On Universally Composable Notions of Security for Signature, Certification and Authentication

Ran Canetti*

November 16, 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. While 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 formal and automatable analysis of such systems), 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 motivate the choices made in that definition. 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. Indeed, given ideal signatures, authenticated communication can be obtained in an unconditional and errorless way; this facilitates potential formal and automated analysis.

*IBM T.J. Watson Research Center. Email: canetti@watson.ibm.com.

Contents

1 Introduction

Digital Signatures (first proposed by Diffie and Hellman in [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 network. Here signature-based authenticated key exchange protocols plays a major role (see e.g. [DOW92, ISO93, IPSEC98]). Other uses include guaranteeing various integrity properties within cryptographic protocols (see e.g., [DDN00]).

A widely accepted formalization of the security requirements from signature schemes was put forth in [GMRi88]. Essentially, the requirement it that, when the public and secret key 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 on 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.

Recently, some efforts were made toward defining the security requirements from signature schemes using the approach of emulating an ideal process [C01, CK02, CR03, BPW03a]. (The first three works use the framework of universally composable (UC) security; the last one uses the framework of [PW00].) In this approach, one formulates an "ideal signature functionality" that captures the desired security properties of signature schemes; 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; 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 ideal-model based analysis for signature schemes is attractive is that such analysis provides strong secure composability properties. Indeed, in both frameworks mentioned above, security was shown to be preserved under universal composition [C01, BPW04]. This composability property opens the door to a number of potential advantages over the "standard" notion of CMA-security. Let us highlight two main ones.[1]

Enabling component-wise analysis of complex systems: A common design methodology for protocols that use signature schemes is to have multiple instances of a protocol (or even multiple different protocols) use the same instance of a signature scheme. Examples include authenticated key exchange protocols, authenticated broadcast protocols, and more. When the standard formulation of CMA-security is used, we are forced to analyze all protocol instances that use the same instance of the signature scheme as a single, complex unit. In contrast, using composable notions, it is possible to analyze each protocol instance as stand alone, and then use a general composition theorem (such as, say, universal composition with joint state [CR03]) to deduce the security of the entire, multi instance system.

Sound realization of the "Dolev-Yao methodology": Given an ideal signature functionality (and using the universal composition theorem), it is possible to use the following modular approach to protocol analysis: (a) Given a concrete, real-life protocol that uses a signature scheme, first decompose the protocol into a "signature module" and a "high-level module". (b) Prove that the

[1] Several other abstractions of the security requirements from signature schemes were of course made in the past, e.g. [SM93, CHH00]. However, none of these abstractions enjoy the secure composability property which is central to our goals.

"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 a 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, and is representable within formal proof tools. Furthermore it seems amenable to eventual automation. Indeed, Dolev and Yao [DY83] have proposed the approach of analyzing the "high-level module" assuming access to an ideal signature oracle, and representing the high-level protocol in a language that allows applying formal logic tools. Using the above approach, we can obtain a variant of the "Dolev-Yao methodology" that guarantees security also for the original concrete protocol that does not use idealized constructs.

It turns out that the task of providing a good ideal-process-based composable notion of security for signature schemes is not trivial. There have been two main approaches to defining an "ideal signature functionality" within a composable-security framework. The first one is the approach of [C01]. This approach aims at capturing the basic security properties of a signature scheme as a tool within other protocols, rather than as an application by itself. Here each instance of $\mathcal{F}_{\text{SIG}}$ 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], contain a number of technical flaws that make the claim of equivalence to CMA security incorrect.

An alternative approach to formalizing ideal signature schemes was subsequently formalized in [BPW03a]. This formulation provides a more abstract (and somewhat restricted) modeling of signatures. Furthermore, it captures, 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 and secure communication channels. (See more discussion on this modeling within. Let us only remark here that this "monolithic" approach essentially loses much of the ability to carry out modular analysis of systems and much of the applicability to sound realization of Dolev-Yao.)

The present work has several contributions. First, we present a corrected (and somewhat simplified) formulation of the signature functionality of [C01, CK02, CR03]. While the corrections are rather technical, they are necessary for the equivalence with CMA-security of signature schemes. We identify three points; 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 the ideal signature functionality 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 our signature functionality is aimed at capturing the basic properties of a (single instance of) a CMA-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" $\mathcal{F}_{\text{CERT}}$. It is intuitively clear that $\mathcal{F}_{\text{CERT}}$ cannot be realized in a bare communication model where there are no a-priori means for authenticating the sender of a message. (We formalize and prove this statement, see below). We thus 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 $\mathcal{F}_{\text{CA}}$. We show that $\mathcal{F}_{\text{CERT}}$ can be realizes 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.

Several remarks are in order here. 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 casting of these protocols and analyses (as well as similar ones) within formal and automated analysis tools. (The above two points can be regarded as manifestations of the above-mentioned advantages of our formalization of signature schemes.)

Third, we note that an alternative way of using $\mathcal{F}_{\text{SIG}}$ to provide authenticated communication is provided in [CK02]. There, $\mathcal{F}_{\text{SIG}}$ is used to realize key-exchange protocols, which are in turn used to obtain secure communication sessions via symmetric encryption and authentication. That formulation is considerably more complex than the present one. In particular, even when given ideal signatures, the construction is still only computationally secure. We remark, however, that the treatment of [CK02] can be simplified by using $\mathcal{F}_{\text{CERT}}$ (as formalized here), rather than directly using $\mathcal{F}_{\text{SIG}}$ and making some set-up assumptions.

In addition to the above modular treatment of authentication given signatures, we provide a somewhat unrelated alternative formalization of signature schemes. This formalization is aimed at overcoming the following drawback of $\mathcal{F}_{\text{SIG}}$: The formulation of $\mathcal{F}_{\text{SIG}}$ allows the adversary to learn the values of all signed messages and signatures generated in the system, as soon as they are generated. While this property is in accordance with the basic notion of CMA-security (which does not make any secrecy requirements from signatures), it becomes problematic when coming to use signatures in a context where the signature should be revealed only to a restricted set of parties. The alternative formalization, denoted $\mathcal{F}_{\text{PRIV-SIG}}$ (for "privacy-preserving signatures") guarantees that signatures remains secret until explicitly publicized by the signer, while maintaining all the other properties of $\mathcal{F}_{\text{SIG}}$.

Finally, we remark that the attempt to formulate an ideal process that captures the security properties of signature schemes brings up a number of interesting issues pertaining to the expected security properties from such schemes. Let us highlight three such issues (see more discussion within):

- 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 on 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 on m with respect to v'?

- 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?

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 presents the privacy preserving signature functionality. Appendix B reviews the UC framework, and presents some updates to it (specifically, to the notion of PPT ITMs). Finally, Appendix C provides a critical review of the [BPW03a] formalization of an ideal signature functionality.

2 The basic signature functionality

This section presents a corrected version ot the ideal signature functionality of [C01, CK02, CR03]. We also use this opportunity to provide an extended motivation for the definitional approach, as well as the specific choices made. (Many 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 considerations leading to the present formalization of the basic signature functionality, $\mathcal{F}_{\text{SIG}}$. The discussion also describes and motivates some of the definitional choices. Further motivation is provided in subsequent sections. Section 2 presents the updated formulation of $\mathcal{F}_{\text{SIG}}$, which includes some corrections and simplifications.

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 notion

of a "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 "registration process", where a party registers as a signer and obtains a "public verification key" from the functionality. (Since we do not give any meaning to the actual value of the verification key, we allow the adversary to choose it.) 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 must take care of 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. This of course does not correspond to our intuitive notion of a signature scheme. (In addition, it mandates that a signature protocol employs authenticated communication channels. This is a strange requirement, given that signature schemes are often used to *set up* such channels.) 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 a "signature string", that will be presented at verification time. (Also here, since we do not give any meaning to the actual value of the signature string, we allow the adversary to choose it.) The ideal verification process will use the message, the 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.

We remark that the formalization of ideal signature schemes within the "ideal cryptographic library" of [BPW03a] is reminiscent of the approach taken by functionality $\mathcal{F}_1$ above. See more discussion on this formalization in Appendix C.

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

This section presents an ideal functionality that captures the basic properties of digital signature schemes. Following the discussion in the previous sections, the functionality models signature schemes as a process that allows a distinguished party (the signer) to attach tags (signatures) to documents, so that everyone can verify, by locally running some public algorithm, that the signature was generated by no one else but the signer. This is a more "low-level" abstraction that regards

signature schemes as a "technical tool" within other protocols. We show that realizing $\mathcal{F}_{\text{SIG}}$ is essentially *equivalent* to existential unforgeability against chosen message attacks as in [GMRi88].

Functionality $\mathcal{F}_{\text{SIG}}$ is presented in Figure 1. The basic idea is to have $\mathcal{F}_{\text{SIG}}$ provide a "registry service" where a distinguished party (the signer, P_i in the figure) can register (*message,signature*) pairs. Any party that provides the right verification key can check whether a given pair is registered. Formalizing this idea in a way that is not over-restrictive and allows for natural realization (but still guarantees the intuitive requirements from "ideal signature schemes") requires some care.

The formulation here differs from the formulations in [C01, CK02, CR03] in two main respects. First, the formulation here lets any party generate public keys and signatures (still, only the first public key to be generated, and the signatures generated with respect to this public key, are registered); in contrast, the previous formulations *ignored* all registration requests except for the first one. Second, the formulation here explicitly allows a corrupted signer to assume any signature as its own; in contrast, the previous formulations forced the verification procedure to reject messages that were never signed, even if the signer is corrupted. Below we highlight and motivate these points, as well as other the choices taken in the formulation of $\mathcal{F}_{\text{SIG}}$.[2]

Making the verification key part of the interface. Functionality $\mathcal{F}_{\text{SIG}}$ provides the signing party with a public verification key. This key has to be later presented by the verifier at verification time. Furthermore, if the verification key presented by the verifier is not the one provided by $\mathcal{F}_{\text{SIG}}$ then there are no guarantees regarding the outcome of the verification process. This modeling captures the fact that the basic functionality of a signature scheme only binds messages and signatures to verification keys, rather than 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.

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 in an obvious way on the signed message (or on all the messages signed so far). An alternative (and considerably more restrictive) formulation would require that signatures have some predetermined distribution. Such a requirement 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.

Disclosing the signer identity. Note that $\mathcal{F}_{\text{SIG}}$ discloses the identity of the signer to the adversary. Indeed, there is nothing in the basic intuitive requirements from signature schemes that requires that the identity of the signer remains secret. (See Appendix A for a formulation that guarantees this privacy requirement.

Allowing public modification of signatures. When presented with a verification request for a pair (m, σ), where m was legitimately signed but with a different signature σ', $\mathcal{F}_{\text{SIG}}$ lets the adversary decide whether the verification should succeed. This reflects the fact that the basic

[2] A third point of difference from previous formalizations has to do with the formalization of probabilistic polynomial time interactive Turing machines (PPT ITMs). Since this technical point is general to the framework, we describe it together with the review of the framework in Appendix B.

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

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

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party P_i, hand $(\texttt{KeyGen}, sid)$ to the adversary. Upon receiving $(\texttt{Verification Key}, sid, v)$ from the adversary, send $(\texttt{Verification Key}, sid, v)$ to P_i, and request the adversary to deliver this message immediately. In addition, if this is the first activation then record the pair (P_i, v); otherwise the generated verification key v is discarded.

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from some P_j, hand $(\texttt{Sign}, sid, P_j, m)$ to the adversary. Upon receiving $(\texttt{Signature}, sid, P_j, m, \sigma)$ from the adversary, set $s_m = \sigma$, send $(\texttt{Signature}, sid, m, \sigma)$ to P_j, and request the adversary to deliver this message immediately. In addition, if $P_j = P_i$ then record the pair (m, s_m); otherwise do nothing.

Signature Verification: Upon receiving a value $(\texttt{verify}, sid, m, \sigma, v')$ from some party P_j, first determine the value of the verification bit f:

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

Once the value of f is set, send $(\texttt{verified}, id, m, f)$ to P_j, and request the adversary to deliver this message immediately.

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

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 of any pair (m, σ) that is not explicitly recorded. This would exclude signature schemes that allow generating new signatures from old ones. This stronger requirement is considered in e.g. [GO92].

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. (This more restricted variant makes sense only if public modification of signatures is prohibited, as described in the previous paragraph. Indeed, this variant also has the flavor of verifiable random functions.)

Allowing Corrupted signers to claim any signature as their own. If the signer is corrupted, and $\mathcal{F}_{\text{SIG}}$ is asked to verify a signature σ on a message m and public-key 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 claiming any signature to be its own. 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 on any message as "its own".[3]

We remark that 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 on 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.

Immediate message delivery. Functionality $\mathcal{F}_{\text{SIG}}$ asks the adversary to deliver the output values to the (dummy) parties *immediately.* This represents the fact that typically in signature schemes we expect the key generation, signature, and verification algorithms to run locally and provide output immediately, without waiting for any incoming messages from other parties. Of course, if any of these three activities is realized by a distributed protocol then $\mathcal{F}_{\text{SIG}}$ would have to be relaxed accordingly.

Dealing with multiple signers. $\mathcal{F}_{\text{SIG}}$ distinguishes a single party (the first to activate $\mathcal{F}_{\text{SIG}}$ with a $(\texttt{KeyGen}, sid)$ message) as the legitimate signer; That is, only signature generation requests made by this party are recorded. Yet, $\mathcal{F}_{\text{SIG}}$ allows any party to generate verification keys and to obtain signatures for messages. These verification keys and associated signatures carry no security guarantee, since the verification process allows the adversary to decide on whether such "signatures" verify in case that the verification key in the verification request differs from the recorded verification key.

It may appear at first glance that this additional provision is redundant, and that $\mathcal{F}_{\text{SIG}}$ can safely ignore requests for key and signature generation by parties other than the legitimate signer. (Indeed, the formalizations of $\mathcal{F}_{\text{SIG}}$ in [C01, CK02, CR03] did not have this provision.) However, without such a provision, $\mathcal{F}_{\text{SIG}}$ would be unrealizable by "natural" signature protocols, or more specifically by protocols where the key generation and signature activities are performed locally by the signer without any interaction, for the following "technical" reason. In a multiparty setting, if different parties locally activate the key generation and signature modules, independently of each other, then they would all obtain legitimate verification keys and signatures. This of course holds even if all parties happen to use the same session id. Thus, had $\mathcal{F}_{\text{SIG}}$ ignored the key and signature generation requests made by parties other than

[3] 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.

the signer, then an environment could have easily distinguished between the real execution and the ideal one.

We remark that this seemingly "technical" aspect of the formulation of $\mathcal{F}_{\text{SIG}}$ in fact highlights a real concern regarding the use of signature schemes: It is pointless to use signature schemes without being certain about the validity of the verification key. Indeed, if the verification process is given a verification key that is not the valid one then there is no guarantee of security. Furthermore, the verification process cannot figure out by itself whether the given verification key is valid; this information has to be obtained by means external to the protocol.

2.3 Equivalence with the [GMRi88] notion of security

Recall that [C01] asserts that realizing $\mathcal{F}_{\text{SIG}}$ is equivalent to resilience against existential forgery by 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, the proof there 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.)

First, we describe how to translate a signature scheme $S = (gen, sig, ver)$ as in [GMRi88] into a protocol π_S in the present setting. This is done as follows: When P_i, running π_S, receives an input `(KeyGen,`sid`)`, it executes algorithm gen, keeps the signing key s and outputs the verification key v. When the signer receives an input `(Sign,`sid, m`)`, it sets $\sigma = sig(s, m)$ and outputs `(Signature,`sid, m, σ`)`. When a party gets an input `(verify,`sid, m, σ, v'`)`, it outputs `(verified,`$sid, m, ver(v', m, \sigma)$`)`.

Claim 1 *Let $S = (gen, sig, ver)$ be a signature scheme as in* [GMRi88]. *Then π_S securely realizes $\mathcal{F}_{\text{SIG}}$ if and only if S is existentially unforgeable against chosen message attacks.*

Proof: For the "only if" direction, let $S = (gen, sig, ver)$ be a signature scheme, and assume that π_S securely realizes F_{SIG}. We show that S is existentially unforgeable against chosen message attacks. That is, assume that there is a [GMRi88] forger G against S. We construct an environment $\mathcal{Z}$ and an real-life adversary $\mathcal{A}$ such that, for any ideal-process adversary $\mathcal{S}$, environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{F}_{\text{SIG}}$ and $\mathcal{S}$ in the ideal process, or with π_S and $\mathcal{A}$ in the real-life model.

Environment $\mathcal{Z}$ proceeds as follows. It first activates some uncorrupted party P_i with input `(KeyGen,`sid`)` for some value of sid (say, $sid = 0$), and forwards the returned key v to $\mathcal{A}$. From now on, whenever $\mathcal{A}$ asks $\mathcal{Z}$ to sign a message m, $\mathcal{Z}$ activates the signer with input `(Sign,`sid, m`)`, and reports the output to $\mathcal{A}$. When $\mathcal{A}$ asks $\mathcal{Z}$ to verify 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 uncorrupted party with input `(verify,`sid, m, σ, v`)` and outputs whatever that party outputs.

The real-life adversary $\mathcal{A}$ first waits to hear some party P_i send a public verification key v. (This will happen when $\mathcal{Z}$ activates P_i to be the signer.) Then, $\mathcal{A}$ runs G on input v. Whenever G generates a message m to be signed, $\mathcal{A}$ asks $\mathcal{Z}$ to sign m. When $\mathcal{A}$ obtains the signature σ from $\mathcal{Z}$, it hands σ to G. When G outputs a pair (m^*, σ^*) (supposedly a new message and its forged signature), $\mathcal{A}$ asks $\mathcal{Z}$ to verify (m^*, s^*).

It can be seen that whenever G succeeds (i.e., whenever $ver(m^*, \sigma^*) = 1$ and m^* was not previously signed by P_i), $\mathcal{Z}$ outputs 1. Thus, under the assumption that G succeeds with non-

negligible probability, in the real-life model $\mathcal{Z}$ outputs 1 with non-negligible probability. However, in the ideal process with $\mathcal{F}_{\text{SIG}}$ $\mathcal{Z}$ never outputs 1, regardless of what the ideal-process adversary does.

For the "if" direction, assume that 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_S and $\mathcal{A}$ in the real-life model. We construct a [GMRi88] forger G against S.

Let us first consider the following ideal-process adversary (simulator), $\mathcal{S}$. Simulator $\mathcal{S}$ runs a simulated copy of $\mathcal{A}$. In addition:

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 $(\texttt{KeyGen}, sid, P_i)$ from $\mathcal{F}_{\text{SIG}}$, it proceeds as follows. If this is not the first request from P_i then $\mathcal{S}$ ignores this request. Otherwise, $\mathcal{S}$ runs the key generation algorithm gen, obtains a pair (s, v) of keys, returns $(\texttt{Verification Key}, sid, v)$ to $\mathcal{Z}$, and records (P_i, s_i, v_i).

3. Whenever $\mathcal{S}$ receives a message $(\texttt{Sign}, sid, P_i, m)$ from $\mathcal{F}_{\text{SIG}}$, it looks for a recorded triple (P_i, s_i, v_i). If such triple is found, then $\mathcal{S}$ computes $\sigma = sig(s_i, m)$, records (P_i, m, σ), and hands $(\texttt{Signature}, sid, m, \sigma)$ back to $\mathcal{F}_{\text{SIG}}$. Otherwise, it does nothing.

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

5. When $\mathcal{A}$ corrupts some P_i, $\mathcal{S}$ corrupts P_i in the ideal process. If P_i is the signer, then $\mathcal{S}$ reveals the signing key s as the internal state of P_i.

Let B denote the event that, in a run of π_S, $ver(v_i, m, \sigma) = 1$ for some message m and signature σ, but P_i is the legitimate signer (i.e., P_i was the first to be activated with input $(\texttt{KeyGen}, sid)$), is uncorrupted, and never signed m during the execution of the protocol. Observe that, as long as event B does not occur, $\mathcal{Z}$'s view of an interaction with $\mathcal{A}$ and parties running the protocol is statistically close to its view of an interaction with $\mathcal{A}$ and $\mathcal{F}_{\text{SIG}}$ in the ideal process.[4] However, we are guaranteed that there exist a real-life adversary $\mathcal{A}$ and environment $\mathcal{Z}$ that distinguishes between the two interactions with non-negligible probability. Thus, we are guaranteed that in the real-life model event B occurs with non-negligible probability.

We turn to constructing the [GMRi88] forger G. G runs a simulated copy of $\mathcal{Z}$, and simulates an interaction of $\mathcal{Z}$ with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\text{SIG}}$ (where G plays the role of S). This is done as follows. 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_i, v_i), G hands $\mathcal{A}$ the public verification key v in its input. Instead of running the signing algorithm to obtain $\sigma = sig(s_i, m)$, G asks its oracle to sign m and obtains the signature σ. Whenever the simulated $\mathcal{Z}$ activates some uncorrupted party with input $(\texttt{Verify}, sid, m, \sigma, v)$, G checks whether m was never signed before and $ver(v, m, \sigma) = 1$. Once such a pair (m, σ) is found, G outputs that pair and halts. (If $\mathcal{A}$ asks to corrupt the signer than G halts with a failure output.)

[4] The views may differ if in the run of π_S a message m was signed by P_i with a legitimately generated signature σ, and the verification algorithm rejects (v_i, m, σ). But this event has negligible probability when S is a valid signature scheme.

Notice that, from the point of view of $\mathcal{A}$ and $\mathcal{Z}$, the interaction with G looks the same as an interaction in the real-life model with π_S. Thus, we are guaranteed that event B (and, thus, successful forgery by G) will occur with non-negligible probability. □

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 a protocol π securely realizes $\mathcal{F}_{\text{SIG}}$ against adaptive adversaries if and only if π securely realizes $\mathcal{F}_{\text{SIG}}$ against non-adaptive adversaries. Also, it is interesting to note that the proof of Claim 1 does not involve any data erasures.

3 Using signatures to provide certification

This section studies an abstraction of signature schemes that is geared towards the task of binding documents (or,messages) 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 and motivates 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 on a message and the identity of the signer. (In contrast, $\mathcal{F}_{\text{SIG}}$ provides binding only to a 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. Some of the choices regarding the formulation of $\mathcal{F}_{\text{CERT}}$ are highlighted and motivated below. In addition, most of the discussion regarding the formulation of $\mathcal{F}_{\text{SIG}}$ is relevant here as well.

Encoding the signer identity in the session ID. Functionality $\mathcal{F}_{\text{CERT}}$ does not involve a verification key in the interface with the environment. Instead, it provides ideal and direct binding between the signed messages and the present copy of F_{CERT}. This binding is provided via the session ID. Indeed, the session ID is assumed to include the identity of the signer, thus any verifier can read the identity of the signer off of the session ID. (Said otherwise, if the verifier associates the signature with a different signer identity, then it would use a different copy of $\mathcal{F}_{\text{SIG}}$.)[5]

[5]We remark that it is not essential to encode the identity of the signer within the session ID: An alternative, equivalent formulation would carry the signer's identity as an additional parameter; the functionality would then

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

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

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

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

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

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

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

Non-immediate message delivery. Functionality $\mathcal{F}_{\text{CERT}}$ does not require immediate message delivery, for either the signature generation or verification processes. Furthermore, the adversary is notified whenever some party makes a verification request. This reflects the fact that the need for binding the signature to a party ID implies that some interaction is needed for message generation and verification. Indeed, in protocol CAS below for realizing $\mathcal{F}_{\text{CERT}}$ given $\mathcal{F}_{\text{SIG}}$ the signature generation and the verification procedures are sometime interactive. It is possible to strengthen $\mathcal{F}_{\text{CERT}}$ to allow interaction only periodically, or in, say, the first verification by each party.

Dealing with revocations. Functionality $\mathcal{F}_{\text{CERT}}$ does not deal with identity revocations. Indeed, modeling revocations requires more involved structures that include a trusted revocation entity. We leave it out of scope.

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. (It should be remarked that there of course exist methods for realizing $\mathcal{F}_{\text{CERT}}$ without using $\mathcal{F}_{\text{CA}}$, for instance using direct out-of-band exchange of public verification keys. Still, as shown in Claim 4, $\mathcal{F}_{\text{CERT}}$ cannot be realized in the bare unauthenticated model; *some* set-up assumption is necessary.)

ignore signing requests by parties other than the legitimate signer, and would reject verification requests that do not have the correct signer identity in them. The reason to encode the signer's ID within the session ID is to simplify the formulation and use of $\mathcal{F}_{\text{CERT}}$.

The certificate authority functionality. The ideal certificate authority functionality, $\mathcal{F}_{\text{CA}}$, is presented in Figure 3. As in functionality $\mathcal{F}_{\text{CERT}}$, each copy of$\mathcal{F}_{\text{CA}}$ is bound to a single party identity; for ease of presentation we unify the party identity with the session identity of the functionality. $\mathcal{F}_{\text{CA}}$ 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 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 $\mathcal{F}_{\text{AUTH}}$.

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

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

1. Upon receiving the first message $(\texttt{register}, sid, v)$ from party P_i, send $(\texttt{registered}, sid, v)$ to the adversary; upon receiving `ok` from the adversary, and if $id = P_i$ and this is the first request from P_i, then record the pair (P_i, v).
2. Upon receiving a message $(\texttt{retrieve}, sid)$ from party P_j, send $(\texttt{retrieve}, sid, P_j)$ to the adversary. If there is a recorded pair (sid, v) then send $(\texttt{retrieve}, sid, v)$ to P_j. Else return $(\texttt{retrieve}, sid, \perp)$.

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

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.

Claim 2 *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 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 signature generation. When $\mathcal{S}$ receives in the ideal process a message $(\texttt{Sign}, sid, m)$ from $\mathcal{F}_{\text{CERT}}$, where $sid = (P_i, s)$ and P_i is uncorrupted, it proceeds as follows:

1. If this is the first time that P_i 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 $(\texttt{KeyGen}, sid)$, obtain the response $(\texttt{Verification Key}, sid, v)$ from $\mathcal{A}$, and send to $\mathcal{A}$ the message $(\texttt{registered}, sid, v)$ from $\mathcal{F}_{\text{CA}}$. When $\mathcal{A}$ sends `ok` to $\mathcal{F}_{\text{CA}}$, mark the pair (sid, 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 $(\texttt{Sign}, sid, m)$, forward the response $(\texttt{Signature}, sid, m, \sigma)$ to $\mathcal{F}_{\text{CERT}}$.

Protocol CAS

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

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

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

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

Simulating the interaction of a corrupted signer proceeds as follows. If $\mathcal{A}$ instructs a corrupted P_i to send $(\texttt{KeyGen}, sid)$ 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 $sid \neq (P_i, s)$ for some s, then ignore this instruction. Else, send to $\mathcal{A}$ (in the name of $\mathcal{F}_{\text{SIG}}$) the message $(\texttt{KeyGen}, sid)$; when $\mathcal{A}$ responds with $(\texttt{Verification Key}, sid, v)$, send $(\texttt{Verification Key}, sid, v)$ to P_i. 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_i sends (sid, v') to $\mathcal{F}_{\text{CA}}$, send to $\mathcal{A}$ the message $(\texttt{registered}, sid, v')$ from $\mathcal{F}_{\text{CA}}$. When $\mathcal{A}$ sends $\texttt{ok}$ to $\mathcal{F}_{\text{CA}}$, then mark the pair (sid, 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_j made a verification request, proceed as follows.

1. If this is the first verification request made by P_j, then simulate for $\mathcal{A}$ the exchange between P_j and $\mathcal{F}_{\text{CA}}$. That is, simulate for $\mathcal{A}$ a message $(\texttt{retrieve}, sid, P_j)$ coming from $\mathcal{F}_{\text{CA}}$. If $\mathcal{F}_{\text{CA}}$ has a pair (sid, v) recorded then simulate a response (sid, v) from $\mathcal{F}_{\text{CA}}$ to P_j. Otherwise, simulate a response $(sid, \perp)$ from $\mathcal{F}_{\text{CA}}$ to P_j.
2. If a message $(\texttt{Verify}, sid, P_j, m, \sigma)$ arrives from $\mathcal{F}_{\text{SIG}}$, 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_j is corrupted then the simulation is modified in the natural way. That is, when P_j sends a message $(\texttt{Verify}, sid, m, \sigma, v)$ 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 Using $\mathcal{F}_{\text{CERT}}$ to obtain authenticated communication

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].)

We complete this section by proving a complementary claim: $\mathcal{F}_{\text{AUTH}}$ cannot be realized in the bare unauthenticated model. This claim formalize then 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.

We first recall $\mathcal{F}_{\text{AUTH}}$ in Figure 5. 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 $\mathcal{F}_{\text{AUTH}}$

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

1. Upon receiving a message $(\texttt{send}, sid, P_j, m)$ from party P_i, send $(\texttt{sent}, sid, P_i, P_j, m)$ to P_j and to the adversary, and 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 P_j, with session identifier sid, party P_i simply signs (m, P_j) and sends the signed message to P_j. A more complete description appears in Figure 6.

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. 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

Protocol SBA

1. Upon receiving an input $(\texttt{send}, sid, P_j, m)$, party P_i sets $sid' = (P_i, sid)$, sets $m' = (m, P_j)$, sends $(\texttt{Sign}, sid', m')$ to $\mathcal{F}_{\text{CERT}}$, obtains the response $(\texttt{Signed}, sid', m', s)$, and sends (sid, P_i, m, s) to P_j.
2. Upon receiving (sid, P_i, m, s), P_j sets $sid' = (P_i, sid)$, sets $m' = (m, P_j)$, sends $(\texttt{Verify}, sid', m', s)$ to $\mathcal{F}_{\text{CERT}}$, and obtains a response $(\texttt{Verified}, sid', m', s, f)$. If $f = 1$ then P_j outputs $(\texttt{sent}, sid, P_i, P_j, m)$ and halts. If $f = 0$ then P_j halts with no output.

Figure 6: The signature-based authentication protocol, SBA

receiver P_j 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 [BLR03] 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 method for guaranteeing uniqueness of the sid 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 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 3 *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 $\mathcal{Z}$ of 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 P_i is activated with input $(\texttt{send}, sid, P_j, m)$, $\mathcal{S}$ 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 $(\texttt{Sign}, (P_i, sid), (m, P_j))$ from $\mathcal{F}_{\text{CERT}}$, and obtains a value s from $\mathcal{A}$. Next, $\mathcal{S}$ hands $\mathcal{A}$ the message (sid, P_i, m, s) sent from P_i to P_j.

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 P_i 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 P_i in the namer of $\mathcal{F}_{\text{CERT}}$.

Simulating the recipient. When $\mathcal{A}$ delivers a message (sid, P_i, m, s) to an uncorrupted party P_j, $\mathcal{S}$ 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 $(\texttt{Verify}, sid' = (P_i, sid), m' = (m, P_j), s)$ to $\mathcal{A}$ (that is, if P_i 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 $(\texttt{Verified}, sid', m', s, f = 1)$ to P_j (that is, if $\mathcal{A}$ responded with $f = 1$ or the message m' was recorded with signature s) then deliver the message $(\texttt{received}sid, P_i, P_j, m)$ which was sent in the ideal process from $\mathcal{F}_{\text{AUTH}}$ to P_j. 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 $(\texttt{Verified}, sid', m', s, f = 1)$ from $\mathcal{F}_{\text{CERT}}$ for an incoming message (sid, P_i, m, s), while P_i is uncorrupted and never sent the message (sid, P_i, m, s). However, if P_i never sent (sid, P_i, m, s), then the message $m' = (m, P_j)$ was never signed by the copy of$\mathcal{F}_{\text{CERT}}$ with session id (P_i, sid); thus, according to the logic to $\mathcal{F}_{\text{CERT}}$, P_j would always obtain $(\texttt{Verified}, sid', m', s, f = 0)$ 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 protocols that realize $\mathcal{F}_{\text{CERT}}$ in the plain model. More precisely, say that a multiparty protocol is *terminating* if, whenever the adversary corrupts no party and delivers all messages unmodified and with no delay[6], then at least one party generates output with non-negligible probability. We show:

Claim 4 *There exist no terminating 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 P_1, P_2. We construct the following environment $\mathcal{Z}$ and real-life adversary $\mathcal{A}$. $\mathcal{Z}$ activates no party with any input. If party P_2 generates output $(\texttt{received}, sid = 0, P_1, m = 0)$, then $\mathcal{Z}$

[6]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.

outputs 1. Otherwise $\mathcal{Z}$ outputs 0.[7] Adversary $\mathcal{A}$ simulates for P_2 an execution of π on input $(\texttt{send}, sid = 0, P_2, m = 0)$ for P_1, where no party is corrupted, and all messages are delivered without delay. Note that $\mathcal{A}$ can do so successfully, since it can feed P_2 with any incoming messages, and P_2 shares no prior state with any other party.

Since protocol π is terminating, with non-negligible probability P_2 outputs $(\texttt{received}, 0, P_1, 0)$ in the real execution of π. However, in the ideal process P_2 never generates any output. □

Corollary 5 *There exist no terminating 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 4 and 3. 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 4), 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 Ivan Damgaard, Yehuda Lindell, Phil Mackenzie, Jesper Nielsen and Ke Yang.

References

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

[BP03] Michael Backes and Birgit Pfitzmann. A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. *Cryptology ePrint Archive, Report 2003/121*, 2003, http://eprint.iacr.org/.

[BPW03a] M. Backes, B. Pfitzmann and M. Waidner, "A Universally Composable Cryptographic Library," http://eprint.iacr.org/2003/015.

[BPW03b] Michael Backes and Birgit Pfitzmann and Michael Waidner. Symmetric Authentication Within a Simulatable Cryptographic Library. *Cryptology ePrint Archive, Report 2003/145,* 2003, http://eprint.iacr.org/2003/145.

[7] This instruction seems hard to implement in an asynchronous network, since $\mathcal{Z}$ cannot wait "until P_2 generates output". We thus interpret this instruction as follows: $\mathcal{Z}$ first writes 0 on its output tape. Next, if P_2 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.

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

[BLR03] B. Barak, Y. Lindell and T. Rabin, "A Note on Secure Protocol Initialization and Setup in Concurrent Settings," manuscript, 2003.

[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.

[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.

[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.

[DOW92] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[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 Symposium on Theory of Computing (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] Dennis Hofheinz, Joern Mueller-Quade, and Rainer Steinwandt, personal communication, March 2003.

[IPSEC98] D. Harkins and D. Carrel, ed., "The Internet Key Exchange (IKE)", *RFC 2409,* Nov. 1998.

[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.

[PW00] B. Pfitzmann and M. Waidner, "Composition and integrity preservation of secure reactive systems", *7th ACM Conf. on Computer and Communication Security*, 2000, pp. 245-254.

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

A Guaranteeing privacy of signatures

As mentioned in the Introduction, allowing the adversary to determine the signature values has the consequence that the adversary learns the values of all the messages signed and signatures generated even if the signer never disclosed these values to anyone. This formulation is in accord with the basic security requirements of digital signatures (as in, say, [GMRa89]), which do not make any secrecy requirements from signatures or from the signature generation process. Still, it does not capture the fact that, when the signature generation process is local within the signer, the signature value remains secret until explicitly made public by the signer. Consequently, functionality $\mathcal{F}_{\text{SIG}}$ may become inadequate in applications where the signer wants to disclose the signature or the signed message only to certain parties.

This appendix describes an alternative formulation of $\mathcal{F}_{\text{SIG}}$, denoted $\mathcal{F}_{\text{PRIV-SIG}}$ (for "private signatures"), that does not disclose the values of the signatures or the signed messages to the adversary. In fact, the adversary does not even learn whether signatures were generated at all. Similarly, the adversary does not learn which signatures were verified by honest parties, and of course it does not learn the outcomes of these verifications.

Functionality $\mathcal{F}_{\text{PRIV-SIG}}$ is identical to $\mathcal{F}_{\text{SIG}}$ with the following exceptions. When providing $\mathcal{F}_{\text{PRIV-SIG}}$ with the verification key v, the adversary hands $\mathcal{F}_{\text{PRIV-SIG}}$ 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, $\mathcal{F}_{\text{SIG}}$ keeps a running instance of G and generates each signature by running M on the signed message. Similarly, $\mathcal{F}_{\text{PRIV-SIG}}$ keeps a running copy of machine V. Whenever $\mathcal{F}_{\text{SIG}}$ asks $\mathcal{S}$ for the value of f, $\mathcal{F}_{\text{PRIV-SIG}}$ invokes machine V on the message and signature, and adopts the answer of V.[8]

Functionality $\mathcal{F}_{\text{PRIV-SIG}}$ is presented in Figure 7. All the discussion pertaining to functionality $\mathcal{F}_{\text{SIG}}$ is relevant here as well. In addition, as discussed above, $\mathcal{F}_{\text{PRIV-SIG}}$ guarantees that the adversary does not learn anything about which messages are signed or verified, or on the values of the signatures. We remark that the technique of having the ideal-process adversary supply the functionality with descriptions of ITMs to run was first used in [CK02] in the context of key-exchange protocols. Indeed, this technique is quite general and powerful. In particular, it allows capturing, within a simulation-based definitional framework, requirements such as "any protocol is allowed as long as the protocol satisfies a given property P," where property P need not necessarily be defined in a

[8] We assume that both G and V are (unbounded) 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 maximum between the security parameter and the length of the message m.

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

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

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party P_i, hand $(\texttt{KeyGen}, sid)$ to the adversary. Upon receiving $(\texttt{Verification Key}, sid, v, G, V)$ from the adversary, send $(\texttt{Verification Key}, sid, v)$ to P_i, and request the adversary to deliver this message immediately. Interpret G, V as descriptions of two polytime ITMs, and invoke a copy of G and a copy of V.

In addition, if this is the first activation then record the pair (P_i, v); otherwise the generated verification key v is discarded.

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from some P_j, let $s_m = G(m)$, send $(\texttt{Signature}, sid, m, s_m)$ to P_j, and request the adversary to deliver this message immediately. In addition, if $P_j = P_i$ then record the pair (m, s_m).

Signature Verification: Upon receiving a value $(\texttt{verify}, sid, m, \sigma, v')$ from some party P_j, first determine the value of the verification bit f:

1. If $v' = v$ (i.e., if the verification key in the verification request equals the recorded verification key) and the pair (m, σ) is recorded then set $f = 1$.
2. If $v' = v$, the signer is not corrupted, and no pair (m, σ') for any σ' is recorded (i.e., m was never before singed) then set $f = 0$.
3. In all other cases, let $f = V(v', m, \sigma)$.

Once the value of f is set, send $(\texttt{verified}, id, m, f)$ to P_j, and request the adversary to deliver this message immediately.

Figure 7: The privacy preserving signature functionality, $\mathcal{F}_{\text{PRIV-SIG}}$.

simulation-based way. Here the property P is essentially "unconditional unforgeability" cast in a multiparty protocol setting.

An analogous claim to Claim 1 holds also with respect to $\mathcal{F}_{\text{PRIV-SIG}}$. That is, recall the transformation from a signature scheme S into a protocol $\pi_{\mathcal{S}}$. Then, we have:

Claim 6 *Let $S = (gen, sig, ver)$ be a signature scheme as in* [GMRi88]. *Then π_S securely realizes $\mathcal{F}_{\text{PRIV-SIG}}$ if and only if S is existentially unforgeable against chosen message attacks.*

Proof (sketch): The proof is very similar to the proof of Claim 1. The main difference is that here the simulator $\mathcal{S}$ has to provide $\mathcal{F}_{\text{PRIV-SIG}}$ with the algorithms G and V. This is done as follows: Algorithm G is sig_{sk} (i.e., the signing algorithm of S with the secret key generated by $\mathcal{S}$ together with the public key given to $\mathcal{F}_{\text{PRIV-SIG}}$.) Algorithm V is ver_{vk}, the verification algorithm of S. It can be easily seen that the simulation remains valid. We omit further details. □

A conclusion from Claims 1 and 6 is that for any signature scheme S, the protocol π_S securely realizes $\mathcal{F}_{\text{SIG}}$ if and only if it securely realizes $\mathcal{F}_{\text{PRIV-SIG}}$. We remark that this equivalence of course does *not* hold for general protocols that realize $\mathcal{F}_{\text{SIG}}$. In fact, given any protocol that realizes $\mathcal{F}_{\text{SIG}}$, it is easy to construct a protocol that realizes $\mathcal{F}_{\text{SIG}}$ and does not realize $\mathcal{F}_{\text{PRIV-SIG}}$ (say, by having the signer publicize the value of each message signed and the corresponding signature).

B 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 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, or an "advice". From a complexity-theoretic point of view, this essentially implies that adversaries are non-uniform ITMs.

We assume that all ITMs run in probabilistic polynomial time (PPT). In fact, we formulate two different notions of PPT ITMs. Say that an ITM is strict PPT if there exists a constant c such that: (a) M completes each activation within n^c steps, where n is the maximum between the security parameter k (which is written on a special tape) and the length of the input to this activation, and (b) the machine halts altogether after at most k^c steps. If only the first condition is guaranteed then the ITM is unbounded PPT (UPPT). (Jumping ahead, we assume that all the adversarial entities considered, namely the environment and all adversaries, are strict PPT. The parties running the protocols and the ideal functionalities are only assumed to be UPPT.)[9]

B.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.

[9] We remark that the formalization in [C01] requires that all entities, including the parties running the protocol and the ideal functionalities, are strict PPT ITMs. This 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, this notion of PPT ITMs is too restrictive for capturing the notion of "polynomial time protocols" for reactive 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 strict PPT ITM, and cannot be realized by strict PPT ITMs. We thank Dennis Hofheinz, Joern Mueller-Quade, and Rainer Steinwandt for pointing this issue to us in [HMS03].

The presentation below concentrates on the following model, aimed at representing current realistic communication networks (such as the Internet). The network is *asynchronous* without guaranteed delivery of messages. The communication is public and unauthenticated. (That is, the adversary may delete, modify, and generate messages at wish.) Parties may be broken into (i.e., become corrupted) throughout the computation, and once corrupted their behavior is arbitrary (or, *Byzantine*). Finally, all the involved entities are restricted to probabilistic polynomial time (or, "feasible") computation. The framework can be adapted in natural ways to present other models of computation (e.g., synchronous or authenticated communication). See more details in [C01].

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

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

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

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

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

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

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

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

Once the adversary $\mathcal{S}$ is activated, it may read its own input tape and in addition it can read the *destinations* of the messages on the outgoing communication tape of $\mathcal{F}$. That is, $\mathcal{S}$ can see the identity of the recipient of each message sent by $\mathcal{F}$, but it cannot see the *contents* of this message (unless the recipient of the message is $\mathcal{S}$). $\mathcal{S}$ may either deliver a message from $\mathcal{F}$ to some party by having this message copied the party's incoming communication tape or corrupt a party. Upon corrupting a party, the adversary learns whatever is specified by the functionality.[10] In addition, from the time of corruption on, the adversary controls the party's actions. Also, both $\mathcal{Z}$ and $\mathcal{F}$ are notified that the party is corrupted. If the adversary delivered a message to some uncorrupted (dummy) party in an activation then this party is activated once the activation of the adversary is complete. Otherwise the environment is activated next.

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

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

Securely realizing an ideal functionality. We say that a protocol 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 $\mathcal{F}$. (In a way, $\mathcal{Z}$ serves as an "interactive distinguisher" between the two processes. Here it is important that $\mathcal{Z}$ can provide the process in question with *adaptively chosen* inputs throughout the computation.) A distribution ensemble is called binary if it consists of distributions over $\{0, 1\}$. We have:

Definition 7 *Two* binary *distribution ensembles X and Y are* indistinguishable *(written $X \approx Y$) if*

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

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:*

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

B.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 $\mathcal{F}$. Each copy of $\mathcal{F}$ is identified via a unique session identifier (SID); all messages addressed to this copy and all message sent by this copy carry the corresponding SID.

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

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

Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ 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 $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$. (We stress that here π is a hybrid of a real-life protocol with ideal evaluation calls to $\mathcal{F}$.) Let $\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},Z}$ denote the distribution ensemble $\{\text{HYB}^{\mathcal{F}}_{\pi,\mathcal{A},Z}\}_{k\in\mathbf{N},z\in\{0,1\}^*}$.

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 $\mathcal{H}$ there exists an adversary $\mathcal{H}'$ in the $\mathcal{F}$-hybrid model such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{H}$ and π^ρ in the $\mathcal{G}$-hybrid model or it is interacting with $\mathcal{H}'$ and π in the $\mathcal{F}$-hybrid model.

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.(Here one has to define what it means to securely realize functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model. This is done in the natural way.) We first formalize the notion of protocol emulation:

Definition 9 (Protocol emulation) *Let $\mathcal{F}_1$, $\mathcal{F}_2$ be ideal functionalities and let π_1, π_2 be multi-party protocols. We say that π_1 in the $\mathcal{F}_1$-hybrid model emulates π_2 in the $\mathcal{F}_2$-hybrid model if for any adversary $\mathcal{A}_1$ there exists an adversary $\mathcal{A}_2$ such that for any environment machine $\mathcal{Z}$ we have*

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

The case where π_1 (resp., π_2) runs in the real-life model of computation is captured by setting $\mathcal{F}_1$ (resp., $\mathcal{F}_2$) to be the functionality that does nothing. Note that emulation is transitive. That is, if π_1 in the $\mathcal{F}_1$-hybrid model emulates π_2 in the $\mathcal{F}_2$-hybrid model, and π_2 in the $\mathcal{F}_2$-hybrid model emulates π_3 in the $\mathcal{F}_3$-hybrid model, then π_1 in the $\mathcal{F}_1$-hybrid model emulates π_3 in the $\mathcal{F}_3$-hybrid model.

Theorem 10 (Universal composition [C01]**)** *Let $\mathcal{F}, G$ be ideal functionalities. Let π be an n-party protocol in the $\mathcal{F}$-hybrid model, and let ρ be an n-party protocol that securely realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model. Then protocol π^ρ in the $\mathcal{G}$-hybrid model emulates protocol π in the $\mathcal{F}$-hybrid model. In particular, if π securely realizes some ideal functionality $\mathcal{I}$ in the $\mathcal{F}$-hybrid model then π^ρ securely realizes $\mathcal{I}$ in the $\mathcal{G}$-hybrid model.*

C On the [BPW03a] modeling of signatures

Backes, Pfitzmann and Waidner [BPW03a] propose a "library" of idealized cryptographic primitives, within the framework of [PW00].[11] 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 critique.

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: [12]

[11] This framework has many similarities to the UC framework, although the formalization is somewhat different. In particular, the universal composition theorem of [C01] essentially holds also in that framework; see [BPW04] for details. (We remark that a limited version of this theorem was proven in [PW00].)

[12] We note that the description of the library interface in [BPW03a] is much more detailed. In fact, the level of detail

- 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 (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 secure and authenticated communication functions. That is:

- Signature transmission: When a party A that is registered as a "legitimate verifier" for some signature-handle sends the signature-handle to another party B via a secure or authenticated channel provided by the library, the library sends the signature-handle to B and *adds B to the list of "legitimate verifiers" for this signature-handle.*[13]

Critique. Before proceeding with our critique of the definition of the [BPW03a] library, we wish to re-iterate that, like any other definition of security, ideal functionalities can never be "mathematically wrong". Furthermore, there may of course be more than a single valid way to capture the security properties of a given informal primitive, and different formalizations may have complementary properties. Still we believe that the present formalization of the library has some shortcomings; here we attempt to bring those forth. We start with some specific issues and then draw a more general conclusion.

First, notice that the above definition of the library forces the user of the library to use the secure/authenticated channels provided by the library to transmit signature handles. Indeed, it is impossible to use other secure channels mechanisms (or even secure channels that are provided by a different copy of the library), and maintain the ability to verify signatures. Furthermore, this means that any protocol that uses the signature module of the library must also assume that idealized secure/authenticated channels are already provided. In particular, it does not make sense to model protocols that try to *realize* secure/authenticated channels using signature schemes as protocols with ideal access to the library.

and the density of notation make it somewhat hard to follow. 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.

[13] We remark that 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 via the library then the library would not be realizable at all. See Section 2.1 for more discussion on these issues.

Another consequence of this formalization is that any implementation of the signature part of the library is incomplete unless it implements also the secure/authenticated channels interface. (We remark that the same holds also with respect to the encryption functions of the library, which are not reviewed here.)[14] Another consequence is that the library 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 observe that the present formalization of the library is inherently inadequate for realizing the "Dolev-Yao paradigm" as sketched in the introduction, at least with respect to the type of protocols which are traditionally analyzed using this paradigm. Indeed, recall that the paradigm starts with a concrete protocol that uses some cryptographic primitive (say, a signature scheme), 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 already the first step fails: We need to de-compose the given protocol into a module that realizes the library, plus a "high-level module" that uses the library. But most cryptographic protocols do not contain any module that realizes the entire library, and thus cannot be de-composed as needed. This criticism is of course relevant also to works that use the [BPW03a] library to perform "Dolev-Yao style analysis", such as [BP03, BPW03b].[15]

In conclusion, we point out 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. 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.

[14] In the [BPW03a] implementation of the library, the secure/authenticated channels functions are not actually implemented; rather they are assumed to be provided by the underlying network. A complete implementation of their library would of course have to realize also secure and authenticated channels.

[15] One can informally claim that it is "essentially enough" to find a module within the given protocol that realizes the "corresponding module in the library". Of course, this claim has to be made rigorous in order to make sense; Furthermore, it seems unlikely that such a claim can be made rigorous since. as demonstrated above, the functions of the library themselves are not well-separated.