

†IBM T.J. Watson Research Center. E-mail: {canetti,talr}@watson.ibm.com.

# Contents

# 1 Introduction

Cryptographic systems often involve multiple concurrent instances of a variety of different protocols, where each protocol is relatively simple. Furthermore, even though the overall system may involve an unbounded number of parties, only few parties may participate in each protocol instance. Consequently, instead of directly analyzing the security of such a system as a single unit, we would like to be able to de-compose a complex cryptographic system to simpler components, prove the security of each component in isolation, and then deduce the security of the re-composed system.

This "de-compositional approach" is indeed very attractive. However, its soundness hinges on our ability to deduce the security of the entire system from the security of the components. Here *secure composition theorems* come in handy. Roughly speaking, such theorems assert that if a protocol is secure when considered in isolation, then it remains secure even when multiple instances of this protocol are run in the same system, or (in some cases) even when the protocol is used as a component of an arbitrary system.

A number of *composition operations* (i.e., ways to put together protocols in order to get a composite protocol) have been studied in the context of preservation of security. These include *sequential, parallel,* and *concurrent* composition, when the composed instances are run either by the same sets of parties, or by different sets of parties (e.g., [GK96, Bea91, DDN00, GO94, DNS98, Gol02]). They also include the more general operations of *modular* and *universal* composition, where protocols call other protocols as subroutines [MR91, Can00, DM00, PSW00, Can01]. However, all known composition theorems have the following limitation. They all assume that, at least as far as the honest parties are concerned, the local state of each one of the composed protocol instances is disjoint from the local states of all the other protocol instances run by the party. Furthermore, for each protocol instance, the honest parties are required to use a "fresh" random input that is independent of all the other random inputs. Thus, none of the known composition theorems is applicable when trying to de-compose a system into simpler components, while allowing the components to have some amount of joint state.

In contrast, many cryptographic systems consist of multiple concurrent instances of some (relatively simple) protocol, where all the instances have some limited amount of joint state and joint randomness. Prevalent examples include key-exchange and secure communication protocols, where multiple protocol instances use the same instance of a public-key signature or encryption scheme. Another set of examples include protocols in the common reference string model, where multiple instances use the same short reference string. Indeed, when attempting to analyze such systems, there was so far no alternative but to directly analyze the entire multi-instance system as a single unit.

We formulate a new composition operation for cryptographic protocols, that is applicable even in the case where multiple protocol instances have some amount of joint state. We also demonstrate sufficient conditions for when this operation preserves security. Our new operation, Universal Composition with joint state (JUC, pronounced "juicy"), is formulated within the Universal Composition (UC) framework [Can01], and extends its powers. The new operation drastically simplifies the analysis of systems where multiple instances have joint state, by allowing us to apply the de-composition methodology described above even to such systems. In fact, recent works which originally analyzed their security examining the entire protocol as a single unit have updated and simplified their proofs by utilizing the techniques presented in this paper [CK02, CLOS02].

## 1.1 The New Composition Theorem

We provide a very informal overview of the new composition theorem and its usage. Our system consists of a "high-level protocol", $\pi$, that uses multiple instances of a sub-protocol $\rho$, where the various instances of $\rho$ have some joint state.

To be able to use the JUC theorem, we need to have in hand a protocol, $\hat{\rho}$, where a single instance of $\hat{\rho}$ has essentially the same functionality as multiple independent copies of $\rho$. We then proceed as follows. We first analyze the overall protocol $\pi$ under the assumption that the copies of $\rho$ are independent. This can be done using known composition theorems, such as the UC theorem. We then replace all instances of $\rho$ within $\pi$ with a single instance of $\hat{\rho}$. The JUC Theorem essentially states that protocol $\pi$ behaves the same regardless of whether it is using multiple independent copies of $\rho$, or alternatively a single copy of $\hat{\rho}$.

The JUC theorem proves to be instrumental in de-composing complex systems. Using the terminology of

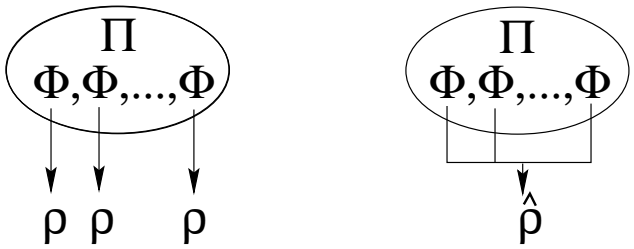


Figure 1: Universal Composition with Joint State: Protocol $\pi$ is analyzed under the assumption that the copies of $\rho$ are independent (left). This is in spite of the fact that all copies of $\rho$ are replaced by a single copy of $\hat{\rho}$ (right).

the previous paragraph, it allows us to de-compose our system into a "$\pi$ part" plus a "$\hat{\rho}$ part," analyze each part in isolation, and then deduce the security of the re-composed system. The important thing to notice is that the "$\pi$ part" treats all the copies of $\rho$ as if they were independent copies without any joint state. This proves to be very useful in cases where the "$\pi$ part" by itself consists of multiple copies of some other, simpler protocol $\phi$, where different copies of $\phi$ call different copies of $\rho$. We can now analyze each copy of $\phi$ as stand-alone, then compose all copies of $\phi$ into a single protocol $\pi$ (say, using the UC Theorem), and then use the JUC Theorem to compose $\pi$ with all the copies of $\rho$ as described above, in spite of the fact that the copies of $\rho$ (and consequently the copies of $\phi$) have joint state (see Figure 1).

As said above, in order for the JUC Theorem to hold, protocol $\hat{\rho}$ must exhibit the same behavior as multiple "independent" instances of $\rho$. (A formalization of this intuitive requirement appears within.) Clearly, the protocol $\hat{\rho}$ that simply runs multiple independent copies of $\rho$ would guarantee this "independence". However, such a protocol $\hat{\rho}$ would not be very interesting. The power of the JUC theorem is in the cases where protocol $\hat{\rho}$ is more efficient, and in particular makes meaningful use of joint state between copies of $\rho$. The rest of the introduction is dedicated to providing examples for the use of the JUC theorem.

## 1.2 Application to Protocols Using Digital Signatures

We exemplify the use of the JUC Theorem for de-composing systems where multiple protocol instances use the same instance of a signature scheme. Specifically, we concentrate on the prevalent example of key-exchange and secure channel protocols authenticated via digital signatures. Here, multiple parties run multiple instances of a (single session) key-exchange protocol in order to exchange multiple keys, while using the same public-key infrastructure which is the joint state. The exchanged keys are then used to establish secure communication channels. Previous analytical works of key-exchange and secure channel protocols treats the entire multi-party, multi-execution system as a single unit (see, e.g., [BR93, BCK98, Sho99, CK01]).

Here we show how, using the JUC Theorem, one can de-compose the system to individual sessions, analyze each session independently of all others, and deduce the security of the entire multi-session system. We proceed as follows. Using the terminology of the previous subsection, the key-exchange protocol (for exchanging a single key) is denoted $\phi$, and the multi-instance composition of $\phi$ is denoted $\pi$. We are given a protocol $\rho$ which satisfies the security requirements from a digital signature scheme. The protocol $\pi$ uses multiple calls to $\rho$, where different calls are made by different copies of $\phi$ within $\rho$. (We assume that each of these independent calls is assigned a unique identifier $i$. This assumption is central in our solution. See discussion about the identifiers in Section 2).

In order to be able to apply the JUC Theorem, we show how to construct a protocol $\hat{\rho}$ that behaves, within a single instance, like multiple independent copies of $\rho$. In fact, this can be done with essentially the complexity of a single instance of $\rho$. Protocol $\hat{\rho}$ runs a single copy of $\rho$. When the $i$th instance of $\phi$ invokes its instance of $\rho$ to generate a signature on message $m$, protocol $\hat{\rho}$ uses its single instance of $\rho$ to sign the message $(i, m)$. Similarly, whenever asked to verify whether $s$ is a signature of instance $i$ on a message $m$, $\hat{\rho}$ uses its single instance of $\rho$ to verify whether $s$ is a signature on $(i, m)$. We show that if $\rho$ is a secure signature protocol then $\hat{\rho}$ satisfies the conditions required by the JUC Theorem (i.e., $\hat{\rho}$ behaves essentially

like multiple independent copies of a signature scheme). This allows us to deduce the security of the entire, multi-session key exchange protocol even though we only analyzed the security of the single session protocol $\phi$ and the signature protocol $\hat{\rho}$.

The same methodology applies also to the treatment of secure session protocols. That is, it is possible to analyze the security of a single session protocol as stand alone, and then use the JUC theorem to deduce the security of the composite multi-session system — in spite of the fact that all sessions use the same instance of the signature scheme. Indeed, the updated version of [CK02] on key-exchange and secure session protocols has modified its presentation and analysis to utilize the JUC Theorem as proven here. Furthermore, the treatment of message authentication in [Can03] uses the JUC theorem in a central way.

### 1.3 Application to Protocols in the Common Reference String Model

A similar phenomenon happens in the case of protocols in the common reference string (CRS) model, where all parties have access to a reference string taken from a predefined distribution. We often have protocols where multiple instances use the same short reference string. These instances may be run either by the same set of parties or by different sets. So far, the only known way to analyze such multi-instance systems was to directly analyze them as a single unit. Using universal composition with joint state, we show how one can de-compose a multi-instance system to individual instances, analyze each instance in isolation, and then deduce security of the entire multi-instance system — in spite of the fact that all the protocol instances use the same short reference string. We demonstrate several alternative ways to go about this de-composition.

The first and most general way to de-compose multi-instance systems in the CRS model proceeds as follows. We first recall that the CRS model can be captured as the model that provide the parties with access to an "idealized protocol" $\rho$ that returns the same string to all parties, where the string is chosen from a predefined distribution. Let $\phi$ be a protocol that uses the CRS (i.e., $\phi$ invokes protocol $\rho$) and let $\pi$ be some protocol that runs multiple instances of protocol $\phi$. In order to prove the security of $\pi$, we proceed in four steps: (a) We prove the security of a single instance of $\phi$ using a single instance of $\rho$, in a stand-alone setting where no other protocol executions exist. (b) Using known composition theorems (e.g., the UC theorem), we deduce that the multi-instance protocol $\pi$ is also secure. Here, however, protocol $\pi$ uses multiple independent instances of $\rho$, which corresponds to having multiple independent copies of the reference string. (c) We construct a protocol, $\hat{\rho}$, that mimics the behavior of multiple independent copies of $\rho$, using only a single copy of $\rho$. In other words, $\hat{\rho}$ generates multiple "independent" copies of the reference string, given only a single copy of the string. (d) Using the JUC Theorem, we deduce that the entire composed protocol (consisting of the multi-instance protocol $\pi$, where all calls to all copies of $\rho$ are replaced with calls to a single instance of protocol $\hat{\rho}$) is secure. We stress that here all the copies of $\phi$ within $\pi$ use the same copy of protocol $\rho$, namely only a single instance of the reference string.

In order to complete the de-composition process, we need to come up with a protocol $\hat{\rho}$ that realizes multiple instances of $\rho$, given only a single instance of $\rho$. Our construction of protocol $\hat{\rho}$ is essentially the three-message coin-tossing-into-the-well protocol of Blum [Blu82], where the commitments are taken to be universally composable commitments, e.g. those of [CF01, CLOS02, DN02].

However, while the above de-composition method is quite general, it is not completely satisfactory because of the need to run the additional interactive protocol $\hat{\rho}$. In particular, in the composed protocol each copy of $\phi$ is interactive — even if the original construction of $\phi$ is non-interactive. We would like to be able to carry out the de-composition paradigm without paying the price in communication rounds.

At a first glance, it may appear that the way to avoid adding rounds is to come up with better constructions of protocol $\hat{\rho}$, which would be non-interactive. However, we show that this is not possible in the UC framework. That is, we show that *any* protocol which realizes in the UC framework multiple "independent" copies of the CRS, given only a single copy of the CRS, must be interactive. Essentially, each party must send at least one message per each new instance of the reference string. Furthermore, this message must be essentially at least as long as the generated string.

Given the impossibility of a non-interactive solution for the general problem of generating multiple CRSs given a single short CRS, we turn to other, less general ways to de-compose multi-instance systems in the CRS model. Specifically, we describe how our methodology can be applied to Zero-Knowledge (ZK) protocols and commitment protocols in the CRS model. Let us first sketch how this works for ZK protocols. Here we let protocol $\rho$ be a single-instance ZK protocol. That is, protocol $\rho$ carries out a single ZK proof. Assume we

have a protocol $\pi$ that consists of multiple copies of some protocol $\phi$, where each instance of $\phi$ uses (perhaps multiple) copies of $\rho$. We can now use the JUC theorem to replace all instances of $\rho$ with a single instance of protocol $\hat{\rho}$ that realizes multiple ZK proofs within a single instance. Such protocols $\hat{\rho}$ exist, and use only a single short instance of the CRS for all instances of the ZK proof [CF01, DCO+01]. In particular, the protocol of [DCO+01] is non-interactive.

In the case of commitment protocols we follow the same steps, with the exception that protocol $\rho$ is a commitment protocol for a single commitment-decommitment process. The "composite protocol" $\hat{\rho}$ now provides the functionality of multiple commitments and decommitments, while using only a single short instance of the CRS. Such protocols exist, e.g., those of [CF01, CLOS02, DN02].

We remark that our formalization and results for the CRS model play a central role in the updated proofs for the general construction in [CLOS02]. Earlier versions of the paper analyzed these constructions directly as multi-instance protocols and were considerably more cumbersome.

**Organization.** Section 2 briefly reviews the notion of UC security and the UC theorem of [Can01]; a more thorough review is deferred to the Appendix. Section 3 presents and proves the JUC Theorem. The applications to protocols in the CRS model and to protocols using signature schemes are presented in Section 4 and 5, respectively.

## 2 Review of the UC framework

We provide a brief review of the universally composable security framework [Can01]. The framework allows for defining the security properties of cryptographic tasks so that the security of protocols is maintained under a general composition operation with an unbounded number of instances of arbitrary protocols running concurrently in the system. This composition operation is called universal composition. Similarly, definitions of security in this framework are called universally composable (UC).

As in other general definitions (e.g., [GL90, MR91, Bea91, Can00, PSW00]), the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). Informally, a protocol securely carries out a given task if running the protocol with a realistic adversary amounts to "emulating" an ideal process where the parties hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. We call the algorithm run by the trusted party an ideal functionality.

In order to allow proving the universal composition theorem, the notion of emulation in this framework is considerably stronger than previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary, $\mathcal{A}$, that controls the communication channels and potentially corrupts parties. "Emulating an ideal process" means that for any adversary $\mathcal{A}$ there should exist an "ideal process adversary" (or, simulator) $\mathcal{S}$ that causes the *outputs* of the parties in the ideal process to have similar distribution to the outputs of the parties in an execution of the protocol. In the UC framework the requirement on $\mathcal{S}$ is more stringent. Specifically, an additional entity, called the environment $\mathcal{Z}$, is introduced. The environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol and the environment there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$.

The following *universal composition theorem* is proven in [Can01]. Consider a protocol $\pi$ that operates in the $\mathcal{F}$-hybrid model, where parties can communicate as usual, and in addition have ideal access to an unbounded number of *copies* of an ideal functionality $\mathcal{F}$. Let $\rho$ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let $\pi^\rho$ be identical to $\pi$ with the exception that the interaction with *each copy* of $\mathcal{F}$ is replaced with an interaction with a *separate instance* of $\rho$. Then, $\pi$ and $\pi^\rho$ have essentially the same input/output behavior. In particular, if $\pi$ securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then $\pi^\rho$ securely realizes $\mathcal{I}$ in the standard model (i.e. without ideal functionality).

**On the Session Identifiers (SID's).** Let us highlight one detail regarding the hybrid model that will become important in subsequent sections. Each copy of the ideal functionality $\mathcal{F}$ is assumed to have a unique identifier, called the session identifier (SID) of that copy. Each message sent to $\mathcal{F}$ in the hybrid model should contain a SID, and is then forwarded to the corresponding copy of $\mathcal{F}$. (If there is no such copy then a new one is invoked and given this SID.) Similarly, each message from a copy of $\mathcal{F}$ to a party contains the SID of that copy. The SIDs are determined by the protocol running in the hybrid model. Notice that this formalization allows each copy of $\mathcal{F}$, and each instance of the protocol that later replaces this copy, to know its own SID. It also guarantees that no two copies of $\mathcal{F}$ can ever have the same SID. This is essential for the composition theorem to hold. (See discussion below.) Let $\mathcal{F}(sid)$ denote the copy of functionality $\mathcal{F}$ with SID $=(sid)$.

We stress that the model does not specified how the parties learn, or "agree" on the SID. While we cannot always assume that a set of uncoordinated parties can agree on an SID, there are many interesting and important settings where this assumption is reasonable. As an example consider a network in which two-party protocols are being executed by a set of uncoordinated parties. In this case a unique SID can be easily chosen by the two parties $A$ and $B$ themselves, by choosing locally unique strings $r_A$ and $r_B$, respectively, which they exchange. The session id is defined to be $A \circ B \circ r_A \circ r_B$. Here an honest party is guaranteed to have a unique SID, even if the other party is cheating. Another example is a closed network of multiple users executing multiple protocols, where the protocols have some ordering and thus can be allotted unique SIDs.

Nonetheless, it should be stressed that in some cases the requirement for unique SIDs is imperative for providing security. For instance, in [LLR02] it is shown that Byzantine Agreement can not be reached in some specific settings unless unique identifiers are provided.

# 3 Universal Composition with Joint State (JUC)

Recall that the universal composition operation requires replacing each copy of $\mathcal{F}$ with a different invocation of protocol $\rho$, where all the invocations of $\rho$ in $\pi^\rho$ must have disjoint states and independent random inputs. However, in our setting the copies of $\rho$ have joint state. In fact, we wish to replace all the invocations of $\mathcal{F}$ with a *single* instance of some "joint protocol" $\hat{\rho}$. In order to do that, we essentially require that $\hat{\rho}$ has the same functionality as that of multiple independent copies of $\rho$. To formalize this requirement we define the multi-session extension of an ideal functionality. But first we define the composition operation.

**The Composition Operation.** The new composition operation, called universal composition with joint state (JUC), takes two protocols as arguments: a protocol $\pi$ in the $\mathcal{F}$-hybrid model and a protocol $\hat{\rho}$ realizing the multi-session functionality. The result is a composed protocol, denoted $\pi^{[\hat{\rho}]}$, and described as follows. Essentially, universal composition with joint state is identical to universal composition, with two exception: First, each party $P_i$ invokes only a single copy of $\hat{\rho}$ and replaces all calls to copies of $\mathcal{F}$ with activations of (the single copy of) $\hat{\rho}$. Second, now each activation of $\hat{\rho}$ includes *two* ids, the SID of $\hat{\rho}$ is set to some fixed, predefined value $sid_0$. The second id, SSID, is set to the original id of the invocation of $\mathcal{F}$. More specifically, protocol $\pi^{[\hat{\rho}]}$ behaves like $\pi$ with the following changes.

1. *Replacing calls to $\hat{\rho}$ for communication with $\mathcal{F}$:*
   (a) When activated for the first time within party $P_i$, $\pi^{[\hat{\rho}]}$ initiates a copy of protocol $\hat{\rho}$ with SID$= sid_0$.
   (b) Whenever $\pi$ instructs party $P_i$ to send a message $(sid, v)$ to $\mathcal{F}(sid)$, protocol $\pi^{[\hat{\rho}]}$ instructs $P_i$ to call $\hat{\rho}$ with input value $(sid_0, sid, v)$.
   (c) When (the single copy of) $\hat{\rho}$ generates an output value $(sid_0, sid, v)$ within $\pi^{[\hat{\rho}]}$, then $\pi^{[\hat{\rho}]}$ proceeds just as $\pi$ proceeds upon receiving output message $(sid, v)$ from $\mathcal{F}(sid)$.
2. *Dealing with messages sent by $\hat{\rho}$ and addressed to $\hat{\rho}$:*
   (a) Whenever protocol $\hat{\rho}$ wishes to send a message $m$, generated by the computation relating to $(sid_0, sid)$, to some party $P_j$, then $P_i$ writes the message $(sid_0, sid, m)$ on its outgoing communication tape.

(b) Upon delivery of a message $(sid_0, sid, m)$ from $P_j$, party $P_i$ activates $\hat{\rho}$ with incoming message $(sid_0, sid, m)$.

**The Multi-Session Extension of an Ideal Functionality.** We formalize the security requirements from the "joint protocol" $\hat{\rho}$. Intuitively, the requirement is that $\hat{\rho}$ should have essentially the same functionality as multiple independent invocations of $\rho$. More formally, we define the following ideal functionality, $\hat{\mathcal{F}}$, which we want $\hat{\rho}$ to realize. Let $\mathcal{F}$ be an ideal functionality. (Intuitively, $\mathcal{F}$ is the functionality realized by a single instance of $\rho$.) According to the UC formalization, $\mathcal{F}$ expects each incoming message to contain a special field consisting of its session identifier (SID). All messages received by $\mathcal{F}$ are expected to have the same value of the SID. (Messages that have different session identifier than that of the first message are ignored.) Similarly, all outgoing messages generated by $\mathcal{F}$ carry the same SID.

The multi-session extension of $\mathcal{F}$, denoted $\hat{\mathcal{F}}$, is defined as follows. $\hat{\mathcal{F}}$ expects each incoming message to contain *two* special fields. The first is the usual session identifier field as in any ideal functionality. The second field is called the sub-session identifier (SSID) field. Upon receiving a message $(sid, ssid, v)$ (where $sid$ is the SID, $ssid$ is the SSID, and $v$ is an arbitrary value or list of values), $\hat{\mathcal{F}}$ first checks if there is a running copy of $\mathcal{F}$ whose (single) session identifier is $ssid$. If so, then $\hat{\mathcal{F}}$ activates that copy of $\mathcal{F}$ with incoming message $(ssid, v)$, and follows the instructions of this copy. Otherwise, a new copy of $\mathcal{F}$ is invoked (within $\hat{\mathcal{F}}$) and immediately activated with input $(ssid, v)$. From now on, this copy is associated with sub-session identifier $ssid$. (That is, $ssid$ is the session identifier of this copy of $\mathcal{F}$.) Whenever a copy of $\mathcal{F}$ sends a message $(ssid, v)$ to some party $P_i$, $\hat{\mathcal{F}}$ sends $(sid, ssid, v)$ to $P_i$, and sends $ssid$ to the adversary. Sending $ssid$ to the adversary implies that $\hat{\mathcal{F}}$ does not hide which copy of $\mathcal{F}$ is being activated within $\hat{\mathcal{F}}$.

It is stressed that $\hat{\mathcal{F}}$ is not explicitly used by $\pi$. It only serves as a criterion for the security of $\hat{\rho}$. Furthermore, while $\hat{\mathcal{F}}$ consists of several copies of $\mathcal{F}$ with disjoint states, there is no requirement that the protocol $\hat{\rho}$ that realizes $\hat{\mathcal{F}}$ would have such structure. Indeed, $\hat{\rho}$ may use some joint state for realizing all the copies of $\mathcal{F}$ within $\hat{\mathcal{F}}$. Clearly, the case where $\hat{\rho}$ uses some joint state is the case of interest for this work, as otherwise we could use the regular universal composition.

It is convenient to state Theorem 1 using the general notion of protocol emulation. Informally, protocol $\pi$ in some $\mathcal{F}$-hybrid model emulates protocol $\pi'$ in some $\mathcal{F}'$-hybrid model if for any adversary $\mathcal{A}$ that interacts with parties running $\pi$ in the real-life model, there exists an adversary $\mathcal{A}'$ that interacts with parties running $\pi'$ in the $\mathcal{F}'$-hybrid model, such that no environment machine $\mathcal{Z}$ can tell whether it is interacting with $\pi$ and $\mathcal{A}$, or alternatively with $\pi'$ and $\mathcal{A}'$. For a more formal definition see the Appendix.

**Theorem 1 (Universal Composition with Joint State)** *Let $\mathcal{F}$ be an ideal functionality. Let $\pi$ be a protocol in the $\mathcal{F}$-hybrid model, and let $\hat{\rho}$ be a protocol that securely realizes $\hat{\mathcal{F}}$, the multi-session extension of $\mathcal{F}$, in the real-life model. Then the composed protocol $\pi^{[\hat{\rho}]}$ in the real-life model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model.*

**Proof:** Let $\mathcal{F}, \pi, \hat{\rho}$ be as in the theorem statement. We show that $\pi^{[\hat{\rho}]}$ in the real-life model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model. This is done in two steps: first we define a protocol $\tilde{\pi}$ and show that $\pi^{[\hat{\rho}]}$ in the real-life model emulates protocol $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model. Next we show that protocol $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model.

Protocol $\tilde{\pi}$ is a slight variation of protocol $\pi$, operating in the $\hat{\mathcal{F}}$-hybrid model. Specifically, $\tilde{\pi}$ is identical to $\pi$, with the following exceptions. (Note that $\tilde{\pi}$ invokes only a single copy of $\hat{\mathcal{F}}$ throughout the computation.)

1. Whenever $\pi$ instructs $P_i$ to send a message $(sid, v)$ to some copy $\mathcal{F}(sid)$ of $\mathcal{F}$, $\tilde{\pi}$ instructs $P_i$ to send $(sid_0, sid, v)$ to $\hat{\mathcal{F}}$.

2. Whenever some party $P_i$, running $\tilde{\pi}$, receives a message $(sid_0, sid, v)$ from $\hat{\mathcal{F}}$, it follows the instructions of $\pi$ upon receipt of the message $(sid, v)$ from $\mathcal{F}(sid)$.

Recall that $\tilde{\pi}^{\hat{\rho}}$ is the protocol obtained by applying the universal composition operation of [Can01] to protocols $\tilde{\pi}$ and $\hat{\rho}$. Since protocol $\hat{\rho}$ securely realizes $\hat{\mathcal{F}}$, it follows from the universal composition theorem that protocol $\tilde{\pi}^{\hat{\rho}}$ in the real-life model emulates protocol $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model. Furthermore, it is easy to see that protocol $\tilde{\pi}^{\hat{\rho}}$ is identical to protocol $\pi^{[\hat{\rho}]}$. (These are two different descriptions of exactly the same protocol.) We thus have that protocol $\pi^{[\hat{\rho}]}$ in the real-life model emulates protocol $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model.

It remains to show that protocol $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model. This is done as follows. Let $\hat{\mathcal{A}}$ be an adversary that interacts with parties running $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model. We construct an adversary $\mathcal{A}$ such that no environment will be able to tell whether it is interacting with $\mathcal{A}$ and $\pi$ in the $\mathcal{F}$-hybrid model or with $\hat{\mathcal{A}}$ and $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model. Adversary $\mathcal{A}$ follows the instructions of $\hat{\mathcal{A}}$, with the following exceptions:

1. Whenever $\mathcal{A}$ is notified that some copy $\mathcal{F}(sid)$ of $\mathcal{F}$ has sent a message with identifier $id$ to some party $P_i$, $\mathcal{A}$ records the pair $(sid, id)$ and notifies $\hat{\mathcal{A}}$ that $\hat{\mathcal{F}}_{(sid_0)}$ has sent a message with identifier $id$ to $P_i$. Note that $\mathcal{A}$ does not see the actual content of the message, but is only aware of the fact that a message has been sent.

2. Whenever $\hat{\mathcal{A}}$ delivers a message with identifier $id$ from $\hat{\mathcal{F}}$ to $P_i$ (in the $\hat{\mathcal{F}}$-hybrid model), $\mathcal{A}$ looks up the pair $(sid, id)$ and delivers the message with identifier $id$ from $\mathcal{F}(sid)$ to $P_i$.

3. When $\hat{\mathcal{A}}$ corrupts a party $P_i$ (running $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model), $\mathcal{A}$ corrupts $P_i$ (running $\pi$ in the $\mathcal{F}$-hybrid model) and obtains the internal state of $P_i$ for protocol $\pi$. It then 'translates' the internal state of $P_i$ for protocol $\pi$ to be consistent with an internal state of $P_i$ for protocol $\tilde{\pi}$, and hand this information to $\hat{\mathcal{A}}$. (The translation is straightforward: calls to multiple copies of $\mathcal{F}$ are translated into calls to a single copy of $\hat{\mathcal{F}}$, with a fixed session identifier $sid_0$, and the corresponding SSIDs.)

It is straightforward to verify that the view of $\mathcal{Z}$ in an interaction with $\mathcal{A}$ and $\pi$ in the $\mathcal{F}$-hybrid model is distributed identically to the view of $\mathcal{Z}$ in an interaction with $\hat{\mathcal{A}}$ and $\tilde{\pi}$ in the $\hat{\mathcal{F}}$-hybrid model. □

We remark that if $\hat{\rho}$ operates in the $\mathcal{G}$-hybrid model for some ideal functionality $\mathcal{G}$, rather than in the real-life model, then $\pi^{[\hat{\rho}]}$ also operated in the $\mathcal{G}$-hybrid model. In this case, we have that $\pi^{[\hat{\rho}]}$ in the $\mathcal{G}$-hybrid model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model.

# 4 Application to Protocols in the CRS Model

This section exemplifies the use of the JUC theorem for protocols in the common reference string model. As discussed in the Introduction, we approach this issue in two alternative ways. The first is to take a single CRS and "stretch" it into multiple independent CRSs, which will later be utilized by protocols which require independent CRS's. The second is to examine a specific multi-session functionality which employs a CRS and to directly generate the multi-session functionality from the single CRS, and prove its security directly. While the first approach is more general, the second approach will result in more efficient protocols for specific functionalities.

Though the introduction starts with the first method, here we start with a specific example of commitments in the CRS model. This order of presentation is motivated by the fact that our protocol for stretching the CRS will need to employ a multi-session commitment functionality.

## 4.1 Commitment in the CRS Model

Let us first recall the ideal commitment functionality, $\mathcal{F}_{\text{COM}}$, as defined in [CF01] (see Figure 2). Each copy of $\mathcal{F}_{\text{COM}}$ handles the process of a single commitment followed by its single decommitment.

**Functionality $\mathcal{F}_{\text{COM}}$**

$\mathcal{F}_{\text{COM}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$.

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, x)$ from $P_i$, record the value $x$ and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to $P_j$ and $\mathcal{S}$. Ignore any subsequent `Commit` messages.
2. Upon receiving a value $(\texttt{Open}, sid, P_i, P_j)$ from $P_i$, proceed as follows: If some value $x$ was previously recoded, then send the message $(\texttt{Open}, sid, P_i, P_j, x)$ to $P_j$ and $\mathcal{S}$ and halt. Otherwise halt.

Figure 2: The Ideal Commitment functionality

The universal composition theorem allows us to write a protocol $\pi$ in the $\mathcal{F}_{\text{COM}}$-hybrid model, and then replace each copy of $\mathcal{F}_{\text{COM}}$ with an instance of a protocol $\rho$ that securely realizes $\mathcal{F}_{\text{COM}}$. However, we only know how to realize $\mathcal{F}_{\text{COM}}$ in the common reference string model, or in other words in the $\mathcal{F}_{\text{CRS}}$-hybrid model, where $\mathcal{F}_{\text{CRS}}$ is the ideal functionality that provides all parties with a common string taken from some pre-specified distribution. Consequently, if the universal composition theorem is used as is, then each instance of $\rho$ in the composed protocol $\pi^\rho$ would use a different copy of $\mathcal{F}_{\text{CRS}}$ (i.e., an independent copy of the reference string).

Instead, we are looking for a protocol where all copies of the commitment protocol use the same short reference string. In [CF01], this is solved as follows. First, they define an additional ideal functionality, $\mathcal{F}_{mcom}$, where a single copy handles multiple commitments and decommitments by different parties and to different messages. Next, they construct a protocol, $\mathsf{UCC}_{\mathsf{ReUse}}$, that securely realizes $\mathcal{F}_{mcom}$ and uses a single copy of $\mathcal{F}_{\text{CRS}}$ for all the commitments. Thus, if the high-level protocol $\pi$ is written in the $\mathcal{F}_{mcom}$-hybrid rather than the $\mathcal{F}_{\text{COM}}$-hybrid model, then the composed protocol, $\pi^{\mathsf{UCC}_{\mathsf{ReUse}}}$, uses a single copy of the reference string for multiple commitments.

However, having the high-level protocol $\pi$ use $\mathcal{F}_{mcom}$ rather than $\mathcal{F}_{\text{COM}}$ comes at a price in the analysis of the protocol. In order to guarantee that a single copy of $\mathcal{F}_{\text{CRS}}$ is used throughout the computation, $\pi$ has to use only a single copy of $\mathcal{F}_{mcom}$ (since each copy of $\mathcal{F}_{mcom}$ uses a different copy of $\mathcal{F}_{\text{CRS}}$). This puts a considerable restriction on the analysis of $\pi$, as the security needs to be proven for $\pi$ as a single unit. This holds even if $\pi$ consists of multiple instances of some simpler protocol $\rho$. Thus, much of the advantage of using the UC theorem is lost.

These restrictions can be avoided using universal composition with joint state. We first observe that the functionality $\mathcal{F}_{mcom}$ is nothing but a reformulation of $\hat{\mathcal{F}}_{\text{COM}}$. The JUC Theorem thus says that if $\pi$ is a protocol in the $\mathcal{F}_{\text{COM}}$-hybrid model (and uses as many copies of $\mathcal{F}_{\text{COM}}$ as it wishes) then the composed protocol $\pi^{[\mathsf{UCC}_{\mathsf{ReUse}}]}$ runs in the $\mathcal{F}_{\text{CRS}}$-hybrid model, emulates $\pi$, *and uses only a single copy of* $\mathcal{F}_{\text{CRS}}$. In other words, we can allow protocol $\pi$ (and all the higher level protocols that may use $\pi$ as a subroutine) to operate in an idealized model where commitments are completely independent of each other, and then use the JUC Theorem to implement all the commitments using a single short common string.

## 4.2 Protocols for Stretching the Common Reference String

This section investigates the possibility of realizing $\hat{\mathcal{F}}_{\text{CRS}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model. More specifically, we present a protocol $\rho$ that securely realizes $\hat{\mathcal{F}}_{\text{CRS}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model, and uses only a single copy of $\mathcal{F}_{\text{CRS}}$. Protocol $\rho$ allows the parties to generate multiple, computationally independent copies of the random string, using a single copy of the string. Then we can design protocols in the $\mathcal{F}_{\text{CRS}}$-hybrid model, where each protocol instance can assume that no other instances have access to the reference string it is using, and then replace all copies of $\mathcal{F}_{\text{CRS}}$ with a single copy of the reference string.

For simplicity we consider only the case where the reference string is taken from the uniform distribution over $\{0,1\}^t$ for some $t$. Also, we restrict attention to protocols where only *two* parties need to have access to the reference string. It seems that this special case captures much of the essence of the problem.

We first show a simple protocol that securely realizes $\hat{\mathcal{F}}_{\text{CRS}}$ (for two parties and with uniform distribution) using a single copy of $\mathcal{F}_{\text{CRS}}$. The protocol requires interaction between the two parties in order to generate each new copy of the reference string. We then demonstrate that *any* protocol that realizes $\hat{\mathcal{F}}_{\text{CRS}}$ must involve sending at least one message by each of the participants. Furthermore, a new message must be sent for obtaining essentially any new copy of the reference string.

Let us first formulate functionality $\mathcal{F}_{\text{CRS}}$ for the restricted case described above (Fig. 3). The functionality is parameterized by $t$, the length of the reference string. Note that $\mathcal{F}_{\text{CRS}}$ does not limit the identities of the parties that may obtain the common random value $r$. The number (and identities) or parties that actually obtain the string is determined by the protocol that realizes $\mathcal{F}_{\text{CRS}}$.

### 4.2.1 Realizing $\hat{\mathcal{F}}_{\text{CRS}}$.

We show how to realize $\hat{\mathcal{F}}_{\text{CRS}}$, using a single copy of $\mathcal{F}_{\text{CRS}}$, for the case where only two parties obtain each copy of the reference string. The protocol is essentially the coin-tossing-into-the-well protocol (*ct*) of Blum [Blu82], which employs a commitment scheme. We use a universally composable commitment that utilize

**Functionality $\mathcal{F}_{\text{CRS}}^t$**

$\mathcal{F}_{\text{CRS}}$ proceeds as follows, running with parties $P_1, ..., P_n$, and an adversary $\mathcal{S}$, parameterized by an integer $t$.

1. When receiving (CRS,$sid, P_i, P_j$) from $P_i$, choose a value $r \stackrel{R}{\leftarrow} \{0,1\}^t$, send (CRS,$sid, r$) to $P_i$, and send $(sid, P_i, P_j, r)$ to $\mathcal{S}$. Next, when receiving (CRS,$sid, P_i, P_j$) from $P_j$ (and only $P_j$), send (CRS,$sid, r$) to $P_j$ and $\mathcal{S}$, and halt.

Figure 3: The Common Random String functionality

the single common random string. We first present the protocol in the $\mathcal{F}_{\text{COM}}$-hybrid model, and then use the JUC Theorem to compose this protocol with protocol $\mathsf{UCC}_{\mathsf{ReUse}}$ of [CF01] (as discussed in Section 4.1), and obtain the desired protocol. This means that the distribution needed for our protocol is the distribution needed for $\mathsf{UCC}_{\mathsf{ReUse}}$. The protocol in the $\mathcal{F}_{\text{COM}}$-hybrid model is denoted $ct^t$. Whenever parties $P_i$ and $P_j$ are invoked to generate a new copy of the reference string (say, with SID $sid$), they proceed as follows.

**Message 1:** When activated with input CRS,$sid, P_i, P_j$), $P_i$ chooses a random string $r_i \stackrel{R}{\leftarrow} \{0,1\}^t$ and commits to $r_i$ for $P_j$, using a new copy of $\mathcal{F}_{\text{COM}}$ with SID $sid$. (That is, $P_i$ sends $(\texttt{Commit}, sid, P_i, P_j, r_i)$ to $\mathcal{F}_{\text{COM}}$).

**Message 2:** When activated with input CRS,$sid, P_i, P_j$), $P_j$ waits to receive $(\texttt{Receipt}, sid, P_i, P_j)$ from $\mathcal{F}_{\text{COM}}$. Next, $P_j$ chooses $r_j \stackrel{R}{\leftarrow} \{0,1\}^t$ and sends to $P_i$.

**Message 3:** Upon receiving $r_j$, $P_i$ decommits to $r_i$. (That is, $P_i$ sends $(\texttt{Open}, sid, P_i, P_j)$ to $\mathcal{F}_{\text{COM}}$).

**Output:** Both parties output the string $r_i \oplus r_j$.

**Claim 2** *Protocol $ct^t$ securely realizes $\hat{\mathcal{F}}_{\text{CRS}}^t$ in the $\mathcal{F}_{\text{COM}}$-hybrid model.*

**Proof:** Let $\mathcal{A}$ be an adversary that interacts with protocol $ct^t$ in the $\mathcal{F}_{\text{COM}}$-hybrid model. We construct an adversary $\mathcal{S}$ so that no environment can tell whether it is interacting with $\mathcal{S}$ in the ideal process for $\hat{\mathcal{F}}_{\text{CRS}}^t$ or with $\mathcal{A}$ and $ct$ in the $\mathcal{F}_{\text{COM}}$-hybrid model. Adversary $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$, and proceeds as follows. (Without loss of generality we assume that $\mathcal{A}$ does not run the protocol between two corrupted parties.)

1. Throughout, whenever $\mathcal{S}$ receives an input value from $\mathcal{Z}$, it copies this value to $\mathcal{A}$'s input tape. Whenever $\mathcal{A}$ writes a value on its output tape, $\mathcal{S}$ copies this value to its own output tape (to be read by $\mathcal{Z}$).

2. **Corrupted initiator:** If the simulated $\mathcal{A}$ generates a message $(\texttt{Commit}, sid, P_i, P_j, r_i)$ from a corrupted party $P_i$ to $\mathcal{F}_{\text{COM}}$, then $\mathcal{S}$ records $r_i$, and sends a message $(\texttt{CRS}, 0, sid, P_i, P_j)$ from $P_i$ to $\hat{\mathcal{F}}_{\text{CRS}}$ in the ideal process. (That is, the SID of this message is 0, and the SSID is $sid$.) Upon receiving a value $r$ from $\hat{\mathcal{F}}_{\text{CRS}}$, $\mathcal{S}$ waits to receive a message (CRS,$(sid, P_j)$ from $\hat{\mathcal{F}}_{\text{CRS}}$. This message will notify $\mathcal{S}$ that $P_j$ was activated with input $(\texttt{CRS}, 0, sid, r)$. Upon receiving this message, $\mathcal{S}$ sets $r_j = r \oplus r_i$ and activates $\mathcal{A}$ to receive the message $r_j$ from $P_j$. Next, when $\mathcal{A}$ generates the message $(\texttt{Open}, sid, P_i, P_j)$ from $P_i$ to $\mathcal{F}_{\text{COM}}$, $\mathcal{S}$ delivers the message that $\hat{\mathcal{F}}_{\text{CRS}}$ sent to $P_j$. (This message contains the value $r$.)

3. **Corrupted responder:** If $\mathcal{S}$ receives the message $(0, sid, P_i, P_j, r)$ from $\hat{\mathcal{F}}_{\text{CRS}}$ in the ideal process where $P_j$ is corrupted, then $\mathcal{S}$ activates the simulated $\mathcal{A}$ to receive message $(\texttt{Receipt}, sid, P_i, P_j)$ from $\mathcal{F}_{\text{COM}}$. When $\mathcal{A}$ generates a message $r_j$ from $P_j$, $\mathcal{S}$ delivers the message from $\hat{\mathcal{F}}_{\text{CRS}}$ to $P_i$ in the ideal process, and activates $\mathcal{A}$ to receive the message $(\texttt{Open}, sid, P_i, P_j, r_i)$ from $\mathcal{F}_{\text{COM}}$, where $r_i = r \oplus r_j$.

4. **Both parties uncorrupted:** If $\mathcal{S}$ receives the message $(0, sid, P_i, P_j, r)$ from $\hat{\mathcal{F}}_{\text{CRS}}$ in the ideal process where both $P_i$ and $P_j$ are uncorrupted, then it simulates for $\mathcal{A}$ the information it sees when two uncorrupted parties run the protocol and obtain a common string $r$. This information consists of a notice from $\mathcal{F}_{\text{COM}}$ that $P_i$ committed to a value to $P_j$, the random value $r_j$ sent by $P_j$, and the opening of the commitment to $r_i$ such that $r_i \oplus r_j = r$.

5. **Party corruption:** If at any point the simulated $\mathcal{A}$ corrupts a party $P_i$ then $\mathcal{S}$ corrupts $P_i$ in the ideal process, and provides $\mathcal{A}$ with the simulated internal state of $P_i$. (It is easy to verify that this state is always implied by the information already known to $\mathcal{S}$ at the time of corruption.)

Let $\mathcal{Z}$ be an environment machine. It is straightforward to verify that the view of $\mathcal{Z}$ in the ideal process for $\hat{\mathcal{F}}_{\text{CRS}}$ with $\mathcal{S}$ is distributed identically to its view of an execution of $ct$ in the $\mathcal{F}_{\text{COM}}$-hybrid model. Note that this holds even if $\mathcal{Z}$ is computationally unbounded. □

Using the JUC Theorem, we have that protocol $ct^{[\mathsf{UCC}_{\mathsf{ReUse}}]}$ securely realizes $\hat{\mathcal{F}}_{\text{CRS}}$ and uses only a single copy of $\mathcal{F}_{\text{CRS}}$.

### 4.2.2 Limits on Protocols for Realizing $\hat{\mathcal{F}}_{\text{CRS}}$.

We show that any protocol that realizes $\hat{\mathcal{F}}_{\text{CRS}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model, and uses only a few copies of $\mathcal{F}_{\text{CRS}}$, must involve interaction. More specifically, we consider protocols that realize $\mathcal{F}^s_{\text{CRS}}$ using a single copy of $\mathcal{F}^t_{\text{CRS}}$, where $s > t$. (This is a considerable restriction of the original problem: Clearly, any protocol that securely realizes $\hat{\mathcal{F}}_{\text{CRS}}$ realizes also $\mathcal{F}^s_{\text{CRS}}$ for any $s$ that is polynomial in $t$.) We show that any such protocol must require that each of the parties send at least $s - t$ bits of information. Translated back to the task of realizing $\hat{\mathcal{F}}^t_{\text{CRS}}$ using a single copy of $\mathcal{F}^t_{\text{CRS}}$, this bound implies that each party must send at least $t$ bits in order to generate each new copy of the string. (Note that the bound holds also for protocols generating a string among more than two parties.) That is:

**Claim 3** *Let $\pi$ be a protocol that securely realizes $\mathcal{F}^s_{\text{CRS}}$ using a single copy of $\mathcal{F}^t_{\text{CRS}}$, and assume that $\pi$ requires one of the parties to send no more than $u$ bits of information. Then $s \leq t + u$.*

**Proof:** Let $\pi$, $s$, $t$, $u$ be as in the premise of the claim. It follows that for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between an interaction with $\pi$ and $\mathcal{A}$ in the $\mathcal{F}^t_{\text{CRS}}$-hybrid model and an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}^s_{\text{CRS}}$. Let $P_i$ be the party that sends at most $u$ bits.

Consider the following adversary $\mathcal{A}$ and environment $\mathcal{Z}$. $\mathcal{Z}$ instructs $\mathcal{A}$ to corrupt all parties except for $P_i$, uniformly chooses random inputs $\alpha_1, ..., \alpha_n$ for the corrupted parties, and instructs $\mathcal{A}$ to have each corrupted party $P_j$ run $\pi$ with random input $\alpha_j$. $\mathcal{A}$ follows the instructions of $\mathcal{Z}$ and reports the gathered information to $\mathcal{Z}$. This information consists of the $t$-bit value $r_t$ obtained from $\mathcal{F}^t_{\text{CRS}}$, and the $u$-bit concatenation, $m$, of all messages received from $P_i$. Next, $\mathcal{Z}$ obtains the $s$-bit output of $P_i$, $r_s$, picks a corrupted party $P_j$, and outputs 1 iff $P_j$ outputs $r_s$ after running $\pi$ on random input $\alpha_j$, and having received $r_t$ from $\mathcal{F}^t_{\text{CRS}}$ and messages $m$ from $P_i$.

Clearly, if $\mathcal{Z}$ interacts with parties running $\pi$ in the $\mathcal{F}^t_{\text{CRS}}$-hybrid model, then it always outputs 1. On the other hand, let $\mathcal{S}$ be an ideal-process adversary. We claim that if $\mathcal{Z}$ interacts with $\mathcal{S}$ in the ideal process for $\mathcal{F}^s_{\text{CRS}}$ then $\mathcal{Z}$ outputs 1 with probability at most $2^{t+u-s}$. To see this, fix a value of $\vec{\alpha}$, and recall that $r_s \stackrel{R}{\leftarrow} \{0,1\}^s$ is chosen by $\mathcal{F}^s_{\text{CRS}}$, and that the output of $P_j$ is uniquely determined by $r_t$, $\vec{\alpha}$, and $m$. Then, the probability over the choice of $r_s$ that there *exist* a $u$-bit value $m$ and a $t$-bit value $r_t$ such that the output of $P_j$ is $r_s$ is at most $2^{t+u-s}$. Note that the claim holds even if $\mathcal{Z}$ and $\mathcal{A}$ are restricted to polynomial time and even if $P_i$ and $\mathcal{S}$ are unbounded. □

## 4.3 Zero-knowledge in the CRS Model.

Finally, we describe the application of the JUC theorem to Zero-Knowledge protocols in the CRS model. The application is very similar to the case of commitment (Section 4.1). Let us first recall the zero-knowledge functionality, $\mathcal{F}_{zk}$, as defined in [Can01], (see Fig. 4).

As in the case of $\mathcal{F}_{\text{COM}}$, we only know how to realize functionality $\mathcal{F}_{zk}$ in the CRS model (i.e., in the $\mathcal{F}_{\text{CRS}}$-hybrid model). Also here, straightforward composition of a protocol $\pi$ in the $\mathcal{F}_{zk}$-hybrid model with a protocol $\rho$ that securely realizes $\mathcal{F}_{zk}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model would result in a composed protocol $\pi^\rho$ that is highly wasteful of the reference string. We solve the problem by using universal composition with joint state. That is, given a protocol $\hat{\rho}$ that securely realizes $\hat{\mathcal{F}}_{zk}$, the multi-session extension of $\mathcal{F}_{zk}$, we conclude that the composed protocol $\pi^{[\hat{\rho}]}$ runs in the $\mathcal{F}_{\text{CRS}}$-hybrid model and emulates $\pi$. Furthermore, if $\hat{\rho}$ uses only few copies of $\mathcal{F}_{\text{CRS}}$ then so does $\pi^{[\hat{\rho}]}$.

**Functionality $\mathcal{F}_{\text{ZK}}^{R}$**

$\mathcal{F}_{zk}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary $\mathcal{S}$, given a binary relation $R$.

1. Upon receipt of a value $(\texttt{prover}, sid, P_i, P_j, x, w)$ from some party $P_i$, Send $(sid, P_i, x, R(x, w))$ to $P_j$ and $\mathcal{S}$, and halt.

Figure 4: The Zero-Knowledge functionality, $\mathcal{F}_{zk}$

We complete the discussion by pointing to two protocols that securely realize $\hat{\mathcal{F}}_{zk}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model, and use only a single copy of $\mathcal{F}_{\text{CRS}}$. First, recall protocol *hc* in [CF01] that securely realizes $\mathcal{F}_{zk}$ in the $\mathcal{F}_{\text{COM}}$-hybrid model. (We remark that the formalization of $\mathcal{F}_{zk}$ in [CF01] is slightly different than the one here. Nonetheless, it is easy to see that the two formalizations are equivalent.) We claim that this protocol in effect realizes also $\hat{\mathcal{F}}_{zk}$. (Simply run the protocol separately for each interaction.) Thus, using the JUC Theorem, we obtain that the composed protocol, $hc^{[\mathsf{UCC}_{\mathsf{ReUse}}]}$, securely realizes $\hat{\mathcal{F}}_{zk}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model. That is, we have:

**Claim 4** *Protocol* $\text{hc}^{[\mathsf{UCC}_{\mathsf{ReUse}}]}$ *securely realizes* $\hat{\mathcal{F}}_{zk}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model. Furthermore, it uses only a single copy of* $\mathcal{F}_{\text{CRS}}$.

Next, we note that the simulation-sound non-interactive zero-knowledge proof of knowledge protocol of De-Santis et al. [DCO+01] can be written as a protocol that securely realizes $\hat{\mathcal{F}}_{zk}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid model, with respect to non-adaptive adversaries:

**Claim 5** *The protocol of* [DCO+01]*securely realizes* $\hat{\mathcal{F}}_{zk}$ *in the* $\mathcal{F}_{\text{CRS}}$*-hybrid model, with respect to non-adaptive adversaries. Furthermore, it uses only a single copy of* $\mathcal{F}_{\text{CRS}}$.

# 5 Application to Protocols Using Signature Schemes

We demonstrate the applicability of the JUC Theorem to the case of composing multiple protocol instances that use the same instance of a signature scheme. We start by recalling the ideal signature and certification functionality, $\mathcal{F}_{\text{CERT}}$, from [Can03]. (Recall that $\mathcal{F}_{\text{CERT}}$ can be realized in a natural way using standard signature schemes plus a certification authority; see more details below.) Next, we present $\hat{\mathcal{F}}_{\text{CERT}}$, the multi-session extension of $\mathcal{F}_{\text{CERT}}$, and demonstrate how to realize $\hat{\mathcal{F}}_{\text{CERT}}$ with essentially the same overhead as realizing a single copy of $\mathcal{F}_{\text{CERT}}$ per signer. Finally, we recall the applications to key-exchange and authenticated Byzantine agreement protocols.[1]

**The ideal signature and certification functionality, $\mathcal{F}_{\text{CERT}}$.** Functionality $\mathcal{F}_{\text{CERT}}$ is presented in Figure 5. $\mathcal{F}_{\text{CERT}}$ allows for ideal generation and verification of signatures for messages, while providing binding between the signatures and the identity of the signer. (The binding between a signature and the identity of the signer is what provides the "certification" part.) Each copy of $\mathcal{F}_{\text{CERT}}$ corresponds to a single instance of a "signing key"; the identity of the signer is incorporated in the SID of the copy. See more discussion on the formalization of $\mathcal{F}_{\text{CERT}}$ in [Can03]. It is also shown there how $\mathcal{F}_{\text{CERT}}$ can be realized given any signature scheme that is existentially unforgeable against chosen message attack, as defined in [GMRi88], plus a "certification authority" that registers party identities together with arbitrary public values provided by the registered party.

[1] Earlier versions of this report applied the JUC theorem directly to the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$. Here we take advantage of the recent formalization of [Can03], and apply the JUC theorem to protocols realizing the certification functionality, $\mathcal{F}_{\text{CERT}}$, that models a signature scheme together with a"certification authority" that binds verification keys to party identities. Using $\mathcal{F}_{\text{CERT}}$ instead of $\mathcal{F}_{\text{SIG}}$ has two advantages: First, it simplifies the presentation and analysis; Second, it provides a stronger result since it allows a single instance of a certification authority to handle multiple public keys.

We also remark that the formulation of $\mathcal{F}_{\text{SIG}}$ earlier versions of this report contained some inconsistencies. These points, and the corresponding corrections, are discussed in [Can03].

**Functionality $\mathcal{F}_{\text{CERT}}$**

$\mathcal{F}_{\text{CERT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary.

**Signature Generation:** Upon receiving a value $(\texttt{Sign}, sid, m)$ from $P_i$, first verify that $sid = (P_i, s)$ for some identifier $s$; if not, then ignore this input. (This verifies that $P_i$ is the authorized signer for this instance of $F_{\text{CERT}}$.) Then, send $(\texttt{Sign}, sid, m)$ to the adversary. Upon receiving $(\texttt{Signature}, sid, m, \sigma)$ from the adversary, send $(\texttt{Signature}, sid, m, \sigma)$ to $P_i$. Record the pair $(m, \sigma)$.

**Signature Verification:** Upon receiving a value $(\texttt{Verify}, sid, m, \sigma)$ from $P_j$, do:

1. If the pair $(m, \sigma)$ is recorded then set $f = 1$.
2. If the signer is not corrupted, and no pair $(m, \sigma')$ for any $\sigma'$ is recorded (i.e., $m$ was never before singed), then set $f = 0$.
3. In all other cases, let the adversary decide on the value of $f$. That is, hand $(\texttt{Verify}, sid, P_j, m, \sigma)$ to the adversary. Upon receiving $(\texttt{Verified}, sid, m, \phi)$ from the adversary let $f = \phi$.

Once the value of $f$ is set, send $(\texttt{Verified}, sid, m, f)$ to $P_j$.

Figure 5: The signature and certification functionality, $\mathcal{F}_{\text{CERT}}$.

**Functionality $\hat{\mathcal{F}}_{\text{CERT}}$**

$\hat{\mathcal{F}}_{\text{CERT}}$ proceeds as follows, running with parties $P_1, ..., P_n$ and an adversary.

**Signature Generation:** Upon receiving a value $(\texttt{Sign}, sid, ssid, m)$ from $P_i$, first verify that $ssid = (P_i, s)$ for some identifier $s$; if not, then ignore this input. (This verifies that $P_i$ is the authorized signer for this value of the sub-session identifier.) Then, send $(\texttt{Sign}, sid, ssid, m)$ to the adversary. Upon receiving $(\texttt{Signature}, sid, ssid, m, \sigma)$ from the adversary, send $(\texttt{Signature}, sid, ssid, m, \sigma)$ to $P_i$. Record the triple $(ssid, m, \sigma)$.

**Signature Verification:** Upon receiving a value $(\texttt{Verify}, sid, ssid, m, \sigma)$ from some $P_j$, do:

1. If the triple $(ssid, m, \sigma)$ is recorded then set $f = 1$.
2. If the signer is not corrupted, and no triple $(ssid, m, \sigma')$ for any $\sigma'$ is recorded (i.e., $m$ was never before singed for this $ssid$), then set $f = 0$.
3. In all other cases, let the adversary decide on the value of $f$. That is, hand $(\texttt{Verify}, sid, ssid, P_j, m, \sigma)$ to the adversary. Upon receiving $(\texttt{Verified}, sid, ssid, m, \phi)$ from the adversary let $f = \phi$.

Once the value of $f$ is set, send $(\texttt{Verified}, sid, ssid, m, f)$ to $P_j$.

Figure 6: $\hat{\mathcal{F}}_{\text{CERT}}$, the multi-session extension of functionality $\mathcal{F}_{\text{CERT}}$.

**The multi-session extension, $\hat{\mathcal{F}}_{\text{CERT}}$.** Functionality $\hat{\mathcal{F}}_{\text{CERT}}$ is obtained from functionality $\mathcal{F}_{\text{CERT}}$ via the transformation described in Section 3. For clarity, we present an explicit description of $\hat{\mathcal{F}}_{\text{CERT}}$ in Figure 6 below. We show how to realize functionality $\hat{\mathcal{F}}_{\text{CERT}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model. Specifically, we describe a simple protocol that allows each signer to maintain a single copy of the signing algorithm for all the signature generation requests made to $\hat{\mathcal{F}}_{\text{SIG}}$. The protocol, denoted SSI (for Sign the Session Identifier), is presented in Figure 7. The idea is to have the signer $P_i$ use a single signing key throughout the computation. Let $sid$ be the session ID of the current copy of SSI. When asked to sign a message $m$ with $ssid = (P_i, s)$, the signer signs the pair $(s, m)$, using the copy of $\mathcal{F}_{\text{CERT}}$ whose SID is $(P_i, sid)$. The verification algorithm verifies that the SSID in the verification request agrees with the signed identifier $s$ and the identity of the signer. Notice each party uses at most a single copy of $\mathcal{F}_{\text{CERT}}$ throughout the computation. Also, the protocol sends no messages, it only uses $\mathcal{F}_{\text{CERT}}$ locally.

**Claim 6** *Protocol* SSI *securely realizes $\hat{\mathcal{F}}_{\text{CERT}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model.*

**Proof:** Let $\mathcal{A}$ be an adversary that interacts with protocol SSI in the $\mathcal{F}_{\text{CERT}}$-hybrid model. We construct an adversary $\mathcal{S}$ so that no environment can tell whether it is interacting with $\mathcal{S}$ in the ideal process for $\hat{\mathcal{F}}_{\text{CERT}}$ or

**Protocol** SSI

Party $P_i$ proceeds as follows:

**Signature generation:** When activated with input $(\texttt{sign}, sid, ssid, m)$, where $ssid = (P_{i'}, s)$, $P_i$ proceeds as follows. If $i' \neq i$ then $P_i$ does nothing. If $i' = i$, then $P_i$ sends $(\texttt{sign}, (P_i, sid), (s, m))$ to $\mathcal{F}_{\text{CERT}}$. (That is, $P_i$ asks the copy of $\mathcal{F}_{\text{CERT}}$ with SID $(P_i, sid)$ to sign $(s, m)$.) When obtaining a signature $\sigma$ from $\mathcal{F}_{\text{CERT}}$, $P_i$ outputs $(\texttt{signature}, sid, ssid, m, \sigma)$.

**Signature verification:** When activated with input $(\texttt{verify}, sid, ssid, m, \sigma')$, where $ssid = (P_j, s')$, $P_i$ sends $(\texttt{verify}, (P_j, sid), (s', m), \sigma)$ to $\mathcal{F}_{\text{CERT}}$. Upon receiving $(\texttt{verified}, (P_j, sid), (s', m), f)$ from $\mathcal{F}_{\text{CERT}}$, $P_i$ outputs $(\texttt{verified}, sid, ssid, m, f)$.

Figure 7: The protocol for realizing $\hat{\mathcal{F}}_{\text{CERT}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model.

with $\mathcal{A}$ and SSI in the $\mathcal{F}_{\text{CERT}}$-hybrid model. Adversary $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$, and proceeds as follows. (We provide an event-driven description of $\mathcal{S}$. The "code" of $\mathcal{S}$ can be derived in a straightforward way.)

1. Throughout, whenever $\mathcal{S}$ receives an input value from $\mathcal{Z}$, it copies this value to $\mathcal{A}$'s input tape. Whenever $\mathcal{A}$ writes a value on its output tape, $\mathcal{S}$ copies this value to its own output tape (to be read by $\mathcal{Z}$).

2. $\mathcal{Z}$ **activates an uncorrupted** $P_i$ **with input** $(\texttt{sign}, sid, ssid, m)$ where $P_i$ is the legitimate signer and $ssid$ is a legitimate identifier (i.e., $ssid = (P_i, s)$). In this case, $\mathcal{S}$ receives $(\texttt{sign}, sid, ssid, m)$ from $\hat{\mathcal{F}}_{\text{CERT}}$. It then activates $\mathcal{A}$ to receive a value $(\texttt{sign}, (P_i, sid), (s, m))$ from $\mathcal{F}_{\text{CERT}}$. Upon receiving a signature value $\sigma$ from $\mathcal{A}$, $\mathcal{S}$ hands $\sigma$ to $\hat{\mathcal{F}}_{\text{CERT}}$.

3. **The simulated** $\mathcal{A}$ **sends** $(\texttt{sign}, sid, m)$ **to** $\mathcal{F}_{\text{CERT}}$ **from a corrupted** $P_i$**.** In this case, $\mathcal{S}$ decodes $sid = (P_{i'}, sid')$ and $m = (s, m')$, and sends $(\texttt{sign}, sid', (P_{i'}, s), m')$ from $P_i$ to $\hat{\mathcal{F}}_{\text{CERT}}$ in the ideal process. Next, upon receiving $(\texttt{sign}, sid', (P_{i'}, s), m')$ from $\hat{\mathcal{F}}_{\text{CERT}}$, $\mathcal{S}$ activates $\mathcal{A}$ to receive the value $(\texttt{sign}, sid, m)$ from $\mathcal{F}_{\text{CERT}}$. Upon receiving a signature value $\sigma$ from $\mathcal{A}$, $\mathcal{S}$ hands $\sigma$ to $\hat{\mathcal{F}}_{\text{CERT}}$.

4. $\mathcal{Z}$ **activates an uncorrupted** $P_i$ **with input** $(\texttt{verify}, sid, ssid, m, \sigma)$**.** If $\hat{\mathcal{F}}_{\text{CERT}}$ responds to this input without activating $\mathcal{S}$, then $\mathcal{S}$ of course need not do anything. If $\mathcal{S}$ receives $(\texttt{verify}, sid, ssid, m, \sigma)$ from $\hat{\mathcal{F}}_{\text{CERT}}$, where $ssid = (P_j, s)$, then it activates $\mathcal{A}$ to receive value $(\texttt{verify}, (P_j, sid), (s, m), \sigma')$ from $\mathcal{F}_{\text{CERT}}$, obtains an answer $\phi$ from $\mathcal{A}$, and responds to $\hat{\mathcal{F}}_{\text{CERT}}$ with $\phi$.

5. **The simulated** $\mathcal{A}$ **sends** $(\texttt{verify}, sid, m, \sigma)$ **to** $\mathcal{F}_{\text{CERT}}$ **from a corrupted** $P_i$**.** In this case, $\mathcal{S}$ decodes $sid = (P_{i'}, sid')$ and $m = (s, m')$, and sends $(\texttt{verify}, sid', (P_{i'}, s), m', \sigma)$ to $\hat{\mathcal{F}}_{\text{CERT}}$ from $P_i$. If $\mathcal{S}$ receives $(\texttt{verify}, sid', (P_{i'}, s), P_{i'}, m', \sigma)$ from $\hat{\mathcal{F}}_{\text{CERT}}$, then it activates $\mathcal{A}$ to receive value $(\texttt{verify}, sid, P_{i'}, m, \sigma)$ from $\mathcal{F}_{\text{CERT}}$, obtains an answer $\phi$ from $\mathcal{A}$, and responds to $\hat{\mathcal{F}}_{\text{CERT}}$ with $\phi$.

6. **The simulated** $\mathcal{A}$ **corrupts party** $P_i$**.** In this case, $\mathcal{A}$ expects to see the internal state of $P_i$ in protocol SSI, which consists of the all the past inputs and outputs of $P_i$ (the protocol stores no internal data). $\mathcal{S}$ then corrupts $P_i$ in the ideal process, obtains the inputs and outputs of $P_i$ in the ideal process, and forwards this information to $\mathcal{A}$.

Let $\mathcal{Z}$ be an environment machine. It is straightforward to verify that the view of the simulated $\mathcal{A}$ within $\mathcal{S}$ in the ideal process for $\hat{\mathcal{F}}_{\text{CERT}}$ and with environment $\mathcal{Z}$ is distributed identically to the view of $\mathcal{A}$ in an execution of SSI in the $\mathcal{F}_{\text{CERT}}$-hybrid model. Similarly, the views of $\mathcal{Z}$ from the two interactions are identical. Note that this holds even if $\mathcal{Z}$ and $\mathcal{A}$ are computationally unbounded. □

## 5.1 Application to key-exchange protocols

Key exchange protocols allow pairs of parties to agree on secret "session keys". The session keys are typically used by shared-key encryption and authentication protocols in order to obtain pairwise secure communication during a communication session between the parties. Typically, multiple pairwise key-exchange sessions take place concurrently, where a single party takes part in multiple such sessions. There are several basic

methods for authenticating the parties in a key-exchange protocol, e.g. pre-shared key authentication, third-party authentication, or authentication based on public-key infrastructure (either signatures or public-key encryption). In all methods it is usually the case that a single instance of the authentication mechanism is used in multiple key-exchange sessions.

Prior complexity-based analytical works on key-exchange (e.g., [BR93, BR95, BCK98, Sho99, CK01]) model key-exchange protocols as multi-party protocols where multiple pairwise sessions take place within the same instance of the protocol. This results in a relatively cumbersome presentation and analysis. Furthermore, the complexity of the analysis is inherited by any protocol that uses a key exchange protocol (such as a secure session protocol or a more complex application). In contrast, we would like to be able to present and analyze key-exchange protocols as protocols between two parties, where the entire protocol consists of exchanging a single key. We would then like to be able to use a composition theorem in order to demonstrate security for the case of multiple concurrent sessions.

The present work demonstrates how this can be done in the case of key-exchange protocols that use digital signatures for peer authentication. Specifically, we enable the following methodology:

1. Define an ideal key-exchange functionality, $\mathcal{F}_{\text{KE}}$, that handles a single agreement on a key.
2. Study protocols that realize $\mathcal{F}_{\text{KE}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model. (This is the main bulk of the analytical work.)
3. Use the universal composition theorem to deduce that key exchange protocols remain secure (and in particular suffice for constructing secure channels) even when multiple copies are running concurrently in an adversarially controlled way. Here each exchange of a key potentially uses its own separate copy of $\mathcal{F}_{\text{CERT}}$.
4. Use universal composition with joint state to replace all copies of $\mathcal{F}_{\text{CERT}}$ with a single instance of protocol SSI (that securely realizes $\hat{\mathcal{F}}_{\text{CERT}}$).

Similar methodologies apply also to secure-session protocols that use key-exchange protocols as sub-routines (i.e., they operate in the $\mathcal{F}_{\text{KE}}$-hybrid model). Indeed, the formalization and results proven here play a central role in the [CK02] formalization of key-exchange and secure-session protocols.

## 5.2 Application to authenticated Byzantine agreement and broadcast protocols

Byzantine agreement protocols (respectively, broadcast protocols) are multiparty protocols where $n$ parties agree on a common output which is the input of one of the parties (resp., the input of a special party, called the sender, or general). In the authenticated version of these protocols the parties have access to a common "public-key infrastructure". More specifically, it is assumed that each party has a secret signing key for a secure signature scheme, and all parties have the corresponding verification key. The notion was proposed in [PSL80, LSP82], who also constructed such protocols with strong resilience properties.

We would like to study composability properties of such protocols, in the realistic setting where multiple instances of an authenticated Byzantine agreement protocols use the same public-key infrastructure (i.e., the same instance of a signature scheme). Lindell, Lysyanskaya and Rabin [LLR02] show that parallel or concurrent composition of such protocols is impossible if the different instances of the protocol don't have unique identifiers. We thus restrict ourselves to the case where protocol instances have unique identifiers. Here we enable a methodology similar to the one for key-exchange protocols. (We present the methodology for the case of broadcast.)

1. Define an ideal broadcast functionality, $\mathcal{F}_{\text{BC}}$, that handles broadcasting a single message.
2. Study protocols that realize $\mathcal{F}_{\text{BC}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model.
3. Use the universal composition theorem to demonstrate that broadcast protocols (i.e., protocols that realize $\mathcal{F}_{\text{BC}}$) remain secure under concurrent composition. Here each instance of the broadcast protocol (i.e., each broadcasting of a message) uses its own separate copy of $\mathcal{F}_{\text{CERT}}$.
4. Use universal composition with joint state to replace all copies of $\mathcal{F}_{\text{CERT}}$ with a single instance of protocol SSI (that securely realizes $\hat{\mathcal{F}}_{\text{CERT}}$).

## References


[Bea91] D. Beaver. Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology*, 4:75–122, 1991.

[BCK98] M. Bellare, R. Canetti and H. Krawczyk, "A modular approach to the design and analysis of authentication and key-exchange protocols", *30th Symposium on Theory of Computing (STOC)*, ACM, 1998.

[BR95] M. Bellare and P. Rogaway. Provably Secure Session Key Distribution– the Three Party Case. In *Proc. 27th STOC*. ACM, 1995.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-cryptographic Fault-Tolerant Distributed Computations. In *Proc. 20th STOC*, pages 1–10. ACM, 1988.

[Blu82] M. Blum. Coin Flipping by Telephone . In *IEEE Spring COMPCOM*, pages 133–137, 1982.

[BR93] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *Crypto '93*, pages 232–249, 1993. LNCS No. 773.

[Can00] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proc. 42st FOCS*, pages 136–145. IEEE, 2001. http://eprint.iacr.org/2000/067.

[Can03] R. Canetti. On universally composable notions of signature, certification, and authentication. http://eprint.iacr.org/2003.

[CF01] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Crypto '01*, pages 19–40. LNCS No. 2139.

[CK01] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels . In *Eurocrypt '01*, pages 453–474, 2001. LNCS No. 2045.

[CK02] R. Canetti and H. Krawczyk. Universally Composable Key Exchange and Secure Channels . In *Eurocrypt '02*, pages 337–351, 2002. LNCS No. 2332.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *Proc. 34th STOC*, pages 494–503.

[CR03] R. Canetti and T. Rabin. Universal Composition with Joint State. Available online, http://eprint.iacr.org.

[DCO+01] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero-Knowledge. In *Crypto '01* LNCS No. 2139.

[DDN00] D. Dolev, C. Dwork, and M. Naor. Non-malleable Cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[DM00] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Crypto '00*, pages 74–92, 2000. LNCS No. 1880.

[DN02] I. Damgard and J. Nielsen. Universally Composable Commitment Schemes with Constant Expansion Factor. In *Crypto '02* LNCS No. 2442.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proc. 30th STOC*, pages 409–418.

[Gol01] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001. Preliminary version *http://philby.ucsd.edu/cryptolib.html/*.

[Gol02] O. Goldreich. Concurrent Zero-Knowledge With Timing Revisited. In *Proc. 34th STOC*.

[GK96] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM. J. Computing*, 25(1):169–192, 1996.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *Proc. 19th STOC*, pages 218–229. ACM, 1987.

[GO94] O. Goldreich and Y. Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *Journal of Cryptology*, 7(1):1–32, 1994. Preliminary version by Y. Oren in FOCS87.

[GL90] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Crypto '90*, 1990. LNCS No. 537.

[GMRa89] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Comput.,* Vol. 18, No. 1, 1989, pp. 186-208.

[GMRi88] S. Goldwasser, S. Micali, and R.L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Comput.*, April 1988, pages 281–308.

[LSP82] L. Lamport, R.E. Shostack, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. and Systems*, 4(3):382–401, 1982.

[LLR02] Y. Lindell, A. Lysyanskya, and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *Proc. 34th STOC*, pages 514–523.

[MR91] S. Micali and P. Rogaway. Secure Computation. In *Crypto '91*, pages 392–404, 1991. Manuscript available.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PSW00] B. Pfitzmann, M. Schunter, and M. Waidner. Secure Reactive Systems. IBM Research Report RZ 3206 (#93252), IBM Research, Zurich, May 2000.

[Sho99] V. Shoup. On Formal Models for Secure Key Exchange. Available at: http://www.shoup.org, 1999.

# A Universally Composable Security: A review

We provide a review of the UC security framework. The text is somewhat informal for clarity and brevity, and is mostly taken from the overview section of [Can01], with some local updates and modifications. Full details (as well as a history of works leading to that framework) appear there. We present the real-life model of computation, the ideal process, and the general definition of securely realizing an ideal functionality. Next we present the hybrid model and the composition theorem.

**Protocol syntax.** Following [GMRa89, Gol01], a protocol is represented as a system of interactive Turing machines (ITMs), where each ITM represents the program to be run within a different party. Specifically, the input and output tapes model inputs and outputs that are received from and given to other programs running on the same machine, and the communication tapes model messages sent to and received from the network. Adversarial entities are also modeled as ITMs. We concentrate on a model where the adversaries have an arbitrary additional input, or an "advice". From a complexity-theoretic point of view, this essentially implies that adversaries are non-uniform ITMs. We assume that all ITMs run in probabilistic polynomial time. (See more discussion and elaboration in this issue in [Can03].)

## A.1 The Basic Framework

Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in the presence of an adversary and in a given computational environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality", which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. We overview the model for protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The presentation below concentrates on the following model, aimed at representing current realistic communication networks (such as the Internet). The network is *asynchronous* without guaranteed delivery of messages. The communication is public and unauthenticated. (That is, the adversary may delete, modify, and generate messages at wish.) Parties may be broken into (i.e., become corrupted) throughout the computation, and once corrupted their behavior is arbitrary (or, *Byzantine*). Finally, all the involved entities are restricted to probabilistic polynomial time (or, "feasible") computation. The framework can be adapted in natural ways to present other models of computation (e.g., synchronous or authenticated communication). See more details in [Can01].

**Protocol execution in the real-life model.** We sketch the process of executing a given protocol $\pi$ (run by parties $P_1, ..., P_n$) with some adversary $\mathcal{A}$ and an environment machine $\mathcal{Z}$ with input $z$. All parties have a security parameter $k \in \mathbf{N}$ and are polynomial in $k$. The execution consists of a sequence of *activations,* where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some $P_i$) is activated. The environment is activated first. In each activation it may read the contents of the output tapes of all parties, and may write information on the input tape of either one of the parties or of the adversary. Once the activation of the environment is complete (i,e, once the environment enters a special waiting state), the entity whose input tape was written on is activated next.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or corrupt a party. There are no restrictions on the messages delivered. Upon corrupting a party, the adversary gains access to all the tapes of that party and controls all the party's future actions. Finally, the adversary may write arbitrary information on its output tape. In addition, whenever a party is corrupted the environment is notified (say, via a message that is added to the output tape of the adversary). If the adversary delivered a message to some uncorrupted party in its activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

Once a party is activated (either due to an input given by the environment or due to a message delivered by the adversary), it follows its code and possibly writes local outputs on its output tape and outgoing messages on its outgoing communication tape. Once the activation of the party is complete the environment is activated. The protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and parties running protocol $\pi$ on security parameter $k$, input $z$ and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \dots r_n$ as described above ($z$ and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{A}}$ for $\mathcal{A}$; $r_i$ for party $P_i$). Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

**The ideal process.** Security of protocols is defined via comparing the protocol execution in the real-life model to an *ideal process* for carrying out the task at hand. A key ingredient in the ideal process is the ideal functionality that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM that interacts with the environment and the adversary via a process described below. More specifically, the ideal process involves an ideal functionality $\mathcal{F}$, an ideal process adversary $\mathcal{S}$, an environment $\mathcal{Z}$ with input $z$, and a set of dummy parties $\tilde{P}_1, ..., \tilde{P}_n$.

As in the process of protocol execution in the real-life model, the environment is activated first. As there, in each activation it may read the contents of the output tapes of all (dummy) parties, and may write information on the input tape of either one of the (dummy) parties or of the adversary. Once the activation of the environment is complete the entity whose input tape was written on is activated next.

The dummy parties are fixed and simple ITMs: Whenever a dummy party is activated with input $x$, it forwards $x$ to the ideal functionality $\mathcal{F}$, say by writing $x$ on the incoming communication tape of $\mathcal{F}$. In this case $\mathcal{F}$ is activated next. Whenever a dummy party is activated due to delivery of some message it copies this message to its output. In this case $\mathcal{Z}$ is activated next.

Once $\mathcal{F}$ is activated, it reads the contents of its incoming communication tape, and potentially sends messages to the parties and to the adversary by writing these messages on its outgoing communication tape. Once the activation of $\mathcal{F}$ is complete, the entity that was last activated before $\mathcal{F}$ is activated again.

Once the adversary $\mathcal{S}$ is activated, it may read its own input tape and in addition it can read the *destinations* of the messages on the outgoing communication tape of $\mathcal{F}$. That is, $\mathcal{S}$ can see the identity of the recipient of each message sent by $\mathcal{F}$, but it cannot see the *contents* of this message (unless the recipient of the message is $\mathcal{S}$). $\mathcal{S}$ may either deliver a message from $\mathcal{F}$ to some party by having this message copied the party's incoming communication tape or corrupt a party. Upon corrupting a party, the adversary learns whatever is specified by the functionality.[2] In addition, from the time of corruption on, the adversary controls the party's actions. Also, both $\mathcal{Z}$ and $\mathcal{F}$ are notified that the party is corrupted. If the adversary delivered a message to some uncorrupted (dummy) party in an activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

As in the real-life model, the protocol execution ends when the environment completes an activation without writing on the input tape of any entity. The output of the protocol execution is the (one bit) output of $\mathcal{Z}$.

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ denote the output of environment $\mathcal{Z}$ after interacting in the ideal process with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$, on security parameter $k$, input $z$, and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{S}}, r_{\mathcal{F}}$ as described above ($z$ and $r_{\mathcal{Z}}$ for $\mathcal{Z}$, $r_{\mathcal{S}}$ for $\mathcal{S}$; $r_{\mathcal{F}}$ for $\mathcal{F}$). Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)$ denote the random variable describing $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})$ when $\vec{r}$ is uniformly chosen. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z)\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

**Securely realizing an ideal functionality.** We say that a protocol $\rho$ securely realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running $\rho$ in the real-life process, or it is interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process. This means that, from the point of view of the environment, running protocol $\rho$ is 'just as good' as interacting with an ideal process for $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.) A distribution ensemble is called binary if it consists of distributions over $\{0, 1\}$. We have:

**Definition 7** *Two* binary *distribution ensembles $X$ and $Y$ are* indistinguishable *(written $X \approx Y$) if for any $c \in \mathbf{N}$ there exists $k_0 \in \mathbf{N}$ such that for all $k > k_0$ and for all $a$ we have*

$$|\Pr(X(k, a) = 1) - \Pr(Y(k, a) = 1)| < k^{-c}.$$

**Definition 8** *Let $n \in \mathbf{N}$. Let $\mathcal{F}$ be an ideal functionality and let $\pi$ be an $n$-party protocol. We say that $\pi$* securely realizes *$\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have:*

$$\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}.$$

## A.2 The Composition Theorem

**The Hybrid Model.** In order to state the composition theorem, and in particular in order to formalize the notion of a real-life protocol with access to multiple copies of an ideal functionality, the hybrid model of

[2] Past formulations have specified that,upon corruption, the adversary learns all the past inputs and outputs of the party. The present, more general formulation allows capturing also properties such as forward secrecy, etc.

computation with access to an ideal functionality $\mathcal{F}$ (or, in short, the $\mathcal{F}$-hybrid model) is formulated. This model is identical to the real-life model, with the following additions. On top of sending messages to each other, the parties may send messages to and receive messages from an unbounded number of copies of $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, once a party sends a message $m$ to a copy of $\mathcal{F}$ with SID $s$, that copy is immediately activated to receive this message. (If no such copy of $\mathcal{F}$ exists then a new copy of $\mathcal{F}$ is created and immediately activated to receive $m$.) Furthermore, although the adversary in the hybrid model is responsible for delivering the messages from the copies of $\mathcal{F}$ to the parties, it does not have access to the contents of these messages.

Note that the hybrid model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol in the hybrid model. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

Let $\text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable describing the output of environment machine $\mathcal{Z}$ on input $z$, after interacting in the $\mathcal{F}$-hybrid model with protocol $\pi$, adversary $\mathcal{A}$, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$. (We stress that here $\pi$ is a hybrid of a real-life protocol with ideal evaluation calls to $\mathcal{F}$.) Let $\text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},Z}$ denote the distribution ensemble $\{\text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A},Z}\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

**Replacing a call to $\mathcal{F}$ with a protocol invocation.** Let $\pi$ be a protocol in the $\mathcal{F}$-hybrid model, and let $\rho$ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol $\pi^\rho$ is constructed by modifying the code of each ITM in $\pi$ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of $\rho$ with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of $\rho$, with the contents of that message given to $\rho$ as new input. Each output value generated by a copy of $\rho$ is treated as a message received from the corresponding copy of $\mathcal{F}$.

If protocol $\rho$ is a protocol in the real-life model then so is $\pi^\rho$. If $\rho$ is a protocol in some $\mathcal{G}$-hybrid model (i.e., $\rho$ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is $\pi^\rho$.

**Theorem statement.** In its general form, the composition theorem basically says that if $\rho$ securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model for some functionality $\mathcal{G}$, then an execution of the composed protocol $\pi^\rho$ "emulates" an execution of protocol $\pi$ in the $\mathcal{F}$-hybrid model. That is, for any adversary $\mathcal{H}$ there exists an adversary $\mathcal{H}'$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{H}$ and $\pi^\rho$ in the $\mathcal{G}$-hybrid model or it is interacting with $\mathcal{H}'$ and $\pi$ in the $\mathcal{F}$-hybrid model.

A corollary of the general theorem states that if $\pi$ securely realizes some functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model, and $\rho$ securely realizes $\mathcal{F}$ in the real-life model, then $\pi^\rho$ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model. (Here one has to define what it means to securely realize functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.) We first formalize the notion of protocol emulation:

**Definition 9 (Protocol emulation)** *Let $\mathcal{F}_1$, $\mathcal{F}_2$ be ideal functionalities and let $\pi_1$, $\pi_2$ be multiparty protocols. We say that $\pi_1$* in the $\mathcal{F}_1$-hybrid model emulates *$\pi_2$* in the $\mathcal{F}_2$-hybrid model *if for any adversary $\mathcal{A}_1$ there exists an adversary $\mathcal{A}_2$ such that for any environment machine $\mathcal{Z}$ we have*

$$\text{EXEC}^{\mathcal{F}_1}_{\pi_1,\mathcal{A}_1,\mathcal{Z}} \approx \text{EXEC}^{\mathcal{F}_2}_{\pi_2,\mathcal{A}_2,\mathcal{Z}}.$$

The case where $\pi_1$ (resp., $\pi_2$) runs in the real-life model of computation is captured by setting $\mathcal{F}_1$ (resp., $\mathcal{F}_2$) to be the functionality that does nothing. Note that emulation is transitive. That is, if $\pi_1$ in the $\mathcal{F}_1$-hybrid model emulates $\pi_2$ in the $\mathcal{F}_2$-hybrid model, and $\pi_2$ in the $\mathcal{F}_2$-hybrid model emulates $\pi_3$ in the $\mathcal{F}_3$-hybrid model, then $\pi_1$ in the $\mathcal{F}_1$-hybrid model emulates $\pi_3$ in the $\mathcal{F}_3$-hybrid model.

**Theorem 10 (Universal composition [Can01])** *Let $\mathcal{F}, G$ be ideal functionalities. Let $\pi$ be an $n$-party protocol in the $\mathcal{F}$-hybrid model, and let $\rho$ be an $n$-party protocol that securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid*

*model. Then protocol $\pi^\rho$ in the $\mathcal{G}$-hybrid model emulates protocol $\pi$ in the $\mathcal{F}$-hybrid model. In particular, if $\pi$ securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then $\pi^\rho$ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model.*