Universally Composable Signature, Certification, and Authentication[*]

Ran Canetti[†]

August 15, 2004

Abstract

Recently some efforts were made towards capturing the security requirements from digital signature schemes as an ideal functionality within a composable security framework. This modeling of digital signatures potentially has some significant analytical advantages (such as enabling component-wise analysis of complex systems that use signature schemes, as well as symbolic and automatable analysis of such systems). However, it turns out that formulating ideal functionalities that capture the properties expected from signature schemes in a way that is both sound and enjoys the above advantages is not a trivial task.

This work has several contributions. We first correct some flaws in the definition of the ideal signature functionality of Canetti, 2001, and subsequent formulations. Next we provide a minimal formalization of "ideal certification authorities" and show how authenticated communication can be obtained using ideal signatures and an ideal certification authority. This is done while guaranteeing full modularity (i.e., each component is analyzed as stand-alone), and in an unconditional and errorless way. This opens the door to symbolic and automated analysis of protocols for these tasks, in a way that is both modular and cryptographically sound.

[*] An extended abstract of this work appears in the proceedings of the 17th Computer Security Foundations Workshop (CSFW). This is a slightly corrected version. The first version of this paper, which dates November 18 2003, contained by mistake the date November 16, 2004 on its title page.

[†] IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

Contents

1 Introduction

Digital Signatures (first proposed by Diffie and Hellman [DH76]) are widely used in a variety of contexts. One main context is for binding between documents and "physical entities" such as human individuals or organizations. This is essential in electronic financial transactions and contracts. Another use is for guaranteeing authenticated communication over an unauthenticated medium. Here signature-based authenticated key exchange protocols plays a major role (see e.g. [ISO93, TLS, IPSEC]). Other uses include guaranteeing various integrity properties within cryptographic protocols (see e.g., [DDN00, CHK04]).

A widely accepted formalization of the security requirements from signature schemes was put forth in [GMRi88]. Essentially, the requirement is that, when the public and secret keys are honestly generated, then honestly generated signatures will always verify, and in addition it will be infeasible for an adversary to come up with a message m that was not honestly signed, and an alleged signature σ, such that σ will verify as a valid signature for m with respect to the given public key. This simple-to-state notion (called *existential unforgeability against chosen message attacks,* or CMA-security in short) has proven to be very useful, and in particular it seems appropriate in all the above contexts.

An alternative approach for capturing the security properties of signature schemes is via emulating an "ideal signature process" in an ideal-process-based general framework for analyzing security of protocols (such as [PW01, C01]). In this approach (taken by [C01, CK02, CR03, BPW03a, BH03]), one first formulates an "ideal signature functionality" that captures the desired security properties of signature schemes in an abstract way. A signature scheme is said to be secure if it "emulates" the ideal signature functionality. The effort to provide ideal-process-based definitions of security for signature schemes may seem surprising at first; indeed, ideal-process-based definitions of security have been traditionally used for capturing the security of *distributed protocols,* rather than for capturing the security of more basic cryptographic primitives such as digital signatures. The reason that this ideal-model based analysis of signature schemes is attractive is that it provides very strong *secure composability* properties. Specifically, in both frameworks mentioned above security was shown to be preserved under a very general composition operation called universal composition (see [C01], and [BPW04] based on [PW00, PW01]). These properties open the door to a number of potential advantages over the "traditional" notion of CMA-security. Let us highlight two main ones:

Enabling component-wise analysis of complex systems: When analyzing multi-protocol, multi-instance systems, we can separately analyze each protocol instance as if it is "stand alone," and then use the composition theorem to deduce the security of all instances when running concurrently in the same system. When applied to signature schemes, this means that we may be able to analyze each instance of a signature scheme separately, and then deduce that the composition of all the individual instances remains secure (i.e., maintains the functionality of multiple independent "signature services"). This would allow us to perform component-wise analysis also to complex protocols that *use* signature schemes. In fact, such separation can sometime be carried out even when multiple protocol instances use the *same* instance of a signature scheme [CR03].

Sound realization of the "Dolev-Yao methodology": Using the universal composition theorem, it is possible to use the following modular approach to analysis of protocols that use signatures: (a) Given a concrete, real-life protocol that uses a signature scheme, first de-compose the protocol into a "signature module" and a "high-level module" that uses the signature. (b) Prove that the "signature module" securely realizes the ideal signature functionality. (c) Prove that the "high-level module" has the desired security properties when having access to the ideal signature functionality.

(d) Use the composition theorem to assert that the original (concrete) protocol maintains the same security property as the "high-level module". The crux of this analytical approach is that step (c), namely the analysis of the "high-level module", can often be carried out unconditionally. Furthermore, it can potentially be carried out using *symbolic analysis* of protocols. Specifically, it may be possible to use the methodology proposed by Dolev and Yao [DY83], where encryption and signature are represented as abstract symbolic operations. This would demonstrate that the "Dolev-Yao methodology" can be used to argue security of the original protocol that uses concrete algorithms.

However, coming up with a sound, ideal-process-based notion of security for signature schemes, that realizes these potential advantages, turns out to be non-trivial. Two main approaches have been proposed. In this work we concentrate on the approach of [C01], which aims at capturing the basic security properties of a signature scheme as a stand-alone module, which is then used as a component withing larger protocols. The other approach, which was subsequently taken in [BPW03a], is discussed at the end of the Introduction. Here each instance of the "ideal signature functionality", denoted $\mathcal{F}_{\text{SIG}}$, corresponds to a single instance of a signature scheme (i.e., a single pair of signature and verification keys). Furthermore, it was claimed that a signature scheme is CMA-secure if and only if the scheme (or, rather, a simple protocol based on this scheme) securely realizes $\mathcal{F}_{\text{SIG}}$. This approach has a number of advantages, including the above mentioned points of sound realization of the Dolev-Yao paradigm and modular analysis of multi-protocol systems. However, the formalization in [C01], as well as the subsequent re-formulations of $\mathcal{F}_{\text{SIG}}$ in [CK02, CR03, BH03], contain a number of technical flaws that make the claim of equivalence to CMA security incorrect.

The present work has several contributions. First, we present a corrected (and somewhat simplified) formulation of the signature functionality. While the corrections are rather technical, they are necessary for providing a sound abstraction of signature schemes. In particular, they are necessary the equivalence with CMA-security of signature schemes. We identify three issues; they are discussed within. We invite the scrutiny of the community to verify the absence of flaws in the current formalization.

Next, we demonstrate the usefulness of $\mathcal{F}_{\text{SIG}}$ for realizing the tasks of certification of documents (i.e., the binding of documents to identities of principals), and obtaining authenticated communication. This is done in several steps, as follows. Recall that the signature functionality is aimed at capturing the basic properties of a CMA-secure signature scheme. This essentially means that the functionality provides binding between a message m and a public verification key v. (The binding is done via a "signature string", s.) We first define a somewhat "higher-level" functionality, that provides direct binding between messages and *the identity of the signer*. We call this "ideal certification functionality" $\mathcal{F}_{\text{CERT}}$. We formulate a minimal "set-up assumption" that allows realization of $\mathcal{F}_{\text{CERT}}$. Specifically, we assume existence of a "certification authority" with which parties have ideally authenticated communication, and whose sole role is to register party identities together with public values provided by the registered parties. We formulate this set-up assumption as an ideal functionality, called $\mathcal{F}_{\text{CA}}$, and show that $\mathcal{F}_{\text{CERT}}$ can be realized in a natural way, given ideal access to $\mathcal{F}_{\text{CA}}$ and to $\mathcal{F}_{\text{SIG}}$.

The next step is to show that ideally authenticated communication can be obtained given ideal access to $\mathcal{F}_{\text{CERT}}$. For this purpose we recall the ideal message authentication functionality, $\mathcal{F}_{\text{AUTH}}$, of [C01], and show a natural protocol that realizes F_{AUTH} given ideal access to $\mathcal{F}_{\text{CERT}}$. Finally, to complement this method of obtaining authenticated communication, and to justify the use of $\mathcal{F}_{\text{CA}}$, we show that $\mathcal{F}_{\text{AUTH}}$ (and consequently also $\mathcal{F}_{\text{CERT}}$) cannot be realized in the bare model of [C01] by any "useful" protocol.

Let us highlight two points. First, throughout the analysis we consider only protocols for a single instance of a signature scheme, and for authenticating a single message. Security for the multi-session case is obtained "automatically" via the UC, and the UC with joint state (JUC) theorems. (JUC is needed for demonstrating how multiple messages can be authenticated using a single instance of a signature scheme.) Second, both the protocol for realizing $\mathcal{F}_{\text{CERT}}$ given $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$, and the protocol for realizing $\mathcal{F}_{\text{AUTH}}$ given $\mathcal{F}_{\text{CERT}}$, are *unconditionally secure, with no error probability.* This fact greatly simplifies potential symbolic and automated analysis of these protocols using existing tools. (These points can be regarded as manifestations of the above-mentioned advantages of our formalization of signature schemes.)

Finally, we remark that the attempt to formulate an ideal process that captures the security properties of signature schemes in an abstract way brings up a number of interesting issues pertaining to the expected security properties of such schemes. Let us highlight four issues:

- Is it OK for a signature scheme to allow an adversary, given a public key v, a message m, and a signature s, to come up with a different signature $s' \neq s$ such that s' is a valid signature for m with respect to v?
- Is it OK for a signature scheme to allow an adversary, given a public key v, a message m, and a signature s, to come up with a different public key $v' \neq v$ such that s is a valid signature for m with respect to v'?
- Is it OK for a signature scheme to have the signature disclose information on messages that were signed in the past?
- Is it OK for a signature scheme to allow an adversarial signer to post a public key that causes the verification algorithm to accept any signature as valid for any message?
- When registering a new (public key, identity) pair with a certification authority, does the owner of the public key have to "prove possession of the private key" to the authority? If so, then what exactly has to be proven?

Future directions. The present formalization of $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{CERT}}$ is attractive in that it allows a very modular approach where each instance of the ideal functionality handles only a single instance of a signature scheme (i.e., a single pair of signature and verification keys). This has several advantages as described in this work. However, the interface of the signature scheme is somewhat less abstract than we may have wanted. Specifically, the interface contains an idealized "signature string" that is passed around among parties (see discussion in Section 2.1). An interesting research challenge is to show how the present interface can be rigorously linked to existing formal models and automated tools for symbolic protocol analysis. This would allow us to enjoy the advantages of fully modular analysis, together with the advantages of automation. One possible direction to go about this challenge is to prove "correspondence theorems" between states of protocols and adversaries in the UC framework, given ideal signatures, and their states in a more abstract formalism. The works of Abadi and Rogaway [AR00], and Micciancio and Warinschi [MW04] can be regarded as steps in this direction. Note, however, that in contrast to these works, here the correspondence theorems can potentially be *unconditional,* without involving computational issues at all. This may considerably simplify potential solutions.

More on related work. Several other abstractions of the security requirements from signature schemes were made in the past, e.g. [SM93, P95, CHH00, LMMS99]. However, none of these abstractions enjoy the secure composability property which is central to our goals. Signature schemes were also considered in [PSW00]; however, they were treated as part of a larger protocol and there was no attempt to capture signatures as an abstract primitive.

An alternative approach to writing an ideal functionality that captures signature schemes was proposed in [BPW03a]. This formulation provides a more abstract (and somewhat restricted) modeling of signatures. The additional abstraction comes at a price: It forces the [BPW03a] formalization to capture, within a single copy of the functionality, all the signature instances in the system, plus a number of other cryptographic services such as public-key encryption. Furthermore, all the communication among the parties (secure, authenticated, unauthenticated) must be handled by this single instance of the functionality. This "monolithic" approach loses much of the ability to carry out modular analysis of protocols. Furthermore, it means that sound Dolev-Yao style analysis as sketched above can be applied only to real-life protocols where the "low-level component" realizes their entire multi-instance, multi-function ideal functionality. This may be a limitation when attempting to analyze protocols that implement and use only, say, signatures but not, say, encryption. More discussion on this approach appears in Appendix B.

In a concurrent work to ours [BH03], Backes and Hofheinz point out some flaws in the [C01, CK02, CR03] formalization of $\mathcal{F}_{\text{SIG}}$, and propose a corrected version. While they point out some important flaws (e.g., the fact that an adversarial signer can provide inconsistent answers to different verification requests of the same message and signature), their formalization still contains some (other) flaws. See more details within.

Organization. The paper is organized as follows. Section 2 presents the corrected formulation of the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$, and re-proves the equivalence with CMA-security. Section 3 presents the ideal certification functionality, $\mathcal{F}_{\text{CERT}}$, and demonstrates how to realize it given $\mathcal{F}_{\text{SIG}}$ and a certification authority. Section 4 demonstrates how to realize authenticated communication given $\mathcal{F}_{\text{CERT}}$, and proves that obtaining authenticated communication in the bare unauthenticated model is impossible. Appendix A reviews the UC framework. Finally, Appendix B provides a more detailed review of the [BPW03a] formalization of ideal signatures.

2 The basic signature functionality

This section presents a corrected version of the ideal signature functionality of [C01, CK02, CR03, BH03]. We also use this opportunity to provide an extended motivation for the definitional approach, as well as the specific choices made. (Some of these considerations were already mentioned in these works; still, we hope that the additional elaboration provided here will prove useful.) Section 2.1 informally discusses the first considerations leading to the present formalization of the basic signature functionality, $\mathcal{F}_{\text{SIG}}$. Section 2.2 presents the updated formulation of $\mathcal{F}_{\text{SIG}}$, along with a number of additional considerations. Section 2.3 proves equivalence with CMA-security. The reader is first referred to Appendix A for an overview of the UC framework.

2.1 First attempts

When presented in the most abstract way, a signature scheme provides a way to bind messages to some publicly known entity, or a party. An immediate realization of this concept as an ideal functionality may proceed as follows: The ideal functionality (which may be thought of as a "trusted service" that is available to all parties) simply serves as a "depository of signed messages". That is, the functionality allows a single party, called the signer, to register messages as signed. Any party can then ask the functionality whether some message m is registered as signed. Let us call this ideal functionality $\mathcal{F}_1$.

Functionality $\mathcal{F}_1$ indeed captures a basic ideal concept of digital signatures. However, it is somewhat "too ideal", in a number of respects. One such respect is that $\mathcal{F}_1$ directly binds a message to an identity of a party (the signer). Realizing this requires some prior communication among all parties, including some global agreement or broadcast. Arguably, such communication need not be part of the basic definition of digital signatures. (Rather, it is part of a "certification process" that builds on top of digital signatures.) Another limitation of this direct binding of messages to parties is that it excludes other uses of signature schemes, such as binding to organizations, or other uses within cryptographic protocols. In addition, natural operations such as "certifying a public key" by signing it using a different key cannot be modeled in a modular way. That is, both the "certifying" and the "certified" public keys have to be part of the same copy of the functionality (which in addition has to explicitly accommodate such recursive operations).

A second attempt at formulating an ideal signature functionality may thus make the "public key" an explicit part of the interface of the ideal functionality. That is, now the ideal signature functionality behaves as $\mathcal{F}_1$ does, except that it incorporates an initial "key set-up" interface, where the signer obtains a "idealized verification key" from the functionality. The task of communicating the public key to other parties or entities is now left to the protocol that uses the ideal service represented by the functionality. (We postpone addressing the question of how the value of the key is determined to the next sub-section.) Verification queries now take the form of "is this message signed with this public key". Let us call this ideal functionality $\mathcal{F}_2$.

Functionality $\mathcal{F}_2$ better captures the basic properties of signature schemes, in that it leaves the binding between the public key and an external identity out of scope. However, $\mathcal{F}_2$ is still somewhat "over-idealized", in that it ideally binds a signed message to a verification key without the mediation of a "signature string". To see where this becomes problematic, consider a real-life situation where party A obtains a signature s on some message m from the signer, and then transfers (m, s) to another party B. If the signature string s is treated as an internal "implementation detail" and is not part of the functionality interface, then the "signature protocol" that realizes $\mathcal{F}_2$ must take care of transferring "the ability to verify that m is signed" (i.e., transferring s) from party A to party B. This means that the signature protocol has to be active whenever signatures are transferred from one party to another in any arbitrary communication. This of course does not correspond to our intuitive notion of a signature scheme. The lack of explicit signature strings also causes some other modeling problems. For instance, modeling natural operations such as sending an "encrypted signature" that is usable only by the holders of the decryption key cannot be done in a modular way, i.e., in a model where parties have access to an ideal signature functionality and a *separate* "ideal encryption functionality".

We conclude that in order to capture our intuitive notion of signature schemes, an ideal signature functionality should make the "signature string" part of its interface. That is, the signing process will generate an "idealized signature string", that will be presented at verification time. (Also here, we postpone addressing the question of how the signature string is determined to the next sub-section). The ideal verification process will use the message, the idealized signature string, and the public key. We demonstrate that this process can be implemented by a real-life "verification algorithm" that is local and does not require any extraneous communication. This indeed corresponds ot our intuitive notion of a signature verification process.

The above discussion captures only few of the considerations that come into play when formulating ideal signature functionalities. Other considerations include the wish to make sure that a single copy of the functionality will correspond to a single instance of a signature scheme. This is essential if one wants to use an ideal signature functionality within other cryptographic protocols and maintain the feature that, at all levels, each protocol instance can be analyzed separately from all others. A number of other considerations are discussed in subsequent sections.

2.2 The basic signature functionality, $\mathcal{F}_{\text{SIG}}$

This section presents and motivates the ideal signature functionality, $\mathcal{F}_{\text{SIG}}$. The basic idea is to have $\mathcal{F}_{\text{SIG}}$ provide a "registry service" where a distinguished party (the signer, S in the figure) can register (*message,signature*) pairs. Any party that provides the right verification key can check whether a given pair is registered. The functionality is presented in Figure 1. Below we highlight some aspects of the formulation and motivate the definitional choices made. We also discuss the differences from prior formulations.

Functionality $\mathcal{F}_{\text{SIG}}$

Key Generation: Upon receiving a value $(\texttt{KeyGen}, \mathit{sid})$ from some party S, verify that $\mathit{sid} = (S, \mathit{sid}')$ for some sid'. If not, then ignore the request. Else, hand $(\texttt{KeyGen}, \mathit{sid})$ to the adversary. Upon receiving $(\texttt{Verification Key}, \mathit{sid}, v)$ from the adversary, output $(\texttt{Verification Key}, \mathit{sid}, v)$ to S, and record the pair (S, v).

Signature Generation: Upon receiving a value $(\texttt{Sign}, \mathit{sid}, m)$ from S, verify that $\mathit{sid} = (S, \mathit{sid}')$ for some sid'. If not, then ignore the request. Else, send $(\texttt{Sign}, \mathit{sid}, m)$ to the adversary. Upon receiving $(\texttt{Signature}, \mathit{sid}, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{Signature}, \mathit{sid}, m, \sigma)$ to S, and record the entry $(m, \sigma, v, 1)$.

Signature Verification: Upon receiving a value $(\texttt{Verify}, \mathit{sid}, m, \sigma, v')$ from some party P, hand $(\texttt{Verify}, \mathit{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{Verified}, \mathit{sid}, m, \phi)$ from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. *(This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)*
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. *(This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)*
3. Else, if there is an entry (m, σ, v', f') recorded, then let $f = f'$. *(This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)*
4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ).

Output $(\texttt{Verified}, \mathit{id}, m, f)$ to P.

Figure 1: The basic signature functionality, $\mathcal{F}_{\text{SIG}}$.

Determining the values of the verification key and the signatures. Functionality $\mathcal{F}_{\text{SIG}}$ lets the adversary determine the values of the verification key and the legitimate signatures. This reflects the fact that the intuitive notion of basic security of signature schemes does not make any requirements on these values. In particular, the signature values may depend on the identity of the signer, on the signed message, or even on all the messages signed so far. There is also no requirement that signatures on different messages be different from each other. (Indeed, observe that the security properties guaranteed by $\mathcal{F}_{\text{SIG}}$ hold even if all signature strings are identical. Said otherwise, making sure that strings are different from each other is only a technical tool that allows realizing the abstract requirement in an algorithmic way.)

Alternative (and more restrictive) formulations would require that signatures depend only on the currently signed message, or have some pre-determined distribution. The first requirement was proposed in [BPW03b]. The second is reminiscent of verifiable random functions [MRV99]; however, whereas verifiable random functions do not guarantee that the signature value "appears random" to the signer, the above alternative formulation does.

Verification with an incorrect key. If the verification key v' presented by the verifier is not the registered one (i.e., v) then $\mathcal{F}_{\text{SIG}}$ provides no guarantees regarding the outcome of the verification process. This captures the fact that the basic functionality of a signature scheme only binds messages and signatures to verification keys, rather than to other entities such as party identities. It is the responsibility of the protocol that uses $\mathcal{F}_{\text{SIG}}$ to make sure that the verifying party has the correct verification key. (Jumping ahead, we remark that the binding of signatures to party identities is provided by the "certification functionality", $\mathcal{F}_{\text{CERT}}$, presented in Section 3.)

Allowing multiple signatures for a message. When the singer activates $\mathcal{F}_{\text{SIG}}$ multiple times for signing the same message, $\mathcal{F}_{\text{SIG}}$ allows the adversary to generate multiple different signatures for that message. This reflects the fact that the standard security requirement from signature schemes does not prohibit schemes that, in different activations, generate different signatures for the same message. A more restrictive variant would mandate that each message may have at most a single valid signature.

Allowing public modification of signatures. When presented with a verification request for (m, σ, v), where m was legitimately signed but with a different signature σ', $\mathcal{F}_{\text{SIG}}$ lets the adversary decide whether the verification should succeed (subject to the consistency condition). This reflects the fact that the basic notion of security of signature schemes makes no requirement on the verification procedure in such a case. In particular, it may be possible to publicly generate new signatures for a message from existing ones. An alternative (and more restrictive) formulation would force rejection whenever σ was not generated by the signing algorithm as a signature on m. (In our formalization this would mean that (m, σ, v) is rejected unless m is explicitly recorded as signed with σ.) This stronger requirement is considered in e.g. [GO92].

Allowing Corrupted signers to claim signatures. If the signer is corrupted, and $\mathcal{F}_{\text{SIG}}$ is asked to verify (m, σ, v), then $\mathcal{F}_{\text{SIG}}$ allows the ideal-process adversary to force the answer to be "1", even if m was never before singed and v is the correct verification key. This feature captures the fact that the basic security properties from signature schemes do not prevent a corrupted signer from causing the verification algorithm to accept any signature as valid for any message. Let us clarify this point via an example. Take any "secure" signature scheme s and modify it into a signature scheme s' that is identical to s except that a bit b is perpended to the verification key. If $b = 0$ then the verification procedure remains unchanged. But if $b = 1$

then the verification procedure always accepts its input signature as a valid signature for its input message. We claim that the scheme s' is still "secure". (In particular, if s is CMA-secure then so is s'.) Still, s' allows a a corrupted party P to register with a public key that begins with a "1". In this case, P can claim any signature for any message as "its own". (As artificial as the scheme s' may look, similar properties are shared by existing signature schemes. For instance when using any RSA-based signature scheme, a corrupted party can publish a verification key that specifies an RSA modulus $N = 1$. In this case, the verification algorithm would accept any message-signature pair.)[1]

Requiring consistency in signature verification. Functionality $\mathcal{F}_{\text{SIG}}$ guarantees that if some verification request for (m, σ, v') returned an answer $f \in \{0, 1\}$, then all future verification requests for the same (m, σ, v') will return the same answer. This is so, regardless of whether σ was legitimately generated by $\mathcal{F}_{\text{SIG}}$ or not, or whether the verification key v' in the verification request is the value generated by $\mathcal{F}_{\text{SIG}}$ or not. This feature is central in our intuitive notion of signatures: A party wants to be assured that if it successfully verified some triple (m, σ, v') then any other party that follows the protocol would successfully verify the same triple, regardless if this triple was legitimately generated or not. (This basic consistency property holds trivially in most known signature schemes, where the verification algorithm is deterministic and does not use external information other than its input. Still, in an abstract functional specification like $\mathcal{F}_{\text{SIG}}$, consistency of verification answers has to be required explicitly.)

Encoding the signer's identity in the SID. $\mathcal{F}_{\text{SIG}}$ ignores all key generation and signature generation requests, unless the identity of the signer agrees with the signer-identity that is encoded in the session ID (SID). (Recall that in the UC framework each instance of an ideal functionality has a unique SID, which is determined by the first call to this instance.) This convention makes sure that only a single party (presumably, the first party to call this instance) is able to generate a key and sign messages. Furthermore, it allows the protocol that realizes $\mathcal{F}_{\text{SIG}}$ to simply ignore all `KeyGen` and `Sign` requests unless the SID in the request agrees with the local party identity. Notice that this convention makes the signer identity public, thus essentially prohibiting anonymity in signatures. Indeed, if anonymity is desired then a different formulation is needed.

Addressing secrecy and anonymity concerns. The present formulation of $\mathcal{F}_{\text{SIG}}$ does not provide any secrecy or anonymity guarantees for the singer. In particular, the signer identity is made public via the SID, and the adversary is aware of each signed message and the value of each generated signature. While formulating an ideal functionality that provides secrecy and anonymity guarantees is out of scope of this work, we mention some possible techniques for obtaining such properties within the current general approach.

Hiding the signer identity can be obtained by replacing the signer's identity in the SID with a random string (which is chosen by the calling protocol within the signer). Preventing the adversary from learning the values of the signed messages and the signatures can be done as follows: When providing $\mathcal{F}_{\text{SIG}}$ with the verification key v, the adversary would give

[1] The following additional property of signature schemes was recently proposed [MS03]. Given a public verification key v, a message m and a signature s on m that was "honestly generated" using the signing key that corresponds to v, it should be infeasible to come up with a different verification key $v' \neq v$ such that s passes as a legitimate signature for m with respect to public key v'. Functionality $\mathcal{F}_{\text{SIG}}$ does not guarantee this property. Indeed, while this property may be convenient in some specific uses, it is arguably not a basic requirement from signature schemes in general protocol settings.

$\mathcal{F}_{\text{SIG}}$ a "signature generation algorithm" (namely an ITM G), and a "signature verification algorithm" (namely an ITM V). Now, in the signature generation process, instead of asking the adversary to generate each signature value σ, $\mathcal{F}_{\text{SIG}}$ would keep a running instance of G and generate each signature by running G on the signed message. Similarly, $\mathcal{F}_{\text{SIG}}$ would keep a running copy of machine V; when processing a verification request, $\mathcal{F}_{\text{SIG}}$ would interact with V instead of interacting with $\mathcal{A}$. (Here both G and V are PPT ITMs. That is, there is a fixed polynomial $p()$ associated with each of the machines, and in each activation the machine makes only up to $p(n)$ steps, where n is the length of its input.)

Additional secrecy requirements may be captured as follows. Requiring that each signature depends only on the currently signed message can be captured by requiring that G does not keep state between signatures. Requiring that the signature does not disclose information on the message (other than the ability to verify) can be captured by letting G sign a random "alias" r_m instead of signing m.

Differences from prior formulations of $\mathcal{F}_{\text{SIG}}$. The formulation here differs from prior formulations in three main respects. (I) Different formulations did not encode the signer identity in the SID. The formulation of [C01] allowed any party to be the signer, as long as it is the first one to call this copy of $\mathcal{F}_{\text{SIG}}$. This prevented secure realizations since the adversary could always "get ahead" of any good signer and register itself as the signer. (This issue was discovered independently in [BH03] and in an earlier version of this work. The solutions proposed there are different than here.) (II) The formulations in [C01, CK02, CR03] do not allow a corrupted signer to cause the verification algorithm to accept arbitrary signature as valid for arbitrary messages. (See the above motivating discussion on this property of $\mathcal{F}_{\text{SIG}}$.) (III) Neither of the previous formulations provided the consistency guarantee discussed above in a satisfactory way. The formulations in [C01, CK02, CR03] do not address this point at all. In [BH03] the issue is explicitly addressed for the first time. Still, the guarantee there is only partial in that it is made only when the verification key in the the verification request is the recorded one; this allows an adversarial signer to "repudiate" signatures by failing to properly register its verification key. The formulation in [C04] suffers from two flaws. First, it is too restrictive in that it requires consistency per (message,signature) pair, rather than per (message,signature,public-key) triple as here. Consequently, it unnecessarily rules out signature schemes where the same (message,signature) pair may be valid for one public key, and invalid for another public key. Second, it does not record a negative verification answer in case 2 (matching public key, uncorrupted signer, and message never before signed). This would allow signature protocols where a negative answer for a (message,signature) pair is later changed to a positive one (say, after the message is signed). I thank Ivan Damgard for pointing out the shortcomings of the [C04] formulation.

Another point of difference from previous formalizations has to do with the formalization of probabilistic polynomial time interactive Turing machines. Since this technical point is general to the framework, we describe it together with the review of the framework in Appendix A.

2.3 Equivalence with the [GMRi88] notion of security

Recall that it is claimed in [C01] that realizing $\mathcal{F}_{\text{SIG}}$ is equivalent to existential unforgeability against chosen message attacks as in [GMRi88]. However, the proof there relates only to non-adaptive adversaries (i.e., to the case where the identities of the corrupted parties are fixed in advance). Furthermore, it is flawed in that it overlooks the flaws in the [C01] formulation $\mathcal{F}_{\text{SIG}}$. This section presents a corrected and extended version of that proof, that addresses the above issues, and in addition addresses also adaptive adversaries. (The overall structure of the proof remains unchanged; the modifications are all local in nature.)

We first briefly restate the [GMRi88] notion, while making explicit some requirements that remain implicit there. A signature scheme is a triple of PPT algorithms $\Sigma = (gen, sig, ver)$, where sig may maintain local state between activations.

Definition 1 *A signature scheme $\Sigma = (gen, sig, ver)$ is called* EU-CMA *if the following properties hold for any negligible function $\nu()$, and all large enough values of the security parameter k.*

Completeness: *For any message m,* $\text{Prob}[(s,v) \leftarrow gen(1^k);\ \sigma \leftarrow sig(s,m);\ 0 \leftarrow ver(m,\sigma,v)] < \nu(k)$.

Consistency (Non-repudiation): *For any m, the probability that $gen(1^k)$ generates (s,v) and $ver(m,\sigma,v)$ generates two different outputs in two independent invocations is smaller than $\nu(k)$. (This requirement is implicit in [*GMRi88*]. Indeed, it holds trivially in their case, where ver is a local and deterministic algorithm.)*

Unforgeability: *For any PPT forger F,* $\text{Prob}[(s,v) \leftarrow gen(1^k);\ (m,\sigma) \leftarrow F^{sig(s,\cdot)}(v); 1 \leftarrow ver(m,\sigma,v)$ *and F never asked sig to sign* $m] < \nu(k)$.

Next, we describe how to translate a signature scheme $\Sigma = (gen, sig, ver)$ into a protocol π_Σ in the present setting. This is done as follows: When party S, running π_Σ, receives an input $(\texttt{KeyGen,}sid)$, it verifies that $sid = (S, sid')$ for some sid'. If not, it ignores the input. Also, it runs algorithm gen, keeps the signing key s and outputs the verification key v. When S receives an input $(\texttt{Sign,}sid, m)$ for an sid for which it has a signing key s, it sets $\sigma = sig(s,m)$ and outputs $(\texttt{Signature,}sid, m, \sigma)$. When any party gets an input $(\texttt{Verify,}sid, m, \sigma, v')$, it outputs $(\texttt{Verified,}sid, m, ver(m,\sigma,v'))$. We show:

Theorem 2 *Let $\Sigma = (gen, sig, ver)$ be a signature scheme. Then π_Σ securely realizes $\mathcal{F}_{\text{SIG}}$ if and only if Σ is* EU-CMA.

Proof: For the "only if" direction, assume that Σ violates Definition 1. We show that π_Σ does not securely realize F_{SIG}. This is done by constructing an environment Z and a real-life adversary $\mathcal{A}$ such that for any ideal-process adversary $\mathcal{S}$, $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π_Σ or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SIG}}$. In all cases, $\mathcal{Z}$ corrupts no party and sends not messages to $\mathcal{A}$. (Since protocol π_Σ never sends any messages, this means that $\mathcal{A}$ is never activated.) In addition:

(I) Assume Σ is not complete, i.e. there exists m such that $\text{Prob}[(s,v) \leftarrow gen(1^k);\ \sigma \leftarrow sig(s,m);$ $0 \leftarrow ver(m,\sigma,v)] < \nu(k)$ for infinitely many k's. Then $\mathcal{Z}$ simply sets $sid = (S, 0)$ and activates some party S with input $(\texttt{KeyGen}, sid)$, followed by $(\texttt{Sign}, sid, m)$, obtains v and σ, and then activates some party V with $(\texttt{Verify}, sid, m, \sigma, v)$ and outputs the returned verification value. Clearly in the ideal process $\mathcal{Z}$ outputs 1 always, whereas in the interaction with π_Σ $\mathcal{Z}$ outputs 0 with non-negligible probability.

(II) Assume Σ is not consistent. Then $\mathcal{Z}$ operates similarly except that it activates V twice with $(\texttt{Verify}, sid, m, \sigma, v)$, and outputs 1 iff the two answers are the same. Again, in the ideal process $\mathcal{Z}$ outputs 1 always, whereas in the interaction with π_Σ $\mathcal{Z}$ outputs 0 with non-negligible probability.

(III) Assume Σ is not unforgeable, i.e. there exists a successful forger G for Σ. Then $\mathcal{Z}$ proceeds as above, except that it internally runs a copy of G and hands it the public key v obtained from S. From now on, whenever G asks its oracle to sign a message m, $\mathcal{Z}$ activates S with input $(\texttt{Sign,}\mathit{sid} = (S, 0), m)$, and reports the output to G. When G generates a pair (m, σ), $\mathcal{Z}$ Proceeds as follows. If m was signed before then $\mathcal{Z}$ outputs 0 and halts. Else, $\mathcal{Z}$ activates some party with input $(\texttt{Verify,}\mathit{sid}, m, \sigma, v)$ and outputs the verification result. It can be readily seen that, when $\mathcal{Z}$ interacts with π_Σ, G sees exactly a standard chosen message attack on Σ, thus $\mathcal{Z}$ outputs 1 with non-negligible probability. However, if $\mathcal{A}$ is interacting with the ideal process for $\mathcal{F}_{\text{SIG}}$, then $\mathcal{Z}$ never outputs 1.

For the "if" direction, assume that π_Σ does not realize $\mathcal{F}_{\text{SIG}}$, i.e. there is a real-life adversary $\mathcal{A}$ such that for any ideal-process adversary $\mathcal{S}$ there exists an environment $\mathcal{Z}$ that can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$ in the ideal process, or with π_Σ and $\mathcal{A}$ in the real-life model. We show that Σ violates Definition 1.[2] Since $\mathcal{Z}$ succeeds for any $\mathcal{S}$, it also succeeds for the following "generic" $\mathcal{S}$. $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$; then:

1. Any input from $\mathcal{Z}$ is forwarded to $\mathcal{A}$. Any outputs of $\mathcal{A}$ is copied to $\mathcal{S}$'s output (to be read by $\mathcal{Z}$).

2. Whenever $\mathcal{S}$ receives a message $(\texttt{KeyGen,}\mathit{sid}, S)$ from $\mathcal{F}_{\text{SIG}}$, it does: If sid is not of the form (S, sid') then $\mathcal{S}$ ignores this request. Otherwise, $\mathcal{S}$ runs $(s, v) \leftarrow \mathit{gen}(1^k)$, records s, and returns $(\texttt{Verification Key,}\mathit{sid}, v)$ to $\mathcal{F}_{\text{SIG}}$.

3. Whenever $\mathcal{S}$ receives a message $(\texttt{Sign,}\mathit{sid}, S, m)$ from $\mathcal{F}_{\text{SIG}}$, if $\mathit{sid} = (S, \mathit{sid}')$ and there is a recorded signing key s, then $\mathcal{S}$ computes $\sigma = \mathit{sig}(s, m)$, and hands $(\texttt{Signature,}\mathit{sid}, m, \sigma)$ back to $\mathcal{F}_{\text{SIG}}$. Otherwise, it does nothing.

4. Whenever $\mathcal{S}$ receives $(\texttt{Verify}, \mathit{sid}, m, \sigma, v)$ from $\mathcal{F}_{\text{SIG}}$, it returns $(\texttt{Verified}, \mathit{sid}, m, \phi)$ where $\phi = \mathit{ver}(v, m, \sigma)$.

5. When $\mathcal{A}$ corrupts some party P, $\mathcal{S}$ corrupts P in the ideal process. If P is the signer, then $\mathcal{S}$ reveals the signing key s (and potentially the internal state of algorithm sig, if such state exists) as the internal state of P.

Now, assume that scheme Σ is both complete and consistent (otherwise the theorem is proven). We demonstrate that it is not unforgeable, by constructing a forger G. This is done as follows. G runs a simulated copy of $\mathcal{Z}$, and simulates for $\mathcal{Z}$ an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SIG}}$ (where G plays the role of both $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ for $\mathcal{Z}$). Like $\mathcal{S}$, G runs a simulated copy of $\mathcal{A}$. However, in the first activation, instead of running gen to obtain the keys (s, v), G hands $\mathcal{A}$ the public verification key v in G's input. Instead of running the signing algorithm to obtain $\sigma = \mathit{sig}(s, m)$, G asks its oracle to sign m and obtains the signature σ. Whenever the simulated $\mathcal{Z}$ activates some uncorrupted party with input $(\texttt{Verify,}\mathit{sid}, m, \sigma, v)$, G checks whether (m, σ) constitute a forgery (i.e., whether m was never signed before and $\mathit{ver}(v, m, \sigma) = 1$). If (m, σ) is a forgery, then G outputs that pair and halts. Else it continues the simulation. If $\mathcal{A}$ asks to corrupt the signer than G halts with a failure output.

We analyze the success probability of G. Let B denote the event that, in a run of π_Σ with $\mathcal{A}$ and $\mathcal{Z}$ with $\mathit{sid} = (S, \mathit{sid}')$, the singer S generates a public key v, and some party is activated

[2] As demonstrated in [c01], it suffices to show this derivation for the case where $\mathcal{A}$ is the "dummy adversary" that only delivers messages, corrupts parties, and reports the gathered information to $\mathcal{Z}$. Still, for clarity we provide a proof for all $\mathcal{A}$.

with a verification request $(\texttt{Verify}, sid, m, \sigma, v)$, where $ver(m, \sigma, v) = 1$, and S is uncorrupted and never signed m. Since Σ is complete and consistent, we have that as long as event B does not occur, $\mathcal{Z}$'s view of an interaction with $\mathcal{A}$ and π_Σ is statistically close to its view of an interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ in the ideal process. (The views may differ in case of a completeness or consistency error. But these happen only with negligible probability.) However, we are guaranteed that $\mathcal{Z}$ distinguishes with non-negligible probability between the interaction with π_Σ and the interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{SIG}}$ Thus we are guaranteed that, when $\mathcal{Z}$ interacts with $\mathcal{A}$ and π_Σ, event B occurs with non-negligible probability.

It remains to observe that, from the point of view of $\mathcal{A}$ and $\mathcal{Z}$, the interaction with the forger G looks the same as an interaction in the real-life model with π_Σ. Thus, we are guaranteed that event B will occur with non-negligible probability. Notice that event B can occur only *before* the signer S is corrupted. This means that whenever event B occurs, G outputs a successful forgery. □

We remark that the adversary $\mathcal{A}$ constructed in the proof of the "only if" direction is non-adaptive, whereas the adversary $\mathcal{A}$ considered in the proof of the "if" direction can be adaptive. This means that for any signature scheme Σ, protocol π_Σ securely realizes $\mathcal{F}_{\text{SIG}}$ against adaptive adversaries if and only if it securely realizes $\mathcal{F}_{\text{SIG}}$ against non-adaptive adversaries. Also, it is interesting to note that the proof of Claim 2 does not involve any data erasures.

3 Using signatures to provide certification

This section presents an abstraction of signature schemes that directly binds messages and signatures to "physical entities," such as parties in a network. We use the term certification to describe such binding. We first formulate an ideal functionality, $\mathcal{F}_{\text{CERT}}$, that provides ideal binding of messages to party identities. Next, we demonstrate how to realize $\mathcal{F}_{\text{CERT}}$ given ideal access to $\mathcal{F}_{\text{SIG}}$. However, even given $\mathcal{F}_{\text{SIG}}$, it is impossible to realize $\mathcal{F}_{\text{CERT}}$ in a completely unauthenticated communication model, such as the bare model provided by the UC framework. (We prove this fact in the next section.) Therefore, we make the minimal set-up assumption that the parties have access to a rudimentary "certification authority" that registers party identities together with public values provided by the registered party. We formalize this set-up assumption via an ideal functionality, $\mathcal{F}_{\text{CA}}$, and show a natural protocol for realizing $\mathcal{F}_{\text{CERT}}$ given ideal access to $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{CA}}$. We note that this protocol is *errorless,* and *unconditionally secure.* That is, it realizes $\mathcal{F}_{\text{CERT}}$ perfectly, and even for unbounded adversary and environment.

Section 3.1 presents the ideal certification functionality, $\mathcal{F}_{\text{CERT}}$. Section 3.2 formulates $\mathcal{F}_{\text{CA}}$ and shows how $\mathcal{F}_{\text{CERT}}$ can be realized given $\mathcal{F}_{\text{SIG}}$ and $\mathcal{F}_{\text{CA}}$.

3.1 The certification functionality, $\mathcal{F}_{\text{CERT}}$

The ideal certification functionality, $\mathcal{F}_{\text{CERT}}$, is presented in Figure 2. It is similar to $\mathcal{F}_{\text{SIG}}$, except that it provides direct binding between a signature for a message and the identity of the signer. (In contrast, $\mathcal{F}_{\text{SIG}}$ provides binding only to the verification key.) Using common terminology, this corresponds to providing signatures accompanied by "certificates" that bind the verification process to the signer's identity. Consequently, in $\mathcal{F}_{\text{CERT}}$ the generation of public keys becomes an "implementation detail" and is not part of the interface with the environment. Furthermore, there are no public keys in the interface of F_{CERT}. We note that $\mathcal{F}_{\text{CERT}}$ does not deal with revocation of certificates. Dealing with revocation requires more structure (such as trusted "revocation authorities") and is left out of scope.

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

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, send $(\texttt{Sign}, sid, m)$ to the adversary. Upon receiving $(\texttt{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{Signature}, sid, m, \sigma)$ to S, and record the entry $(m, \sigma, 1)$.

Signature Verification: Upon receiving a value $(\texttt{Verify}, sid, m, \sigma)$ from some party P, hand $(\texttt{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\texttt{Verified}, sid, m, \phi)$ from the adversary, do:

1. If $(m, \sigma, 1)$ is recorded then set $f = 1$.
2. Else, if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.
3. Else, if there is an entry (m, σ, f') recorded, then set $f = f'$.
4. Else, set $f = \phi$, and record the entry (m, σ', ϕ).

Output $(\texttt{Verified}, sid, m, f)$ to P.

Figure 2: The certification functionality, $\mathcal{F}_{\text{CERT}}$.

3.2 Functionality $\mathcal{F}_{\text{CA}}$ and realizing $\mathcal{F}_{\text{CERT}}$

We present a simple protocol that realizes $\mathcal{F}_{\text{CERT}}$ given $\mathcal{F}_{\text{SIG}}$, with the aid of ideally authenticated communication with a "trusted certification authority." This set-up assumption is formalized as an ideal functionality, $\mathcal{F}_{\text{CA}}$. We start by presenting $\mathcal{F}_{\text{CA}}$. Next we present the protocol and prove its security. (There exist of course other methods for realizing $\mathcal{F}_{\text{CERT}}$, for instance using direct out-of-band exchange of public verification keys. Still, as shown in Claim 5, $\mathcal{F}_{\text{CERT}}$ cannot be realized in the bare unauthenticated model; *some* set-up assumption is necessary.)

The certificate authority functionality. The ideal certificate authority functionality, $\mathcal{F}_{\text{CA}}$, is presented in Figure 3. Also here, each copy of $\mathcal{F}_{\text{CA}}$ is bound to a single party identity via the session ID. In fact, for ease of presentation here we make the SID *identical* to the corresponding party identity. $\mathcal{F}_{\text{CA}}$ accepts only the first registered value, and does not allow for modification or "revocation." Such more advanced features are of course useful, but are not necessary for our basic use. We stress that F_{CA} does not perform any checks on the registered value; it simply acts as a public bulletin board. (In particular, no "proof of possession of secret key" is required.) Consequently, when running in the $\mathcal{F}_{\text{CA}}$-hybrid model, a party can register with some copy of $\mathcal{F}_{\text{CA}}$ using, e.g., the same public value as that of some other party, in another copy of $\mathcal{F}_{\text{CA}}$. Still, as seen below, the present minimal formulation suffices for realizing $\mathcal{F}_{\text{CERT}}$ and subsequently $\mathcal{F}_{\text{AUTH}}$.

Realizing $\mathcal{F}_{\text{CERT}}$ in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model. We present a protocol, CAS (for "certificate-authority-assisted signatures"), that realizes $\mathcal{F}_{\text{CERT}}$ in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model in a straightforward way. See Figure 4.

Functionality $\mathcal{F}_{\text{CA}}$

1. Upon receiving the first message $\texttt{(Register}, sid, v\texttt{)}$ from party P, send $\texttt{(Registered}, sid, v\texttt{)}$ to the adversary; upon receiving $\texttt{ok}$ from the adversary, and if $sid = P$ and this is the first request from P, then record the pair (P, v).
2. Upon receiving a message $\texttt{(Retrieve}, sid\texttt{)}$ from party P', send $\texttt{(Retrieve}, sid, P'\texttt{)}$ to the adversary, and wait for an $\texttt{ok}$ from the adversary. Then, if there is a recorded pair (sid, v) output $\texttt{(Retrieve}, sid, v\texttt{)}$ to P'. Else output $\texttt{(Retrieve}, sid, \perp\texttt{)}$ to P'.

Figure 3: The ideal certification authority functionality, $\mathcal{F}_{\text{CA}}$

Protocol CAS

Signing protocol: When activated with input $\texttt{(Sign}, sid, m\texttt{)}$, party P does:

1. P verifies that $sid = (P, s)$ for some identifier s; if not, then the input is ignored. (That is, P verifies that it is the legitimate signer for this sid.)
2. If this is the first activation then P generates a verification key, i.e., it sends $\texttt{(KeyGen}, sid\texttt{)}$ to $\mathcal{F}_{\text{SIG}}$. Once it obtains $\texttt{(Verification Key}, sid, v\texttt{)}$, it sends $\texttt{(Register}, P, v\texttt{)}$ to $\mathcal{F}_{\text{CA}}$.
3. P sends $\texttt{(Sign}, sid, m\texttt{)}$ to $\mathcal{F}_{\text{SIG}}$. Upon receiving $\texttt{(Signature}, sid, m, \sigma\texttt{)}$ from $\mathcal{F}_{\text{SIG}}$, P outputs $\texttt{(Signature}, sid, m, \sigma\texttt{)}$.

Verification protocol: When activated with input $\texttt{(Verify}, sid, m, \sigma\texttt{)}$, where $sid = (P, s)$, party P' checks whether it has a pair (P, v) recorded. If not, then P' sends $\texttt{(Retrieve}, P\texttt{)}$ to $\mathcal{F}_{\text{CA}}$, and obtains a response $\texttt{(Retrieve}, P, v\texttt{)}$. If $v = \perp$ then P' rejects the signature, i.e. it outputs (Verified, $sid, m, 0$). Else it records (P, v). Next, P' sends $\texttt{(Verify}, sid, m, \sigma, v\texttt{)}$ to $\mathcal{F}_{\text{SIG}}$, and outputs the response $\texttt{(Verified}, sid, m, f\texttt{)}$ from $\mathcal{F}_{\text{SIG}}$.

Figure 4: The protocol for realizing $\mathcal{F}_{\text{CERT}}$ in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model.

Claim 3 *Protocol* CAS *securely realizes functionality $\mathcal{F}_{\text{CERT}}$ in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model.*

Proof: Let $\mathcal{A}$ be an adversary that interacts with parties running CAS in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model. We construct an ideal-process adversary (simulator) $\mathcal{S}$ such that the view of any environment $\mathcal{Z}$ of an interaction with $\mathcal{A}$ and CAS is distributed identically to its view of an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{CERT}}$. As usual, simulator $\mathcal{S}$ runs an internal copy of $\mathcal{A}$ and each of the involved parties. All messages from $\mathcal{Z}$ to $\mathcal{A}$ and back are forwarded. In addition, $\mathcal{S}$ does:

Simulating signature generation. When $\mathcal{S}$ receives in the ideal process a message $\texttt{(Sign}, sid, m\texttt{)}$ from $\mathcal{F}_{\text{CERT}}$, where $sid = (P, s)$ and P is uncorrupted, it proceeds as follows:

1. If this is the first time that P generates a signature, then simulate for $\mathcal{A}$ the process of key generation. That is, send to $\mathcal{A}$ (in the name of $\mathcal{F}_{\text{SIG}}$) the message $\texttt{(KeyGen}, sid\texttt{)}$, obtain the response $\texttt{(Verification Key}, sid, v\texttt{)}$ from $\mathcal{A}$, and send to $\mathcal{A}$ the message $\texttt{(Registered}, P, v\texttt{)}$ from $\mathcal{F}_{\text{CA}}$. When $\mathcal{A}$ sends $\texttt{ok}$ to $\mathcal{F}_{\text{CA}}$, mark the pair (P, v) as recorded.
2. Simulate for $\mathcal{A}$ the process of signing m. That is, send to $\mathcal{A}$ (in the name of $\mathcal{F}_{\text{SIG}}$) the message $\texttt{(Sign}, sid, m\texttt{)}$, forward the response $\texttt{(Signature}, sid, m, \sigma\texttt{)}$ to $\mathcal{F}_{\text{CERT}}$.

Simulating the interaction of a corrupted signer proceeds as follows.

1. If $\mathcal{A}$ instructs a corrupted P to send $\texttt{(KeyGen}, sid\texttt{)}$ to $\mathcal{F}_{\text{CERT}}$ then proceed in the natural way. That is, if sid is different than the sid of the first message received from $\mathcal{F}_{\text{CERT}}$ in this run, or $sid \neq (P, s)$ for some s, then ignore this instruction. Else, send to $\mathcal{A}$ (in the name of $\mathcal{F}_{\text{SIG}}$) the message $\texttt{(KeyGen}, sid\texttt{)}$; when $\mathcal{A}$ responds with $\texttt{(Verification Key}, sid, v\texttt{)}$, send $\texttt{(Verification Key}, sid, v\texttt{)}$ to P.
2. The process of generating a signature by $\mathcal{F}_{\text{SIG}}$ is simulated in a similar way, with the addition that $\mathcal{S}$ records the generated (*message,signature*) pairs. Finally, When P sends (P, v') to $\mathcal{F}_{\text{CA}}$, send to $\mathcal{A}$ the message $\texttt{(Registered}, P, v'\texttt{)}$ from $\mathcal{F}_{\text{CA}}$. When $\mathcal{A}$ sends ok to $\mathcal{F}_{\text{CA}}$, then mark the pair (P, v') as recorded. (Note that v' may be different than v; still the simulation remains valid.)

Simulating signature verification. When notified by $\mathcal{F}_{\text{CERT}}$ that some uncorrupted party P' made a verification request, proceed as follows.

1. If this is the first verification request made by P', then simulate for $\mathcal{A}$ the exchange between P' and $\mathcal{F}_{\text{CA}}$. That is, simulate for $\mathcal{A}$ a message $\texttt{(Retrieve}, P, P'\texttt{)}$ coming from $\mathcal{F}_{\text{CA}}$. Then, when $\mathcal{A}$ responds with ok, simulate $\mathcal{F}_{\text{CA}}$'s output to P': If $\mathcal{F}_{\text{CA}}$ has a pair (P, v) recorded then record that P' obtained a response (P, v) from $\mathcal{F}_{\text{CA}}$. record that P' obtained a response $(P, \perp)$ from $\mathcal{F}_{\text{CA}}$.
2. If a message $\texttt{(Verify}, sid, P', m, \sigma\texttt{)}$ arrives from $\mathcal{F}_{\text{CERT}}$, then forward this message to $\mathcal{A}$ (in the name of $\mathcal{F}_{\text{SIG}}$). Forward $\mathcal{A}$'s response back to $\mathcal{F}_{\text{CERT}}$.

If the verifier P' is corrupted then the simulation is modified in the natural way. That is, when P' sends a message $\texttt{(Verify}, sid, m, \sigma, v\texttt{)}$ to to $\mathcal{F}_{\text{SIG}}$, generate a response following the instructions of $\mathcal{F}_{\text{SIG}}$.

Simulating party corruptions. When $\mathcal{A}$ corrupts a party, $\mathcal{S}$ corrupts that party in the ideal process, and forwards the obtained information to $\mathcal{A}$. This poses no problem since none of the parties maintains any secret information.

It is straightforward to verify that the simulation is perfect. That is, for any (even computationally unbounded) environment $\mathcal{Z}$ and $\mathcal{A}$, it holds that $\mathcal{Z}$'s view of an interaction with $\mathcal{S}$ and $\mathcal{F}_{\text{CERT}}$ is distributed identically to its view of an interaction with parties running protocol CAS in the $(\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{CA}})$-hybrid model. □

Remark: In protocol CAS each party contacts $\mathcal{F}_{\text{CA}}$ only once, and records the verification key it receives for future verifications. Alternatively, a verifier may obtain the verification key from $\mathcal{F}_{\text{CA}}$ upon each signature verification. This would make sense in settings where identities can be revoked, or when the verifier does not maintain state between verifications.

4 Message authentication given $\mathcal{F}_{\text{CERT}}$

We exemplify the usefulness of $\mathcal{F}_{\text{CERT}}$ by demonstrating how it can be used to obtain authenticated communication. Specifically, we recall the message authentication functionality, $\mathcal{F}_{\text{AUTH}}$, from [C01], and show a simple protocol that realizes $\mathcal{F}_{\text{AUTH}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model. The protocol is essentially the signature-based authenticator from [BCK98]. (In fact, we use a corrected formulation of $\mathcal{F}_{\text{AUTH}}$; see below.)

We complete this section by proving a complementary claim: $\mathcal{F}_{\text{AUTH}}$ cannot be realized in the bare unauthenticated model. This claim formalize the intuitive notion that authenticated communication cannot be "bootstrapped" without some initial, out-of-band, authentication of the entities involved. Furthermore, it implies that $\mathcal{F}_{\text{CERT}}$ cannot be realized in the bare, unauthenticated model.

$\mathcal{F}_{\text{AUTH}}$ is presented in Figure 5.[3] Recall that each copy of $\mathcal{F}_{\text{AUTH}}$ handles a single message; this simplifies the presentation and analysis of protocols for realizing it. Section 4.1 demonstrates how to realize $\mathcal{F}_{\text{AUTH}}$ given $\mathcal{F}_{\text{CERT}}$. Section 4.2 demonstrates the impossibility of realizing $\mathcal{F}_{\text{AUTH}}$ in the bare model.

Functionality $\mathcal{F}_{\text{AUTH}}$

1. Upon receiving $(\texttt{send}, \mathit{sid}, B, m)$ from party A, send $(\texttt{sent}, \mathit{sid}, A, B, m)$ to the adversary.
2. Upon receiving $(\texttt{send}, \mathit{sid}, B', m')$ from the adversary, do: If A is corrupted then output $(\texttt{sent}, \mathit{sid}, A, m')$ to party B'. Else, output $(\texttt{sent}, \mathit{sid}, A, m)$ to party B. Halt.

Figure 5: The Message Authentication functionality, $\mathcal{F}_{\text{AUTH}}$

4.1 Realizing $\mathcal{F}_{\text{AUTH}}$ given $\mathcal{F}_{\text{CERT}}$

We present a protocol for realizing $\mathcal{F}_{\text{AUTH}}$ given ideal access to $\mathcal{F}_{\text{CERT}}$. The protocol is very simple: To send an authenticated message m to party B, with session identifier *sid*, party A simply signs (m, B) and sends the signed message to B. A more complete description appears in Figure 6.

Protocol SBA

1. Upon receiving an input $(\texttt{Send}, \mathit{sid}, B, m)$, party A sets $\mathit{sid}' = (A, \mathit{sid})$, sets $m' = (m, B)$, sends $(\texttt{Sign}, \mathit{sid}', m')$ to $\mathcal{F}_{\text{CERT}}$, obtains the response $(\texttt{Signed}, \mathit{sid}', m', s)$, and sends (sid, A, m, s) to B.
2. Upon receiving (sid, A, m, s), B sets $\mathit{sid}' = (A, \mathit{sid})$, sets $m' = (m, B)$, sends $(\texttt{Verify}, \mathit{sid}', m', s)$ to $\mathcal{F}_{\text{CERT}}$, and obtains a response $(\texttt{Verified}, \mathit{sid}', m', s, f)$. If $f = 1$ then B outputs $(\texttt{Sent}, \mathit{sid}, A, B, m)$ and halts. Else B halts with no output.

Figure 6: The signature-based authentication protocol, SBA

Several remarks are in order before setting to prove security of the protocol. First, notice that the protocol contains no explicit mechanisms to protect against adversarial replay of messages. Indeed, the protocol relies on the uniqueness of the session-identifier for each instance of the protocol.

[3] The formulation here is somewhat different than the one in [C01, CK02]: the prior formulation failed to let the adversary change the delivered message and identity of the recipient in case that the sender gets corrupted between sending and delivery. This results in an unnecessarily strong guarantee (that is in fact unrealizable by reasonable protocols). This oversight in the previous formulation was pointed out in a number of places (e.g. [HMS03, AF04]), with respect to related functionalities. We note that [C04] still uses the original (overly strong) version of $\mathcal{F}_{\text{AUTH}}$.

This in essence puts the burden of protecting from replay on the protocol that uses SBA (or, rather, on an "operating system" that verifies uniqueness of sid's). Specifically, it is assumed that if the receiver B obtains outputs from two different copies of SBA, and these two copies have the same sid, then the second output is discarded. While we do not specify how this provision is implemented, we mention two popular ways to do so. One method is to have the recipient maintain a list of past session-identifiers of instances of SBA (potentially grouped by sender identities, and expected to be increasing in value, for more efficient storage). A second method is to have the recipient contribute to the sid by having the parties exchange randomly chosen nonces prior to the initiation of the protocol, and using the concatenation of the nonces as the sid. This method involves an additional round-trip prior to the one-message protocol, and also introduces a small error probability, but has the advantage that no state needs to be kept across protocol instances. See [BLR04] for more discussion and formalization of general methods for guaranteeing uniqueness of SID's. It should also be noted that the protocol obtained by using SBA with the above nonce-based exchange is essentially the signature-based authenticator of [BCK98].

Second, observe that a separate instance of SBA is invoked for each message transmission. This simplifies the protocol and analysis (e.g., there is no need to sign the session identifier), but it also means that a separate copy of $\mathcal{F}_{\text{CERT}}$ is used per message. Using the construction from Section 3, we have that a different instance of a signature scheme is needed for each message, which is of course highly wasteful. However, as shown in [CR03], it is possible to realize multiple instances of $\mathcal{F}_{\text{CERT}}$ (with the same signer) using a single copy of $\mathcal{F}_{\text{CERT}}$, by including the session identifier in the signed text. Using universal composition with joint state, we have that multiple instances of protocol SBA can use the same instance of $\mathcal{F}_{\text{CERT}}$.

Third, we note that functionality $\mathcal{F}_{\text{CERT}}$ can be used also for session-based message authentication. Specifically, the formalization of key-exchange protocols in [CK02] can be readily adapted to use $\mathcal{F}_{\text{CERT}}$. This would simplify the current formalization that uses $\mathcal{F}_{\text{SIG}}$ plus initially authenticated communication between any pair of parties. In addition, using $\mathcal{F}_{\text{CERT}}$ and $\mathcal{F}_{\text{CA}}$ better models the practice of using certificate authorities. Finally, we note that the security of protocol SBA is unconditional, with no computational assumptions and no error probability.

Claim 4 *Protocol* SBA *securely realizes $\mathcal{F}_{\text{AUTH}}$ in the $\mathcal{F}_{\text{CERT}}$-hybrid model.*

Proof: Let $\mathcal{A}$ be an adversary that interacts with parties running SBA in the $\mathcal{F}_{\text{CERT}}$-hybrid model. We construct an ideal-process adversary (simulator) $\mathcal{S}$ such that the view of any environment $\mathcal{Z}$ from an interaction with $\mathcal{A}$ and SBA is distributed identically to its view of an interaction with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{AUTH}}$. As usual, simulator $\mathcal{S}$ runs a copy of $\mathcal{A}$, and forwards all messages from $\mathcal{Z}$ to $\mathcal{A}$ and back. In addition, $\mathcal{S}$ proceeds as follows.

Simulating the sender. When an uncorrupted party A is activated with input $(\texttt{Send}, sid, B)$, $\mathcal{S}$ obtains this value from $\mathcal{F}_{\text{AUTH}}$. Then, $\mathcal{S}$ simulates for $\mathcal{A}$ the expected interaction with $\mathcal{F}_{\text{CERT}}$; that is, $\mathcal{S}$ sends to $\mathcal{A}$ the message $(\texttt{Sign}, (A, sid), (m, B))$ from $\mathcal{F}_{\text{CERT}}$, and obtains a value s from $\mathcal{A}$. Next, $\mathcal{S}$ hands $\mathcal{A}$ the message (sid, A, m, s) sent from A to B.

If the sender is corrupted, then all that $\mathcal{S}$ has to do is to simulate for $\mathcal{A}$ the interaction with $\mathcal{F}_{\text{CERT}}$. That is, whenever a corrupted A sends a message $(\texttt{Sign}, sid'', m'')$ to $\mathcal{F}_{\text{CERT}}$, $\mathcal{S}$ responds with $(\texttt{Sign}, sid'', m'')$ to $\mathcal{A}$, obtain a signature s'', and sends $(\texttt{Signature}, sid'', m'', s'')$ to A in the namer of $\mathcal{F}_{\text{CERT}}$.

Simulating the recipient. When $\mathcal{A}$ delivers a message (sid, A, m, s) to an uncorrupted party B, $\mathcal{S}$ first simulates for $\mathcal{A}$ the interaction with $\mathcal{F}_{\text{CERT}}$: if the logic of $\mathcal{F}_{\text{CERT}}$ would instruct it to send (Verify, $sid' = (A, sid), m' = (m, B), s$) to $\mathcal{A}$ (that is, if A is corrupted, or m' was signed in the past but with a signature different than s) then send this message to $\mathcal{A}$, and record the response of $\mathcal{A}$. Next, if the logic of $\mathcal{F}_{\text{CERT}}$ would instruct it to output (Verified, $sid', m', s, f = 1$) to B (that is, if $\mathcal{A}$ responded with $f = 1$ or the message m' was recorded with signature s) then deliver the message (Receivedsid, A, B, m) which was sent in the ideal process from $\mathcal{F}_{\text{AUTH}}$ to B. Otherwise, do nothing.

Simulating party corruptions. Whenever $\mathcal{A}$ corrupts a party, $\mathcal{S}$ corrupts the same party in the ideal process, and provides $\mathcal{A}$ with the internal state of the corrupted party. This is straightforward to do, since the protocol maintains no secret state at any time.

It can be readily seen that the combined view of $\mathcal{Z}$ and $\mathcal{A}$ in an execution of SBA is distributed identically to the combined view of $\mathcal{Z}$ and the simulated copy of $\mathcal{A}$ within $\mathcal{S}$ in the ideal process. Indeed, the only case where the two views may potentially differ is if the receiver obtains (Verified, $sid', m', s, f = 1$) from $\mathcal{F}_{\text{CERT}}$ for an incoming message (sid, A, m, s), while A is uncorrupted at the time when the message was delivered, and never sent the message (sid, A, m, s). However, if A never sent (sid, A, m, s), then the message $m' = (m, B)$ was never signed by the copy of $\mathcal{F}_{\text{CERT}}$ with session id (A, sid); thus, according to the logic of $\mathcal{F}_{\text{CERT}}$, B would always obtain (Verified, $sid', m', s, f = 0$) from$\mathcal{F}_{\text{CERT}}$. □

4.2 Realizing $\mathcal{F}_{\text{AUTH}}$ in the bare model

This section demonstrates that it is impossible to realize $\mathcal{F}_{\text{AUTH}}$ in the bare unauthenticated model by any "useful" protocol. A corollary from this fact is that there exist no "useful" protocols that realize $\mathcal{F}_{\text{CERT}}$ in the plain model. More precisely, say that a multiparty protocol is useful if, whenever the adversary corrupts no party and delivers all messages unmodified and with no delay, then at least one party generates output with non-negligible probability.[4]

Claim 5 *There exist no useful protocols that realize $\mathcal{F}_{\text{AUTH}}$ in the bare unauthenticated model in a network with at least two parties.*

Proof: Let π be a protocol (that is geared towards realizing $\mathcal{F}_{\text{AUTH}}$), and consider a network with parties A, B. We construct the following environment $\mathcal{Z}$ and real-life adversary $\mathcal{A}$. $\mathcal{Z}$ activates no party with any input. If party B generates output (Received, $sid = 0, A, m = 0$), then $\mathcal{Z}$ outputs 1. Otherwise $\mathcal{Z}$ outputs 0.[5] Adversary $\mathcal{A}$ simulates for B an execution of π on input (Send, $sid = 0, B, m = 0$) for A, where no party is corrupted, and all messages are delivered without delay. Note that $\mathcal{A}$ can do so successfully, since it can feed B with any incoming messages, and B shares no prior state with any other party.

Since protocol π is useful, with non-negligible probability B outputs (Received, $0, A, 0$) in the real execution of π. However, in the ideal process B never generates any output. □

[4] Delivery with no delay can of course be interpreted in a number of ways. To be specific, we stick to "first come first serve" delivery, where the earliest undelivered message is the next to be delivered. The claim holds with respect to other reasonable delivery method.

[5] This instruction seems hard to implement in an asynchronous network, since $\mathcal{Z}$ cannot wait "until B generates output". We thus interpret this instruction as follows: $\mathcal{Z}$ first writes 0 on its output tape. Next, if B generates the said output, then $\mathcal{Z}$ overwrites 1 on its output tape. Now, recall that the output of the execution is defined as the contents of the output tape of $\mathcal{Z}$ when the execution terminates (i.e, when all involved entities either terminate or are at a waiting state). Thus, the present interpretation achieves the desired effect.

Corollary 6 *There exist no useful protocols that realize $\mathcal{F}_{\text{CERT}}$ in the bare unauthenticated model, in a network with at least two parties.*

Proof: If there exist protocols that realize $\mathcal{F}_{\text{SIG}}$ then the corollary follows from Claims 5 and 4. Otherwise, the corollary follows from the fact that $\mathcal{F}_{\text{SIG}}$ can be (trivially) realized in the $\mathcal{F}_{\text{CERT}}$-hybrid model. □

Acknowledgments

Sincere thanks to the many people that have interacted with me on issues of modeling signatures, certification, and authentication within the UC framework, and have provided me with valuable criticism of prior formulations. A far from exhaustive list includes Hugo Krawczyk and Tal Rabin (who have bravely endured endless discussions and constantly changing formulations), Daniele Micciancio (discussions with whom led to Claim 5), Boaz Barak (who helped identify the need to allow corrupt signers to accept any signature as their own), Michael Backes (who has pointed out the need for a signature functionality that does not disclose the signed messages and signature strings to the adversary), Dennis Hofheinz, Joern Mueller-Quade, and Rainer Steinwandt (who have pointed out the need for the relaxation of the notion of PPT ITM), as well as Masayuki Abe, Ivan Damgaard, Yehuda Lindell, Phil Mackenzie, Jesper Nielsen and Ke Yang. I am also grateful to the CSFW'04 reviewers for their very useful comments.

References

[ISO93] ISO/IEC IS 9798-3, Entity authentication mechanisms — Part 3: Entity authentication using asymmetric techniques, 1993.

[AR00] M. Abadi and P. Rogaway, Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption), *International Conference on Theoretical Computer Science IFIP TCS 2000,* LNCS, 2000. On-line version at http://pa.bell-labs.com/ abadi/.

[AF04] M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In proceedings of Crypto'04, 2004.

[BH03] M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. *Cryptology ePrint Archive, Report 2003/240*, 2003, http://eprint.iacr.org/.

[BPW03a] M. Backes, B. Pfitzmann and M. Waidner, A Universally Composable Cryptographic Library, *10th ACM CCS,* 2003. Full version in http://eprint.iacr.org/2003/015.

[BPW03b] M. Backes, B. Pfitzmann, M. Waidner. Reactively Secure Signature Schemes. ISC 2003: 84-95.

[BPW04] M. Backes, B. Pfitzmann and M. Waidner, A General Composition Theorem for Secure Reactive Systems, *1st TCC,* 2004.

[BLR04] B. Barak, Y. Lindell and T. Rabin, A Note on Secure Protocol Initialization and Setup in Concurrent Settings, *Cryptology ePrint Archive,* http://eprint.iacr.org/2004/006.

[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.

[c01] R. Canetti. A unified framework for analyzing cryptographic protocols. ECCC TR 01-16. Also available at http://eprint.iacr.org/2000/067.

[c04] R. Canetti. Universally Composable Signatures, Certification, and Authentication. Proceedings of the *17th Computer Security Foundations Workshop (CSFW),* June 2004.

[chh00] R. Canetti, S. Halevi and A. Herzberg, How to Maintain Authenticated Communication, *Journal of Cryptology,* Vol. 13, No. 1, winter 2000. Preliminary version at *16th Symp. on Principles of Distributed Computing (PODC),* ACM, 1997, pp. 15-25.

[chk04] R. Canetti, S. Halevi, and J. Katz. Chosen-Ciphertext Security from Identity-Based Encryption. *Eurocrypt '04,* 2004. Also available at http://eprint.iacr.org/2003/182.

[ck02] R. Canetti and H. Krawczyk. Universally Composable Key Exchange and Secure Channels . In *Eurocrypt '02*, pages 337–351, 2002. LNCS No. 2332.

[cr03] R. Canetti and T. Rabin, Universal Composition with Joint State, *Crypto'03,* 2003. Also available at http://eprint.iacr.org/2002.

[tls] T. Dierks and C. Allen. The Transport Layer Security (TLS) protocol. IETF RFC 2246, http://www.ietf.org/rfc/rfc2246.txt, 1999.

[dh76] W. Diffie and M. Hellman, New directions in cryptography, *IEEE Trans. Info. Theory* IT-22, November 1976, pp. 644–654.

[ddn00] D. Dolev, C. Dwork and M. Naor, Non-malleable cryptography, *SIAM. J. Computing,* Vol. 30, No. 2, 2000, pp. 391-437. Preliminary version in *23rd STOC,* ACM, 1991.

[dy83] D. Dolev and A. Yao, On the security of public-key protocols, *IEEE Transactions on Information Theory,* 2(29), 1983.

[g01] O. Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2001. Preliminary version *http://philby.ucsd.edu/cryptolib.html/.*

[gmRa89] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. 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.

[go92] S. Goldwasser and R. Ostrovsky, Invariant Signatures and Non-Interactive Zero-Knowledge Proofs Are Equivalent, *Crypto '92,* 1992, pp. 228-245.

[hms03] D. Hofheinz, J. Muller-Quade, R. Steinwandt. Initiator-Resilient Universally Composable Key Exchange. In proceedings of ESORICS'03, 2003.

[hms03a] D. Hofheinz, J. Mueller-Quade, and R. Steinwandt, personal communication, March 2003.

[ipsec] D. Harkins and D. Carrel, ed., The Internet Key Exchange (IKE), *RFC 2409,* Nov. 1998.

[lmms99] P. Lincoln, J. C. Mitchell, M. Mitchell and A. Scedrov. Probabilistic Polynomial-Time Equivalence and Security Analysis. In *World Congress on Formal Methods* 1999: 776-793.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption ins the presence of active adversaries. *Theory of Cryptography Conference (TCC) '04*, 2004.

[MS03] A. Menezes and N. Smart, Security of signature schemes in a multi-user setting, *Designs, Codes and Cryptography,* to appear.

[MRV99] S. Micali, M. Rabin, and S. Vadhan, Verifiable Random Functions, *40th Annual Symposium on Foundations of Computer Science,* 1999.

[P95] B. Pfitzmann. Sorting Out Signature Schemes. CWI Quarterly 8/2 (1995) 147-172.

[PSW00] B. Pfitzmann, M. Schunter and M. Waidner. Provably Secure Certified Mail. IBM Research Report RZ 3207 (#93253), IBM Research, Zurich, August 2000.

[PW00] B. Pfitzmann and M. Waidner, Composition and integrity preservation of secure reactive systems, *7th ACM CCS*, 2000, pp. 245-254.

[PW01] B. Pfitzmann and M. Waidner, "A model for asynchronous reactive systems and its application to secure message transmission", IEEE Symposium on Security and Privacy, May 2001. Preliminary version in http://eprint.iacr.org/2000/066 and IBM Research Report RZ 3304 (#93350), IBM Research, Zurich, December 2000.

[SM93] P. Syverson and C. Meadows, A Logical Language for Specifying Cryptographic Protocol Requirements, *14th IEEE Symp. Sec. Priv.,* 1993.

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 [C01], with some local updates and modifications. Full details (as well as a history of works leading to that framework) appear there. We first 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, G01], 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, representing external information. From a complexity-theoretic point of view, this essentially implies that adversaries are non-uniform ITMs. Each ITM has a session-identifier (SID) that describes which session (or, protocol instance) the ITM belongs to. It also has a party identifier (PID) that describes the role (or, participant identifier) of that ITM within the protocol instance. The pair (SID,PID) is guaranteed to be unique in the system.

We assume that all ITMs run in probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most n^c, where n is the overall number of bits written on the *input tape* of M in this run, plus $k * t$, where k is the security parameter k and t the number of activations with new input M

had in this run. (Figuratively, if we assume that the value 1^k is prepended to each input value to M, then n is the overall input length.) In the notion of PPT ITMs from that of [C01]. [6]

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.

Protocol execution in the real-life model. We sketch the process of executing a given protocol π (run by some set of parties) with an 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 party) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is the first to be activated. In each activation it may read the contents of the output tapes of all parties, it may invoke a new party that runs the current instance of the protocol, or it may write information on the input tape of either one of the parties or of the adversary.

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, or report information to $\mathcal{Z}$ by writing this information on its output tape. The delivered messages need not bear any relation to the messages sent by the parties. (This essentially means that the underlying communication model is unauthenticated.) Upon corrupting a party, the adversary gains access to all the tapes of that party and controls all the party's future actions. 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).

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 a local output on its output tape or an outgoing message on its outgoing communication tape. Once the activation of the party is complete the environment is activated.

[6] This formalization is different than in prior versions of [C01], where it is required that the overall running time is bounded by k^c for some c. This prior formalization is indeed adequate for capturing the notion of "polynomial time protocols" for tasks where the number of activations expected from a party, and the length of the input per activation, are a-priori bounded. (For instance, secure function evaluation is a class of such tasks.) However, it is too restrictive for capturing tasks where the number of activations expected from a party, and the length of the input per activation, may not be a-priori bounded. As an example, consider the intuitive notion of a signature scheme. This notion assumes that the number and lengths of messages to be signed are not a-priori bounded. Rather, these numbers are determined by the adversary with which the scheme interacts; while they are always polynomially bounded, the polynomial may depend on the adversary. In particular, the ideal signature functionality $\mathcal{F}_{\text{SIG}}$ described above is not PPT ITM according the the [C01] notion, and cannot be realized by PPT ITMs according to that notion. We thank Dennis Hofheinz, Joern Mueller-Quade, and Rainer Steinwandt for pointing this issue to us in [HMS03a].

The protocol execution ends when the environment halts. 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 $P_1, ..., P_n$ running protocol π on security parameter k, input z and random input $\vec{r} = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1 \ldots r_n$ 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.

The order of activations in the ideal process is determined as in the real-lie model. In particular, as there, the environment is activated first, and 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 $\mathcal{F}$. Whenever a dummy party is activated due to delivery of some output from $\mathcal{F}$ it copies this output to its own output tape.

Once $\mathcal{F}$ is activated, it reads the contents of its incoming communication tape, and potentially gives output to parties or sends messages to the adversary $\mathcal{S}$.

Once the adversary $\mathcal{S}$ is activated, it may read its own input tape and the messages sent to it from $\mathcal{F}$. (Note that $\mathcal{S}$ *cannot* see the inputs or outputs of the parties.) $\mathcal{S}$ may either send a message to $\mathcal{F}$, or corrupt a party. 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.

As in the real-life model, the protocol execution ends when the environment halts. 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 ρ 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 ρ 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 ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$.

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 called* 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 π be an n-party protocol. We say that π* 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 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 $\mathcal{F}$. The copies of $\mathcal{F}$ are differentiated using their SIDs (the RIDs of all copies of $\mathcal{F}$ are null.) All messages addressed to each copy and all message sent by each 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 activated next. (If no such copy of $\mathcal{F}$ exists then a new copy of $\mathcal{F}$ is created and activated next.) Similarly, once a copy of $\mathcal{F}$ generates output for a dummy party, the dummy party is activated.

The 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 π, adversary $\mathcal{A}$, analogously to the definition of $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$. (We stress that here π 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 π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ (with respect to some class of adversaries). The composed protocol π^ρ is constructed by modifying the code of each ITM in π so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of ρ 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 ρ, with the contents of that message given to ρ as new input. Each output value generated by a copy of ρ is treated as a message received from the corresponding copy of $\mathcal{F}$.

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

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

Theorem 9 (Universal composition [C01]**)** *Let $\mathcal{F}, G$ be ideal functionalities. Let π be a protocol in the $\mathcal{F}$-hybrid model, and let ρ be a protocol that securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then for any adversary $\mathcal{A}_\mathcal{G}$ there exists an adversary $\mathcal{A}_\mathcal{F}$ such that for any environment $\mathcal{Z}$ we have*

$$\text{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A}_\mathcal{F},\mathcal{Z}} \approx \text{EXEC}^{\mathcal{G}}_{\pi^\rho,\mathcal{A}_\mathcal{G},\mathcal{Z}}.$$

A corollary of the general theorem states that if π securely realizes some functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model, and ρ securely realizes $\mathcal{F}$ in the real-life model, then π^ρ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model.

B On the [BPW03a] modeling of signatures

Backes, Pfitzmann and Waidner [BPW03a] propose a "library" of idealized cryptographic primitives, within the framework of [PW01] (which is an asynchronous variant of [PW00]).[7] The main goal of this library is similar to one of the goals of this work, namely to realize the "Dolev-Yao paradigm" (sketched in the introduction) in a computationally sound way. Since this library provides, among other things, an alternative abstract modeling of digital signatures, we review it here in terminology that facilitates comparison and provide some discussion and comparison to the formalism of this work.

The [BPW03a] library is essentially an ideal functionality that provides, within a single copy, multiple instances of an ideal signature service, as well as multiple instances of an ideal public-key encryption scheme, multiple instances of secure or authenticated communication channels, and ideal nonce generation. We concentrate here on the details relevant to the signature service. The service has the following interfaces: [8]

- Registration: In order to register as a signer, a party sends a registration request to the library. It then receives a "key-handle", which is essentially the current value of a "key-handle counter"; in addition, the library internally registers the party as the signer for this handle.
- Signature generation: In order to sign a message m with respect to some key-handle, the signer sends m and the key-handle to the library. In response, the library records m as signed, and provides the signer with a "signature-handle", which is again the current value of some counter. Finally, the library internally records the signer as a "legitimate verifier" for this signature-handle.
- Signature verification: When a party that is registered as a "legitimate verifier" for some signature-handle wishes to verify the signature, it sends a verification request to the library

[7] This framework has many similarities to the UC framework, although the formalization is somewhat different. In particular, the universal composition theorem holds also in that framework; see [BPW04] for details.

[8] We note that the description of the library interface in [BPW03a] is much more detailed. To improve readability, the sketch here is much more high-level and informal. Still, to the best of our understanding it provides an accurate depiction of the [BPW03a] formalism.

(together with the message and the handle). The library then verifies that the party is registered as a "legitimate verifier". If so, then it reports back whether the (message, handle) pair is recorded. Otherwise, the request is ignored.

These are the basic interface functions of the signature scheme. Notice, however, that these functions do not allow any party other than the signer to verify signatures. Transferring the ability to verify signatures is provided via other functions of the library, namely the communication functions. That is:

- Signature transmission: When a party A that is registered as a "legitimate verifier" for some signature-handle asks to send the handle of some signature to another party B via a communication channel provided by the library, the library provides B with a corresponding signature-handle and *adds B to the list of "legitimate verifiers" for this signature-handle.* (More precisely, to accommodate asynchronous and unauthenticated channels, the library provides B with a handle only when the adversary explicitly instructs to deliver this signature handle to B.)[9]

Discussion. An attractive property of this formalism is that it "abstracts out" public keys and signature tags. Instead, users are ideally notified whether a given message was signed. (This is somewhat reminiscent of functionality $\mathcal{F}_1$ in Section 2.1.) This frees the application protocols that use the functionality from dealing with public-keys and signature strings. However, this high level of abstraction has a number of side-effects:

First, we note that the above formulation of the library forces the user of the library to use *only* the communication channels provided by the library to transmit signature handles. It is impossible to use other communication mechanisms, or even communication channels that are provided by a different copy of the library, and maintain the ability to verify signatures. This means that all instances of all signature schemes in a given system must be analyzed as a single unit. Furthermore, all instances of all protocols that use these signature schemes must be analyzed as a single unit. (This is so, since all these instances use the same common instance of the library, and thus have some joint state.) This applies also to non-cryptographic application protocols that use signatures as a small part of their function. Consequently, the present formulation of the library seems to preclude modular analysis of the systems that use it.[10]

Another property of the present formulation of the library is that it is unable to model more general ways of communicating signatures to other parties. For instance, consider a (real-life) protocol where a party wishes to secret-share the signature among several parties, and then have another party determine who will be able to reconstruct the signature. Such use of signature scheme cannot be modeled within the present formulation.

Finally, we note the following difficulty that arises when trying use the [BPW03a] library in order to realize the "Dolev-Yao paradigm" as sketched in the introduction. Recall that the paradigm starts with a concrete protocol that uses some cryptographic primitive (say, a signature scheme),

[9] This "list of legitimate verifiers," while somewhat hidden inside the [BPW03a] formalism, is essential for making the functionality realizable. Indeed, it is easy to see that if parties could verify validity of a signature without being explicitly given the corresponding signature handle by the library then the library would not be realizable at all. See Section 2.1 for more discussion on these issues.

[10] One could potentially hope to use Universal Composition with Joint State (JUC) [CR03] to separately analyze different protocols that use the same instance of a signature scheme. However, the present formalism of the library, and in particular the fact that all instances of signatures must use a joint communication module, seems to preclude the use of this theorem.

de-composes the protocol into a "high-level module" that uses an abstraction of the primitive and a module that realizes the abstract primitive, analyzes each module separately, and then uses a composition theorem to deduce that the original protocol is secure. However, if the present library is used, then we need to de-compose the given protocol into a "low-level" module that realizes the library, plus a "high-level module" that uses the library. It is stressed if one wants to use the composition theorem then the low level module must realize the *entire library.* However, most cryptographic protocols do not contain any module that realizes the entire library. Therefore, they cannot be de-composed as needed. Indeed, the [BPW03a] library cannot be used as-is to analyze such protocols; further mechanisms are needed to validate its applicability.

In conclusion, we note that all the issues discussed here are related to the fact that the library models all instances of all primitives within a single copy of an ideal functionality, which also handles all the communication. Indeed, a formalization where each copy of the functionality captures only a single instance of a single primitive seems more conducive towards effective realization of the Dolev-Yao paradigm, and better suited for modular analysis of large, multi-user, multi-module systems.

Erratum. In the previous version of this work the account in this appendix of the [BPW03a] work contained an error: It was claimed that the [BPW03a] library requires that the "Signature transmission" operation described above be carried out only over authenticated channels. This claim is incorrect, and results from our misunderstanding of the [BPW03a] text. We apologize for the mistake and thank Birgit Pfitzmann and Michael Waidner for correcting us.