Technical Report: Reactively Secure Signature Schemes (Long Version)*

Michael Backes, Birgit Pfitzmann, and Michael Waidner

IBM Research, Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mbc,bpf,wmi}@zurich.ibm.com

November 11, 2005

Abstract. Protocols for problems like Byzantine agreement, clock synchronization or contract signing often use digital signatures as the only cryptographic operation. Proofs of such protocols are frequently based on an idealizing "black-box" model of signatures. We show that the standard cryptographic security definition for digital signatures is not sufficient to ensure that such proofs are still valid if the idealized signatures are implemented with real, provably secure signatures.
We propose a definition of signature security suitable for general reactive, asynchronous environments, called *reactively secure signature schemes*, and prove that for signature schemes where signing just depends on a counter as state the standard security definition implies our definition.
We further propose an idealization of digital signatures which can be used in a reactive and composable fashion, and we show that reactively secure signature schemes constitute a secure implementation of our idealization.

1 Introduction

Protocols for problems like Byzantine agreement, clock synchronization or contract signing often use digital signatures as the only cryptographic operation. Proofs of such protocols typically use a "black-box" model of signatures: they just assume that signatures are unforgeable, i.e., they abstract from all cryptographic details like asymptotic security and error probabilities. Still one should expect that protocols proven secure in this abstract model are also secure if implemented with a real, cryptographically secure signature scheme.

Unfortunately this is not true: Consider an arbitrary signature scheme Sig that satisfies the standard GMR-definition of secure signature schemes [40]: A signature scheme is *secure* if no polynomial-time (in k, a security parameter) adversary can produce a forged signature with not negligible probability. The adversary has exclusive access to a signature oracle, and any signature under

* Parts of this work have been published at the *6th* and *7th Information Security Conference (ISC) [15, 18]*.

a message for which the oracle was not queried counts as a forgery. We now transform the scheme Sig into a new scheme Sig_{strange} which, when signing a message m, attaches to the signature on m all *previously* signed messages and their signatures. This is certainly a strange scheme, but it is easy to see that it is secure under the GMR-definition provided the original scheme Sig is secure, since if the adversary asks for the i-th signature, it has seen all signatures up to the $(i-1)$-st anyway; thus our new scheme is not easier to break than the original one.

Now consider a trivial protocol for fair contract signing: We have two potentially malicious parties A, B and a trusted third party T. Both main parties have inputs c, the contract, and binary values d_X, for $X = A, B$, which tell them whether they should sign ($d_X = 1$) or reject ($d_X = 0$) the contract. The contract should be signed only if each honest party X starts with $d_X = 1$. The protocol roughly works like this: Both A and B sign c, yielding signatures s_A and s_B. If $d_A = 0$ then A stops, and otherwise it sends s_A to T, and similarly B. If T receives both signatures it sends (s_A, s_B) back to A and B, and the contract is considered signed. If T does not receive both signatures (which in an asynchronous network might mean that T non-deterministically decides to terminate) then T stops and the contract is not signed, which means that nobody should get hold of the pair (s_A, s_B). Intuitively this protocol is secure, and one can even prove this in the black-box model. But clearly if the protocol is executed multiple times with our new signature scheme Sig_{strange} then from each successful run one can construct valid contracts for all previous runs, even for those that did not produce a valid contract.

In Section 2 we introduce a general security definition of digital signature schemes that resolves these problems and can be used in general asynchronous, reactive environments. The term "reactive" means that the system interacts with its users multiple times, e.g., in many concurrent protocol runs. In Sections 3 and 4 we show that for certain signature schemes security under the GMR-definition implies security under this definition. As our example has shown, this cannot be true in general; thus we limit ourselves to schemes where signing needs just a counter as state. This is sufficient for many provably secure signature schemes. For instance, in [40] the signer computes a tree and associates each signature with one of the leaves in this tree. Thus it is sufficient for the signing machine to keep track of which leaves are already used, and this can easily be encoded as a counter. Similar arguments apply, e.g., to the schemes in [57, 31, 32, 35].

In Section 5 we propose an idealization of digital signatures that can be used in the aforementioned black-box way, i.e., which, at a low level of abstraction, offers the functionality of a signature scheme in a reactive and composable fashion. More precisely, we show that reactively secure signature schemes constitute

a secure implementation of our idealization. By "low level" we mean that the interface of the idealization is not yet abstract in the sense needed for current automatic tools but still contains signatures as real cryptographic objects. This is in contrast to, e.g., Dolev-Yao-style models [34], where cryptographic objects are represented symbolically without being evaluated to bitstrings. For encryption, such low-level ideal functionalities were introduced in [43]. For signatures, formalizing and proving an ideal version is actually easier because their security property is an integrity property. It was known since [52, 53] that such properties can be formulated abstractly, e.g., in temporal logic. A similar formulation for authentication is known from [58], but without cryptographic proofs with respect to it. In essence, a low-level ideal system for signatures combines the real signature functionality with a system-internal verification whether the desired integrity property is still fulfilled. Such an idealization was first made in [44] for symmetric authentication. A somewhat similar ideal signature system was presented in [26], with variations in [29, 30]. In contrast to our abstraction, the precise approach taken in [26] cannot be used to construct nested abstract terms without revealing all signatures contained in these terms even if the signatures are appropriately encrypted, i.e., while an abstract term $E(pke, S(pks, m))$, denoting the encryption of a signature of a message m, in reality keeps m secret from the adversary even if sent over an insecure connection, its mere construction by an honest participant would give m to the adversary if one used the ideal functionality for signing from [26]. This aspect was not changed in [29, 30, 27].

In Section 6, we finally describe several variants of our idealization such as memory-less signature schemes and schemes restricted to fixed-length messages.

Further Related Work. Problems in reactive environments have already been identified for other cryptographic primitives, e.g., oblivious transfer [22] and public-key encryption [23].[1] For signatures, already the standard definition from [40] is in a reactive setting, but signatures are delivered to the adversary in the same order as they are generated. In a general asynchronous setting these orders may be different, which greatly complicates security proofs. In [56] signatures are used in an encrypted way within secure channels, but the specific usage avoids the general problems. Moreover, our work is closely related to the currently highly active line of research on computational soundness, see e.g., [1, 16, 9, 14, 20, 2, 10, 51, 12, 11, 4].

[1] More generally, achieving security under system composition is widely known to be difficult to achieve in general, see e.g., [46, 42, 47–49, 45, 33, 3, 17, 5, 18, 10].

2 Definitions and Notation

2.1 Notation

Let Σ denote a finite alphabet, and let Σ^+ denote the set of non-empty strings over Σ. We write ":=" for deterministic and "$\leftarrow$" for probabilistic assignment, and "$\xleftarrow{\mathcal{R}}$" for uniform random choice from a set. $\downarrow$ is a distinguished error element available as an addition to the domains and ranges of all functions and algorithms. A function $g\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ is called negligible iff for all positive polynomials Q, $\exists k_0 \forall k \geq k_0\colon\ g(k) \leq 1/Q(k)$.

The fundamental data structures in our definitions and proofs are arrays that store tuples of strings. For each array D, the tuples have a predefined structure, e.g., each tuple stores a message with a signature. To elegantly capture the selection of tuple elements, we adopt some database notation: Elements of a tuple are called $attributes$, e.g., we could have two attributes msg and sig denoting the message and the signature element of each tuple. For a tuple $x \in D$, the value of its attribute att is written $x.att$. If the values of one distinguished attribute att are unique among all tuples in D, i.e., the attribute gives a one-to-one correspondence to the tuples in the array, we call att a primary key attribute. We use this to select elements of a tuple, i.e., if a primary key attribute att exists in D and att_2 is another attribute in D, and D is clear from the context, we simply write $att_2[a]$ instead of $x.att_2$, where x denotes the unique tuple with $x.att = a$. If no such tuple exists, we define $att_2[a] := \downarrow$.

2.2 Non-Reactive Definitions

Signature schemes often have memory. As explained in the introduction, signature schemes that divulge the history of the messages signed earlier are not suited for use in a general asynchronous reactive environment. In the following definition, we therefore do not allow a signature scheme to use arbitrary memory for signing a message, but we model its memory by a counter.

Definition 1 (Counter-based Signature Schemes). *A* counter-based signature scheme *is a triple* $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *of polynomial-time algorithms, where* gen *and* sign *are probabilistic. The algorithm* gen *takes an input* $(1^k, 1^s)$ *with* $k, s \in \mathbb{N}$*, where* s *denotes the desired maximum number of signatures and* k *a security parameter, and outputs a pair* (sk, pk) *of a secret signing key and a public test key in* Σ^+*. The algorithm* sign *takes such a secret key, a counter* $c \in \{1, \ldots, s\}$*, and a message* $m \in \Sigma^+$ *as inputs and produces a signature in* Σ^+*. We write this* $sig \leftarrow \mathsf{sign}_{sk,c}(m)$*. Similarly, we write verification as* $b := \mathsf{test}_{pk}(m, sig)$ *with* $b \in \{\mathsf{true}, \mathsf{false}\}$*. If the result is* true*, we say that the signature is* valid

for m. For a correctly generated key pair, a correctly generated signature for a message m must always be valid for m.

When we speak about a signature scheme in the following, we always mean a counter-based signature scheme in the sense of Definition 1.

Security of a signature scheme is defined against existential forgery under adaptive chosen-message attacks (CMA-security). The definition corresponds to the GMR-definition [40] restricted to counter-based signature schemes.

Definition 2 (CMA-security for Counter-based Signature Schemes). *Given a signature scheme $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ and a polynomial $s \in \mathbb{N}[x]$, the* GMR signature oracle Sig_s*, or short* signature oracle*, is defined as follows: It has variables sk, pk and a counter c initialized with 0, and the following transition rules:*

- *First generate a key pair $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$ and output pk.*
- *On input (sign, m) with $m \in \Sigma^+$, and if $c < s(k)$, set $c := c+1$ and return $sig \leftarrow \mathsf{sign}_{sk,c}(m)$.*

The signature scheme is called CMA-secure *if for every polynomial s and every probabilistic polynomial-time machine $\mathsf{A}_{\mathsf{sig}}$ that interacts with Sig_s and finally outputs two values m and sig (meant as a forged signature for the message m), the probability is negligible (in k) that $\mathsf{test}_{pk}(m, sig) = \mathsf{true}$ and m is not among the messages previously signed by the signature oracle.*

We further state a variant of Definition 2, which we call *skipping security*. It is slightly stronger than the original definition by allowing the adversary to choose the counter value, but only in a strictly monotonic increasing sequence, i.e., he can only skip counter values. This variant is useful in later proofs.

Definition 3 (Skipping Security). *Given a signature scheme $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ and a polynomial $s \in \mathbb{N}[x]$, the* skipping signature oracle $\mathsf{SkipSig}_s$ *is defined as follows: It has variables sk, pk and a counter c initialized with 0, and the following transition rules:*

- *First generate a key pair $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$ and output pk.*
- *On input (sign, m, c^*) with $m \in \Sigma^+$, and if $c < c^* \leq s(k)$, set $c := c^*$ and return $sig \leftarrow \mathsf{sign}_{sk,c^*}(m)$.*

The signature scheme is called skipping secure *if for every polynomial s and every probabilistic polynomial-time machine $\mathsf{A}_{\mathsf{skipsig}}$ that interacts with $\mathsf{SkipSig}_s$ and finally outputs two values m and sig (meant as a forged signature for the message m), the probability is negligible (in k) that $\mathsf{test}_{pk}(m, sig) = \mathsf{true}$ and m is not among the messages previously signed by the signature oracle.*

Lemma 1 (Skipping Signatures). *A signature scheme Sig is CMA-secure if and only if it is skipping secure. This holds with essentially unchanged concrete security, except that a CMA adversary may need up to $s(k)$ oracle calls even if the skipping adversary uses less.*

Proof. The right-to-left direction is clear as an adversary $\mathsf{A}_{\mathsf{skipsig}}$ may ask for a signature for every counter value, i.e., CMA-security is a special case of skipping security.

For the opposite direction, assume that an adversary $\mathsf{A}_{\mathsf{skipsig}}$ successfully attacks the skipping signature oracle $\mathsf{SkipSig}_s$ with not negligible probability. We then construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signature oracle Sig_s as follows. It uses $\mathsf{A}_{\mathsf{skipsig}}$ as a blackbox submachine (also called oracle). $\mathsf{A}_{\mathsf{sig}}$ first selects a message $m^* \xleftarrow{\mathcal{R}} \Sigma^k$. Whenever the adversary $\mathsf{A}_{\mathsf{skipsig}}$ outputs a command (sign, m, c^*) with $c < c^* \leq s(k)$ for the current counter value c of Sig_s, then $\mathsf{A}_{\mathsf{sig}}$ inputs (sign, m^*) to Sig_s for $c^* - c - 1$ times, thus bringing the counter of Sig_s to the value $c^* - 1$. Then $\mathsf{A}_{\mathsf{sig}}$ inputs (sign, m) to Sig_s and forwards the response sig to $\mathsf{A}_{\mathsf{skipsig}}$. It is easy to see that $\mathsf{A}_{\mathsf{sig}}$ together with Sig_s correctly simulates the skipping signature oracle $\mathsf{SkipSig}_s$ because Sig_s computes sig as $sig \leftarrow \mathsf{sign}_{sk,c^*}(m)$ just as $\mathsf{SkipSig}_s$ would. Hence the adversary $\mathsf{A}_{\mathsf{skipsig}}$ outputs a successful forgery (m', sig') with its usual, not negligible probability in this scenario. This is also a valid forgery for $\mathsf{A}_{\mathsf{sig}}$ unless $m' = m^*$. However, no information about m^* in the Shannon sense leaks to $\mathsf{A}_{\mathsf{skipsig}}$; hence this only happens with exponentially small probability. Hence the success probability of $\mathsf{A}_{\mathsf{sig}}$ is still not negligible. As to concrete security, the success probability is even almost unchanged. The oracle calls by $\mathsf{A}_{\mathsf{sig}}$ are bounded by $s(k)$, and its runtime is typically dominated by that of $\mathsf{A}_{\mathsf{skipsig}}$.

2.3 A New Reactive Definition

To obtain a security definition that is sufficient in a reactive environment, we have to extend the capabilities of the adversary when interacting with the signature oracle. More precisely, we still have to allow for signing arbitrary messages, but the obtained signatures are stored within the signature oracle and only output upon request of the adversary. Now a signature is considered a forgery for a message m if the signature is valid for m and if no signature for this message has been revealed. This is captured in the following definition.

Definition 4 (Reactive Signature Security). *Given a signature scheme* $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *and a polynomial* $s \in \mathbb{N}[x]$*, the* reactive signature oracle RSig_s *is defined as follows: It contains variables sk, pk, a counter c initialized with 0, an initially empty set $\mathcal{C}$ of counter values, and an initially empty array $SIGS$*

with attributes c, m, and sig for storing tuples of counter values, messages, and signatures. The counter c can be used as a primary key attribute; this is clear by inspection of the transitions below. The transition rules of RSig_s *are:*

- *First generate a key pair* $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$ *and output* pk.
- *On input* (sign, m) *with* $m \in \Sigma^+$*, and if* $c < s(k)$*, set* $c := c+1$ *and* $sig \leftarrow \mathsf{sign}_{sk,c}(m)$*, and store* (c, m, sig) *in* $SIGS$.
- *On input* (reveal, i)*, and if* $i \leq c$*, set* $\mathcal{C} := \mathcal{C} \cup \{i\}$ *and return* $sig[i]$.

The signature scheme is called reactively secure *if for every polynomial s and every probabilistic polynomial-time machine* $\mathsf{A}_{\mathsf{rsig}}$ *that interacts with* RSig_s *and finally outputs two values m and sig (meant as a forged signature for m), the probability that* $\mathsf{test}_{pk}(m, sig) = \mathsf{true}$ *and* $m \neq m[i]$ *for all* $i \in \mathcal{C}$ *is negligible (in k).*

We first treat a simple but common case where the definitions are equivalent with essentially the same concrete security.

Lemma 2 (Memory-less Schemes). *Let Sig denote a memory-less signature scheme, i.e., the function* sign *does not even depend on a counter. Then Sig is CMA-secure if and only if it is reactively secure. This holds with essentially unchanged concrete security.*

Proof. The right-to-left direction is clear (or see the proof of Theorem 1). If an adversary $\mathsf{A}_{\mathsf{rsig}}$ breaks Sig in the reactive scenario of Definition 4, we can easily construct an adversary $\mathsf{A}_{\mathsf{sig}}$ that has the same success probability against the same signature scheme in the non-reactive scenario of Definition 2: $\mathsf{A}_{\mathsf{sig}}$ simply defers each signature request of the original adversary (which it uses as a blackbox submachine), i.e., each input (sign, m), until $\mathsf{A}_{\mathsf{rsig}}$ first requests the corresponding signature. As signing is memory-less, making the signatures in the wrong order makes no difference. Thus $\mathsf{A}_{\mathsf{sig}}$ together with Sig_s simulates RSig_s correctly. Moreover, every forged signature by $\mathsf{A}_{\mathsf{rsig}}$ is also a suitable forgery for $\mathsf{A}_{\mathsf{sig}}$. Moreover, for the concrete security, $\mathsf{A}_{\mathsf{sig}}$ uses at most as many oracle calls than $\mathsf{A}_{\mathsf{rsig}}$ and its runtime is dominated by that of $\mathsf{A}_{\mathsf{rsig}}$.

3 Reduction Proof for Unchanged Signature Schemes

In this section, we show that a counter-based, CMA-secure signature scheme is already reactively secure. Hence signature schemes that only maintain a counter as their local state can safely be used in an asynchronous reactive environment without having to bother about the problem sketched in the introduction. As a drawback, however, we will see that the concrete security gets worse.

Theorem 1 (Reactive Security of Unchanged Signature Schemes). *A counter-based signature scheme Sig is reactively secure if and only if it is CMA-secure. In the concrete security, we essentially lose a factor of s, the maximum number of signatures, in the adversary's success probability when proving reactive security from CMA security.*

Proof. The proof of the left-to-right direction is straightforward, since a reactively secure system is in particular secure for an adversary requesting all signed messages immediately when they have been made.

We now prove the opposite direction. From Lemma 1 we know that Sig is skipping secure. Hence we only need to show that if there exists a successful adversary $\mathsf{A}_{\mathsf{rsig}}$ against reactive security, there also exists a successful adversary $\mathsf{A}_{\mathsf{skipsig}}$ against skipping security, i.e., against the oracle $\mathsf{SkipSig}_s$. The adversary $\mathsf{A}_{\mathsf{skipsig}}$ uses the adversary $\mathsf{A}_{\mathsf{rsig}}$ as a blackbox; we call the rest of it the *simulator* Sim. This is shown in Figure 1.

Definition of the simulator. Like RSig_s, the simulator Sim maintains a counter c initialized with 0 and an initially empty array $SIGS$ with attributes c, m, and sig for storing tuples of counter values, messages, and signatures, where c is used as a primary key attribute. Additionally, it chooses a counter value $c^* \xleftarrow{\mathcal{R}} \{0, \ldots, s(k)\}$ where it will cheat. Intuitively, c^* is a guess whether $\mathsf{A}_{\mathsf{rsig}}$ will forge a signature on a totally new message ($c^* = 0$) or on a signed message whose signature was not revealed; here c^* denotes the corresponding counter value. Cheating means that the simulator does not input the c^*-th sign query from $\mathsf{A}_{\mathsf{rsig}}$ to the machine $\mathsf{SkipSig}_s$. If the message m^* from this query is ever signed again later, the simulator skips it again. If $c^* = 0$, then Sim does not cheat at all. This is important, since an adversary $\mathsf{A}_{\mathsf{rsig}}$ that requests all signatures would otherwise always catch our simulator cheating. The variable $m^* \in \Sigma^+$ is initialized with $\downarrow$. The transitions of Sim are sketched in Figure 1 and defined as follows:

- First, upon receiving a public key pk from $\mathsf{SkipSig}_s$, choose $c^* \xleftarrow{\mathcal{R}} \{0, 1, \ldots, s(k)\}$ and output pk to $\mathsf{A}_{\mathsf{rsig}}$.
- On input (sign, m) from $\mathsf{A}_{\mathsf{rsig}}$ with $m \in \Sigma^+$: If $c < s(k)$, set $c := c + 1$, otherwise abort, i.e., stop the current transition without further action. We distinguish two cases:
  - (No cheating): If $c < c^*$ or ($c > c^* \wedge m \neq m^*$), output (sign, m, c) to $\mathsf{SkipSig}_s$ and obtain a signature sig. Store (c, m, sig) in $SIGS$.
  - (Cheating): If $c = c^*$ or ($c > c^* \wedge m = m^*$), set $m^* := m$ and store $(c, m^*, \downarrow)$ in $SIGS$.

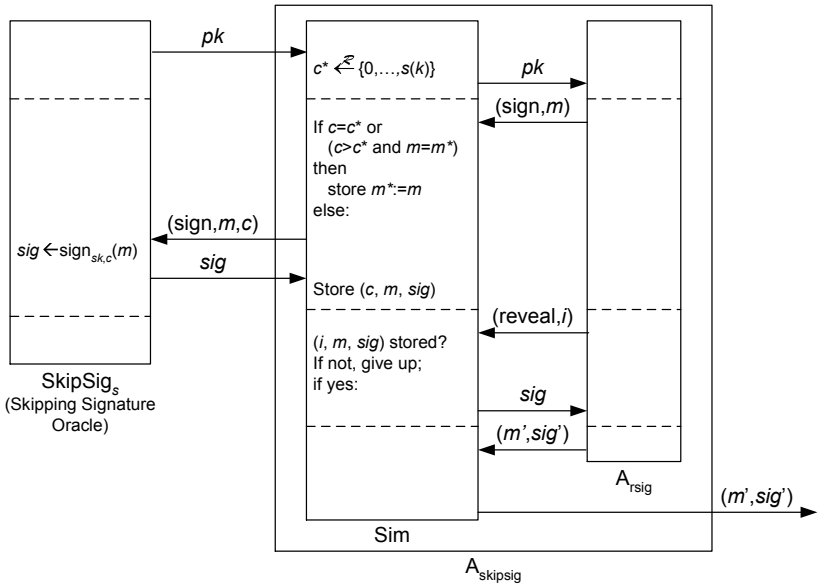

Fig. 1. Overview of the reduction proof for unchanged signature schemes.

- On input (reveal, i) from $\mathsf{A}_{\mathsf{rsig}}$: If $i > c$ abort. If $sig := sig[i] = \downarrow$, give up the simulation. ($\mathsf{A}_{\mathsf{rsig}}$ has requested a signature that the simulator skipped.) Otherwise output sig to $\mathsf{A}_{\mathsf{rsig}}$.
- On input (m', sig') from $\mathsf{A}_{\mathsf{rsig}}$, i.e., a supposed forgery: Output (m', sig').

Proof of correct simulation for correct guess. In the original attack of $\mathsf{A}_{\mathsf{rsig}}$ against the reactive signature oracle RSig_s, let the random variable q over $\{0, \ldots, s(k), \mathsf{not\_valid}\}$ denote the index of the first sign query that contained the forgery message m', with $q = 0$ if m' never occurred in a sign query, and $q = \mathsf{not\_valid}$ if the forgery that $\mathsf{A}_{\mathsf{rsig}}$ outputs is not valid. We show that the simulation is perfect in the case $c^* = q$. More precisely, m' and q are functions of the pair $r = (r_A, skr)$ of the randomness r_A for $\mathsf{A}_{\mathsf{rsig}}$ and the randomness skr for the signature system, containing the secret key and randomness for signing for each counter value c. We show that if the same r is used in the simulated scenario and if $c^* = q := q(r)$, then $\mathsf{A}_{\mathsf{rsig}}$ obtains the same view in the simulated scenario. In particular, m' is thus also the forgery message in that scenario and q the index of the first sign query for m'.

For $c^* = 0$ perfect simulation is clear since the simulator never cheats and uses the skipping oracle to sign all messages in the correct order.

Now let $c^* \in \{1, \ldots, s(k)\}$. Here the proof is in principle an induction over the interaction length. Before the c^*-th sign query the simulation is again

clearly perfect. Thus the c^*-th sign query contains the message m' (for the first time), and hence $m^* = m'$. Sign queries for messages $m \neq m'$ are simulated perfectly also after the c^*-th sign query, in particular because Sim increments the counter c correctly in all sign queries until it reaches $s(k)$ and never modifies it elsewhere; this also implies that $\mathsf{SkipSig}_s$ returns a signature for all calls by Sim. For sign queries for message m', the new tuple in $SIGS$ is wrong, but neither Sim nor RSig_s make an output and thus the view of $\mathsf{A}_{\mathsf{rsig}}$ is unchanged. Now we consider a query (reveal, i). If $i > c$ for the current value of c, then Sim outputs nothing, and so would RSig_s. Otherwise, we know that $m[i] \neq m'$ because so far $\mathsf{A}_{\mathsf{rsig}}$ acts as in the scenario with RSig_s, and there (m', sig') is a valid reactive forgery (because $q \neq \mathsf{not\_valid}$), and a reveal query for the same message would have made it invalid. This implies $sig[i] \neq \downarrow$, and thus Sim does not give up the simulation. Furthermore, $sig[i]$ was thus stored during one of the sign queries that were simulated perfectly, and hence the output of this reveal query is also correct. Altogether this shows that at least for $c^* = q$ the simulation results in the correct view of $\mathsf{A}_{\mathsf{rsig}}$.

Moreover, the signature sig' is a valid forgery for m' also against $\mathsf{SkipSig}_s$ in the case $c^* = q$ since by construction Sim never asks $\mathsf{SkipSig}_s$ to sign m'.

Success probability. By the definition of the random variable q, the probability that $\mathsf{A}_{\mathsf{rsig}}$ successfully attacks RSig_s is

$$P_{\mathsf{A}_{\mathsf{rsig}}}(k) = \sum_{i=0}^{s(k)} \mathsf{Pr}[q = i].$$

The simulator Sim chooses c^* uniformly from $\{0, \ldots, s(k)\}$ and independently of the randomness r that determines q. We also showed above that $\mathsf{A}_{\mathsf{skipsig}}$ is always successful for $c^* = q$. Hence for its success probability $P_{\mathsf{A}_{\mathsf{skipsig}}}$ we obtain

$$\begin{aligned} P_{\mathsf{A}_{\mathsf{skipsig}}}(k) \geq \mathsf{Pr}[c^* = q] &= \sum_{i=0}^{s(k)} \mathsf{Pr}[c^* = i \wedge q = i] \\ &= \sum_{i=0}^{s(k)} \mathsf{Pr}[c^* = i] \cdot \mathsf{Pr}[q = i] = \frac{1}{s(k)+1} P_{\mathsf{A}_{\mathsf{rsig}}}(k). \end{aligned}$$

Since $P_{\mathsf{A}_{\mathsf{rsig}}}(k)$ is not negligible by assumption and s is a polynomial, the success probability of $\mathsf{A}_{\mathsf{skipsig}}$ is not negligible. However, we lost a factor of about $s(k)^{-1}$ in the error probability for concrete security. Otherwise the concrete security is good: $\mathsf{A}_{\mathsf{skipsig}}$ uses at most as many oracle calls as $\mathsf{A}_{\mathsf{rsig}}$ and its runtime is typically dominated by that of $\mathsf{A}_{\mathsf{rsig}}$. Furthermore, Lemma 1 essentially retained the success probabilities and runtime, and we easily see that over both proofs the adversary $\mathsf{A}_{\mathsf{sig}}$ does not make more oracle calls than $\mathsf{A}_{\mathsf{rsig}}$.

4 Stronger Reactively Secure Signature Schemes by Randomization

In this section, we use additional randomization to transform a counter-based CMA-secure signature scheme into a reactively secure scheme with almost the same concrete security. We assume that an efficient encoding of tuples into Σ^+ is given.

Definition 5 (Message Randomization of Signature Schemes). *Let* $Sig = (\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *be a signature scheme. Then we define the* corresponding message-randomized signature scheme $Sig^* = (\mathsf{gen}, \mathsf{sign}^*, \mathsf{test}^*)$ *as follows: Let* $\mathsf{sign}^*_{sk,c}(m) := (r, \mathsf{sign}_{sk,c}((m,r)))$ *for* $r \xleftarrow{\mathcal{R}} \{0,1\}^k$*, and* $\mathsf{test}^*_{pk}(m, sig^*) := \mathsf{test}_{pk}((m,r), sig)$ *if* sig^* *is of the form* (r, sig)*, and* false *otherwise.*

Theorem 2 (Security of Message Randomization). *Let* Sig *be a CMA-secure signature scheme. Then the corresponding message-randomized signature scheme* Sig^* *is reactively secure with essentially the same concrete security.*

Proof. We show that if there exists a successful adversary $\mathsf{A}^*_{\mathsf{rsig}}$ against the signature scheme Sig^* in a reactive scenario, there also exists a successful adversary $\mathsf{A}_{\mathsf{sig}}$ against the scheme Sig in the CMA scenario, i.e., against the corresponding oracle Sig_s. The adversary $\mathsf{A}_{\mathsf{sig}}$ uses $\mathsf{A}^*_{\mathsf{rsig}}$ as a blackbox, and we call the rest of it the simulator Sim; see Figure 2. Let $Sig = (\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ and $Sig^* = (\mathsf{gen}, \mathsf{sign}^*, \mathsf{test}^*)$.

Definition of the Simulator. The simulator Sim maintains a counter c initialized with 0, an initially empty set $\mathcal{C}$ of counter values corresponding to revealed signatures, and an initially empty array $SIGS$ with attributes c, m, r, and sig for storing tuples of a counter values, messages, random values, and signatures, where c is used as the primary key attribute. The transitions of Sim are sketched in Figure 2 and defined as follows:

- First, upon receiving a public key pk from Sig_s, output pk to $\mathsf{A}^*_{\mathsf{rsig}}$.
- On input (sign, m) from $\mathsf{A}^*_{\mathsf{rsig}}$ with $m \in \Sigma^+$: If $c < s(k)$, set $c := c+1$ and $r \xleftarrow{\mathcal{R}} \{0,1\}^k$ and output $(\mathsf{sign}, (m,r))$ to Sig_s and obtain a signature sig. Store (c, m, r, sig) in $SIGS$.
- On input (reveal, i) from $\mathsf{A}^*_{\mathsf{rsig}}$: If $SIGS[i] = \downarrow$ abort. Otherwise let $(i, m, r, sig) := SIGS[i]$, set $\mathcal{C} := \mathcal{C} \cup \{i\}$ and output (r, sig) to $\mathsf{A}^*_{\mathsf{rsig}}$.
- On input (m, sig^*) from $\mathsf{A}^*_{\mathsf{rsig}}$ where $sig^* = (r, sig)$, i.e., sig^* is supposed to be a forgery for m: Output $((m,r), sig)$, i.e., sig should be a forgery for (m,r).

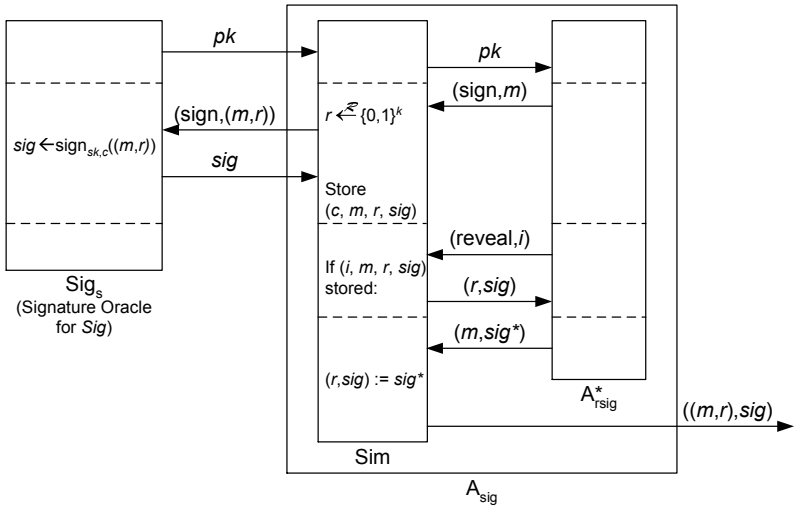

Fig. 2. Overview of the reduction proof for randomized signature schemes.

Correct Simulation and Successful Forgeries. It is easy to see that Sim together with Sig_s perfectly simulates RSig^*_s, the reactive signature oracle for Sig^*. The sets $\mathcal{C}$ are always equal in the two scenarios.

Now assume that $\mathsf{A}^*_{\mathsf{rsig}}$'s output (m, sig^*) is a valid forgery. We want to show that $\mathsf{A}_{\mathsf{sig}}$'s output $((m, r), sig)$ is also a valid forgery except in a certain rare case. The precondition means that $m \neq m[c]$ for all $c \in \mathcal{C}$ and $\mathsf{test}^*_{pk}(m, sig^*) = \mathsf{true}$. Hence sig^* is a pair (r, sig) and $\mathsf{test}_{pk}((m, r), sig) = \mathsf{true}$. Thus $\mathsf{A}_{\mathsf{sig}}$ makes an output, and the output is at least a valid signature.

Hence $\mathsf{A}_{\mathsf{sig}}$'s output is indeed a successful forgery if the message (m, r) has never been signed by Sig_s, i.e., if no tuple (c, m, r, sig) with the given m and r exists in $SIGS$ of Sim.

If a tuple (c, m, r, sig) occurs in $SIGS$, then $c \notin \mathcal{C}$ because of the precondition $m \neq m[c]$ for all $c \in \mathcal{C}$. Intuitively this means that $\mathsf{A}^*_{\mathsf{rsig}}$ has guessed the unknown random value r correctly. The inputs $(r[i], sig[i])$ that $\mathsf{A}^*_{\mathsf{rsig}}$ obtained do not depend on $r[c]$ because the random values are chosen independently and the signatures of a counter-based signature scheme only depend on the current counter and input, and the random value $r[c]$ does not influence this counter.

Success Probability. Due to the perfect simulation, $\mathsf{A}^*_{\mathsf{rsig}}$ outputs a successful forgery in the simulated scenario with the same probability $P_{\mathsf{A}^*_{\mathsf{rsig}}}(k)$ as in interaction with RSig^*_s. For each such output, we have just shown that the output of $\mathsf{A}_{\mathsf{sig}}$ is also a successful forgery unless one of the at most $s(k)$ unrevealed, independent random values $r[c]$ equals the value r in $\mathsf{A}^*_{\mathsf{rsig}}$'s output. Hence $\mathsf{A}_{\mathsf{sig}}$

has almost the same success probability

$$P_{\mathsf{A}_{\mathsf{sig}}}(k) = (1 - \frac{s(k)}{2^k}) P_{\mathsf{A}^*_{\mathsf{rsig}}}(k).$$

It also makes the same number of oracle queries as $\mathsf{A}^*_{\mathsf{rsig}}$, and its runtime is typically dominated by that of $\mathsf{A}^*_{\mathsf{rsig}}$ because the runtime of Sim is essentially given by generating $s(k)$ random values and making $s(k)$ lookups in a sorted array of length $s(k)$.

5 A Low-level Ideal Signature System and its Realization

In this section, we present an ideal system which, at a low level of abstraction, offers the functionality of a secure signature scheme in a reactive and composable fashion. Essentially, it stores which keys belong to honest users and which messages the users signed with these keys, and it never accepts signatures that are supposedly made with one of these keys on different messages, i.e., forgeries. We then show that a canonical realization is as secure as this low-level ideal system in the sense of reactive simulatability.

5.1 The Low-level Ideal System

The ideal system is represented by one centralized machine with the overall functionality, while the realization consists of one machine per participant. Its honest users are, without loss of generality, numbered $\{1, ..., n\}$. We distinguish the in- and outputs from and to different users by making them at so-called ports $\mathsf{in}_{\mathsf{sig},u}?$ and $\mathsf{out}_{\mathsf{sig},u}!$ for $u \in \{1, ..., n\}$.

Definition 6 (Low-level Ideal Signature Machine). *Let a signature scheme $Sig = (\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ and parameters $n \in \mathbb{N}$ and $s \in \mathbb{N}[x]$ be given. Then we define the* corresponding low-level ideal signature machine $\mathsf{Sig}_{\mathsf{low\_id},n,s}$. *It maintains two initially empty arrays, $KEYS$ with attributes $user$, sk, pk, and c for the user who "owns" the key pair, the secret and public key, and the current counter for this key, and $SIGNED$ with attributes pk and m for a public key and a message. The transitions are defined as follows. Let u be the index of the port $\mathsf{in}_{\mathsf{sig},u}?$ where the current input occurs; the resulting output goes to $\mathsf{out}_{\mathsf{sig},u}!$.*

- *On input* $(\mathsf{generate})$*: Set* $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$*, store* $(u, sk, pk, 0)$ *in* $KEYS$*, and output* pk*.*
- *On input* (sign, pk, m)*: Retrieve a tuple* $(u, sk, pk, c) \in KEYS$ *with the given* u *and* pk*. If none or more than one exists or if* $c = s(k)$*, output the error symbol* $\downarrow$*. Else set* $c := c + 1$ *in this tuple in* $KEYS$*. Then set* $sig \leftarrow \mathsf{sign}_{sk,c}(m)$*, store* (pk, m) *in* $SIGNED$*, and output* sig*.*

– *On input* $(\mathsf{test}, pk, m, sig)$*: Retrieve a tuple* $(v,\ sk,\ pk,\ c) \in KEYS$ *with the given* pk*. If none or more than one exists, output* $\mathsf{test}_{pk}(m, sig)$*. Else if* $(pk, m) \in SIGNED$*, output* $\mathsf{test}_{pk}(m, sig)$*, else* false*.*

Other inputs are ignored.

The low-level ideal machine never outputs secret keys. For signing, user u inputs the public key to designate the desired private key, and the machine verifies that the key tuple belongs to u. The test function is a normal signature test for unknown public keys (typically keys generated by the adversary). For known public keys, the low-level ideal machine first verifies that the message was indeed signed with this key, and then it additionally verifies that the signature presented is valid.

The main difference to the signature functionality in [26] is that the adversary learns nothing about what honest users sign. The low-level ideal machine further formally depends on the given signature scheme Sig, like the first similar low-level idealizations in [44]. This could be alleviated by the technique from [28] of letting the adversary choose the algorithms, so that the overall low-level ideal functionality comprises all possible instantiations. However, in all use cases known to us it is not necessary: One can either assume given algorithms because the low-level idealization is only used to prove a larger system, or a really abstract idealization fits better because arguing about the evaluation of an arbitrary algorithm input by an adversary is far beyond the kind of theories implemented in current automated proof tools. In particular, cryptographic objects that would be output by such arbitrary algorithms can be addressed by handles (names, pointers) in such an abstraction, as in [14].

5.2 Cryptographic Realization

The cryptographic realization of the low-level ideal signature functionality is the natural use of digital signatures in a distributed system, i.e., it consists of a separate machine Sig_u for each user u, and each machine signs and tests in the normal way. Together, these machines offer the same ports and accept the same inputs as the ideal machine.

Definition 7 (Real Low-level Signature Machines). *Let a signature scheme* $Sig = (\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *and parameters* $u \in \mathbb{N}$ *and* $s \in \mathbb{N}[x]$ *be given. Then we define the* corresponding low-level real signature machine $\mathsf{Sig}_{u,s}$ for user u*. It has ports* $\mathsf{in}_{\mathsf{sig},u}?$ *and* $\mathsf{out}_{\mathsf{sig},u}!$ *and maintains an initially empty array* $KEYS_u$ *with attributes* sk*,* pk*, and* c *for a secret and public key and the current counter for this key pair. The transitions are defined as follows.*

– *On input* $(\mathsf{generate})$*: Set* $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$*, store* $(sk, pk, 0)$ *in* $KEYS_u$*, and output* pk*.*
– *On input* (sign, pk, m)*: Retrieve a tuple* $(sk, pk, c) \in KEYS_u$ *with the given* pk*. If none or more than one exist or if* $c = s(k)$*, return* $\downarrow$*. Else set* $c := c + 1$ *in this tuple in* $KEYS_u$ *and output* $sig \leftarrow \mathsf{sign}_{sk,c}(m)$*.*
– *On input* $(\mathsf{test}, pk, m, sig)$*, output* $\mathsf{test}_{pk}(m, sig)$*.*

Other inputs are ignored. We denote the set of these machines for n *users by* $\mathsf{Sig}_{\mathsf{real},n,s} := \{\mathsf{Sig}_{u,s} | u \in \{1, \ldots, n\}\}$.

We claim that this real system is as secure as the low-level ideal system in the sense of reactive simulatability.

5.3 Reactive Simulatability

The notion of *reactive simulatability* captures the idea of refinement that preserves not only integrity properties, but also confidentiality properties. Intuitively it can be stated as follows, when applied to the relation between a real and an ideal system. Everything that can happen to users of the real system in the presence of arbitrary adversaries can also happen to the same users with the ideal system, where attack capabilities are usually much more restricted. In particular, this comprises confidentiality because the notion of what happens to users not only includes their in- and outputs to the system, but also their communication with the adversary. This includes whether the adversary can guess secrets of the users or partial information about them.

The formal definition of reactive simulatability relies on the framework of secure reactive systems in asynchronous networks by Backes, Pfitzmann, and Waidner [56]. We give a brief overview of those parts of the framework that are necessary to state the definition of reactive simulatability.

A *system* consists of several possible *structures*. A structure consists of a set $\hat{M}$ of connected machines and a subset S of the free *ports* that the honest users connect to. A *machine* is a probabilistic IO automaton (extended finite-state machine) in a slightly refined model to allow complexity considerations. For these machines Turing-machine realizations are defined, and the complexity of those is tyically measured in terms of a common security parameter k, given as the initial work-tape content of every machine. Each structure is complemented to a *configuration* by an arbitrary *user* machine H and *adversary* machine A, where H connects only to ports in S and A to the rest, and they may interact. The set of configurations of a system Sys is called $\mathsf{Conf}(Sys)$. The general scheduling model in [56] gives each connection c (from an output port $\mathsf{c}!$ to an input port $\mathsf{c}?$) a buffer, and the machine with the corresponding clock port $\mathsf{c}^{\triangleleft}!$ can schedule

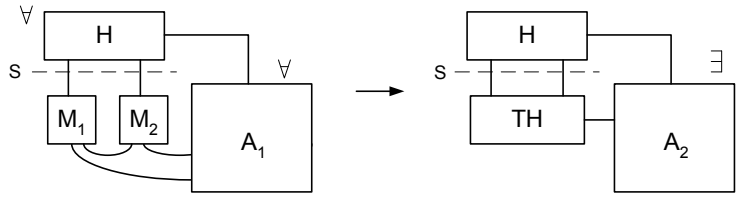

Fig. 3. Simulatability: The two views of H must be indistinguishable.

a message there when it makes a transition. In real asynchronous cryptographic systems, network connections are typically scheduled by A. A configuration is a runnable system, i.e., for each k one gets a well-defined probability space of *runs*. The *view* of a machine in a run is the restriction to all in- and outputs this machine sees and its internal states. Formally, the view $view_{conf}(\mathsf{M})$ of a machine M in a configuration $conf$ is a *family of random variables* with one element for each security parameter value k.

Now reactive simulatability for a real system Sys_{real} and an ideal system Sys_{id} is captured as follows [56]: For every structure $(\hat{M}_1, S) \in Sys_{\mathsf{real}}$, every polynomial-time user H, and every polynomial-time adversary A_1, there exists a polynomial-time adversary A_2 on a corresponding ideal structure $(\hat{M}_2, S) \in Sys_{\mathsf{id}}$ such that the view of H is computationally indistinguishable in the two configurations. This is illustrated in Figure 3. Indistinguishability is a well-known cryptographic notion from [60].

Definition 8 (Computational Indistinguishability). *Two families* $(\mathsf{var}_k)_{k\in\mathbb{N}}$ *and* $(\mathsf{var}'_k)_{k\in\mathbb{N}}$ *of random variables on common domains* D_k *are* computationally indistinguishable *("≈") iff for every algorithm* Dis *(the distinguisher) that is probabilistic polynomial-time in its first input,*

$$|P(\mathsf{Dis}(1^k, \mathsf{var}_k) = 1) - P(\mathsf{Dis}(1^k, \mathsf{var}'_k) = 1)| \in NEGL,$$

where $NEGL$ *denotes the set of all negligible functions.*

Intuitively, given the security parameter and an element chosen according to either var_k or var'_k, Dis tries to guess which distribution the element came from.

Definition 9 (Reactive Simulatability). *For two systems* Sys_{real} *and* Sys_{id}*, one says* $Sys_{\mathsf{real}} \geq Sys_{\mathsf{id}}$ *(*at least as secure as*) iff for every polynomial-time configuration* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_{\mathsf{real}})$*, there exists a polynomial-time configuration* $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_{\mathsf{id}})$ *(with the same* H*) such that* $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

For the signature systems presented above, this is even shown with $\mathsf{A}_2 = \mathsf{A}_1$; we call this strong version computational observational equivalence following [43]. Composition and preservation theorems for reactive simulatability [56, 13, 7, 6, 8] show that it has the properties expected from a refinement notion: First, if we design a larger system based on a specification of a subsystem, and later plug the implementation of the subsystem in, the entire implementation of the larger systems is as secure as its design in the same sense of reactive simulatability. Secondly, if we prove specific security properties for the specification, they also hold for the implementation.

Simulatability was first sketched for secure multi-party function evaluation, i.e., the computation of one output tuple from one tuple containing one secret input from each participant in [59] and defined (with different degrees of generality and rigorosity) in [39, 21, 50, 25]. Problems such as the separation of users and adversaries, or defining runtime restrictions in the face of continuous external inputs, do not occur in this case. The idea of simulatability was subsequently also used for specific reactive problems, e.g., [36, 24, 28], without a detailed or general definition. In a similar way it was used for the construction of generic solutions for large classes of reactive problems [38, 37, 41] (usually yielding inefficient solutions and assuming that all parties take part in all subprotocols). Computational observational equivalence was introduced in [43]. The first general reactive definition of simulatability was proposed (after some earlier sketches, in particular in [38, 54, 25]) in [55] for a synchronous version of our underlying general reactive model. The definition was extended to asynchronous scenarios in [56, 26].

5.4 Simulatability Proof

We claim that the low-level real signature machines for a set of users are as secure as the corresponding low-level ideal system if the underlying signature scheme is secure.[2] More precisely, our sets $\mathsf{Sig}_{\mathsf{real},n,s}$ and $\{\mathsf{Sig}_{\mathsf{low\_id},n,s}\}$, together with the set of all free ports, are each the only structure of a system. We identify $\mathsf{Sig}_{\mathsf{real},n,s}$ and $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ with these systems and write the simulatability claim as follows.

Theorem 3 (Security of the Low-level Real Signature System). *Given a secure signature scheme according to Definition 2, we have for all* $n \in \mathbb{N}$ *and*

[2] We defined both these systems for counter-based signature schemes, and Theorem 1 implies that any CMA-secure counter-based signature scheme is already reactively secure. Both systems and the proof can easily be generalized to CMA-secure signature schemes with arbitrary memory. However, then even the ideal system suffers from the problems mentioned in the introduction.

$s \in \mathbb{N}[x]$*:*

$$\mathsf{Sig}_{\mathsf{real},n,s} \geq \mathsf{Sig}_{\mathsf{low\_id},n,s}.$$

This holds even in the strong sense of computational observational equivalence.

Proof. We show that for an arbitrary polynomial-time overall user machine interacting with either of our systems, the views are indistinguishable. (As an adversary has no special ports here, we need not distinguish users and adversaries.) The proof is in principle an induction over the number of inputs so far, showing equality of the views except in one case, followed by a reduction proof that this case is rare.

The proof uses and shows the following invariant: at all times and for every $u \in \{1, \ldots, n\}$, the array $KEYS_u$ of a real machine $\mathsf{Sig}_{u,s}$ can be derived from $KEYS$ of the ideal machine by restricting $KEYS$ to tuples with $user = u$ and then deleting the attribute $user$. All tuples in $KEYS$ are covered in this way.

We now consider the three acceptable inputs at a port $\mathsf{in}_{\mathsf{sig},u}?$.

- On input $(\mathsf{generate})$, both machines generate a key pair and output pk in the same way. The way they store the keys in $KEYS_u$ and $KEYS$ retains the invariant.
- On input (sign, pk, m), both machines first look for an appropriate secret key. By the invariant, $\mathsf{Sig}_{u,s}$ finds exactly one tuple $(sk, pk, c) \in KEYS_u$ if and only if $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ finds exactly one tuple $(u, sk, pk, c) \in KEYS$ and the tuples then have the same sk and c. Both output $\downarrow$ if none or more than one tuple is found, or if $c = s(k)$. Else both increment the counter c, thus retaining the invariant. Then both output a signature sig generated in the same way. Only $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ additionally stores (pk, m) in $SIGNED$.
- On input $(\mathsf{test}, pk, m, sig)$ and if pk does not exist exactly once in $KEYS$, both machines simply test the signature. Else both machines also output true only if the signature passes the cryptographic test, but $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ additionally requires that (pk, m) occurs in $SIGNED$. However, this sole difference in the views corresponds to signature forgery and has therefore negligible probability.

In more detail, the last step is proved in the following standard way: Let a probabilistic polynomial-time user machine H obtain distinguishable views when interacting with the two systems. By our considerations of all possible inputs, H achieves at least one different signature test output (in a run of the configuration) with not negligible probability (over the possible runs). Let $\mathsf{max\_keys}$ be a polynomial bounding the number of inputs $(\mathsf{generate})$ made by H. We construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signature oracle Sig_s: It chooses $i \xleftarrow{\mathcal{R}} \{1, \ldots, \mathsf{max\_keys}(k)\}$ and starts simulating H and all machines $\mathsf{Sig}_{u,s}$

and $\mathsf{Sig}_{\mathsf{low\_id},n,s}$, using the same keys and signatures in both scenarios. When H makes the i-th input ($\mathsf{generate}$), then $\mathsf{A}_{\mathsf{sig}}$ uses the key pk from the signature oracle instead of generating a key pair. The element sk in the resulting tuple $(u, sk, pk, 0) \in KEYS$ is set to $\downarrow$, and similarly in $KEYS_u$. From then on, $\mathsf{A}_{\mathsf{sig}}$ uses the signature oracle when signing with respect to this key tuple. Thus for each message m signed by the oracle there is a pair (pk, m) in the array $SIGNED$. Hence when H makes an input $(\mathsf{test}, pk, m', sig')$ where sig' is valid but $(pk, m') \notin SIGNED$, then $\mathsf{A}_{\mathsf{sig}}$ can output (m', sig') as a successful forgery in the sense of Definition 2. As the simulation is perfect, this happens with not negligible probability. This contradicts the signature security. Hence the views of H in the two systems are indeed indistinguishable.

6 Variants of the Low-level Ideal System

We now describe some possible variants of the low-level ideal system.

Memory-less version. If we only want to consider memoryless signatures, we can omit the parameter s and the counters from Definitions 6 and 7 and the proof. This is simple text extraction.

Fixed-length schemes. In reactive scenarios where surrounding protocols may employ encryption, length information about encrypted messages may leak. To make this manageable for encrypted message parts like signatures and public keys, it is useful to assume that for given parameters k and s, the length of signatures and public keys is fixed. This can be modeled by length functions $\mathsf{sig\_len}(k, s)$ and $\mathsf{pks\_len}(k)$ for the underlying signature scheme as in [14].

Polynomial time. The machines described are only weakly polynomial-time, i.e., polynomial-time in the overall length of inputs they received, and not strictly polynomial-time, i.e., in the security parameter k alone. In their typical intended application this does not matter. However, all machines can be made strictly polynomial-time by equipping them with (arbitrary polynomial-time) bounds on the length and number of accepted inputs at each port.

Scheduling. As the systems are currently described, both in- and outputs would be scheduled by the users. Instead, one can give the machines clock ports where they immediately schedule each of their outputs. This is advantageous to keep the state space small in higher-level proofs if signature-related operations are only used as local subroutines. If the systems were to be used remotely and in a larger scenario where users and adversaries have to be distinguished again, one could also take all the clock ports out of the set S of ports for the users.

Joint semi-real machine. If one intends to use a low-level idealization only once for proving a completely real implementation of a larger system with respect

to a higher-level idealization as in [14], it may be simpler for the overall proof to also write the low-level real system as one machine, because the fact that the overall real system can be rewritten with such a low-level system implies that the latter is real enough for the given purpose. We call this a semi-real system. This would correspond more closely to the treatment of encryption in [14]. Conversely, that encryption functionality could be rewritten from a semi-real version (one machine) to a real version if one omits the global key counter from the low-level ideal version.

7 Conclusion

We have considered signature schemes in a reactive scenario. Our first observation was that in many protocols, not all signatures become known to the adversary, and a usual assumption in protocol design is that an adversary cannot forge signatures that have been made but not revealed to him. We called security for this scenario "reactive security" and explored its relation to normal security against chosen-message attacks. It turned out that while general signature schemes can be insecure reactively, schemes whose memory (besides a key pair) is at most a counter are always reactively secure. For memory-less schemes this holds with almost unchanged concrete security, while for the general case with counters, we either lose a factor of s, the maximum number of signatures, in the success probability of an adversary in the reduction proof, or we have to additionally randomize the signature scheme.

We further introduced an idealization of signature schemes in the sense of reactive simulatability, which, at a low level of abstraction, makes the functionality of signature schemes usable in frameworks offering composition and property preservation theorems. In contrast to prior low-level idealizations, ours retains the property that the adversary cannot learn signatures that are not explicitly revealed to him from an underlying counter-based signature scheme.

Acknowledgments

We thank an anonymous reviewer for pointing out an improvement of the proof of Theorem 1 that allowed for a tighter reduction.

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

2. M. Backes. A cryptographically sound dolev-yao style security proof of the Otway-Rees protocol. In *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.
3. M. Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. *Journal of Logic and Algebraic Programming (JLAP)*, 2:157–188, 2005.
4. M. Backes and M. Duermuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.
5. M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004. Preprint on IACR ePrint 2003/240.
6. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.
7. M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proc. 7th European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2002.
8. M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proc. 24th IEEE Symposium on Security & Privacy*, pages 140–152, 2003.
9. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas of Computing (JSAC)*, 22(10):2075–2086, 2004.
10. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proceedings of 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004. Preprint on IACR ePrint 2004/059.
11. M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.
12. M. Backes and B. Pfitzmann. Relating cryptographic und symbolic secrecy. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(2):109–123, 2005.
13. M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.
14. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003.
15. M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. In *Proc. 6th Information Security Conference (ISC)*, pages 84–95, 2003.
16. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, 2003:15, 2003.
17. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive system. In *Proceedings of 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.
18. M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2004.

19. M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. *International Journal of Information Security (IJIS)*, 4(4):242–252, 2005.
20. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security (IJIS)*, 4(3):135–154, 2005.
21. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
22. D. Beaver. How to break a "secure" oblivious transfer protocol. In *Advances in Cryptology: EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 285–296. Springer, 1992.
23. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology: EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000.
24. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 419–428, 1998.
25. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
26. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.
27. R. Canetti. On universally composable notions of security for signature, certification and authorization. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004. Extended version in Cryptology ePrint Archive, Report 2003/239, `http://eprint.iacr.org/`.
28. R. Canetti and S. Goldwasser. An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 90–106. Springer, 1999.
29. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels (extended abstract). In *Advances in Cryptology: EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2002. Extended version in IACR Cryptology ePrint Archive 2002/059, `http://eprint.iacr.org/`.
30. R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.
31. R. Cramer and I. Damgård. Secure signature schemes based on interactive protocols. In *Advances in Cryptology: CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 1995.
32. R. Cramer and I. Damgård. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology: CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 1996.
33. A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (extended abstract). In *Proc. 1st ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 11–23, 2003.
34. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
35. C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *Journal of Cryptology*, 11(3):187–208, 1998.

36. R. Gennaro and S. Micali. Verifiable secret sharing as secure computation. In *Advances in Cryptology: EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 1995.
37. O. Goldreich. Secure multi-party computation. Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, June 1998, revised Version 1.4 October 2002, 1998. `http://www.wisdom.weizmann.ac.il/users/oded/pp.htm`.
38. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
39. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.
40. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
41. M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.
42. D. M. Johnson and F. Javier Thayer. Security and the composition of machines. In *Proc. 1st IEEE Computer Security Foundations Workshop (CSFW)*, pages 72–89, 1988.
43. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
44. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. 8th Symposium on Formal Methods Europe (FME 1999)*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer, 1999.
45. H. Mantel. On the composition of secure systems. In *Proc. 23rd IEEE Symposium on Security & Privacy*, pages 88–101, 2002.
46. D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. 8th IEEE Symposium on Security & Privacy*, pages 161–166, 1987.
47. D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.
48. J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. 15th IEEE Symposium on Security & Privacy*, pages 79–93, 1994.
49. J. McLean. A general theory of composition for a class of "possibilistic" security properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.
50. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.
51. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.
52. B. Pfitzmann. Sorting out signature schemes. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 74–85, 1993.
53. B. Pfitzmann. *Digital Signature Schemes – General Framework and Fail-Stop Signatures*, volume 1100 of *Lecture Notes in Computer Science*. Springer, 1996.
54. B. Pfitzmann and M. Waidner. A general framework for formal notions of "secure" systems. Research Report 11/94, University of Hildesheim, Apr. 1994. `http://www.semper.org/sirene/lit/abstr94.html#PfWa_94`.
55. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

56. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.
57. J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 387–394, 1990.
58. P. Syverson and C. Meadows. A logical language for specifying cryptographic protocol requirements. In *Proc. 14th IEEE Symposium on Security & Privacy*, pages 165–177, 2003.
59. A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.
60. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.