

Low-level Ideal Signatures and General Integrity Idealization

Michael Backes, Birgit Pfitzmann, and Michael Waidner

IBM Zurich Research Lab
{mbc,bpf,wmi}@zurich.ibm.com

Abstract. Recently we showed how to justify a Dolev-Yao type model of cryptography as used in virtually all automated protocol provers under active attacks and in arbitrary protocol environments. The justification was done by defining an ideal system handling Dolev-Yao-style terms and a cryptographic realization with the same user interface, and by showing that the realization is as secure as the ideal system in the sense of reactive simulatability. This holds the standard model of cryptography and under standard assumptions of adaptively secure primitives. While treating a term algebra is the point of that paper, a natural question is whether the proof could be more modular, e.g., by using a low-level idealization of signature schemes similar to the treatment of encryption.

We present a low-level ideal signature system that we tried to use as a lower layer in prior versions of the library proof. It may be of independent interest for cryptography because idealizing signature schemes has proved surprisingly error-prone. However, we also explain why using it makes the overall proof of the justification of the Dolev-Yao type model more complicated instead of simpler.

We further present a technique, integrity idealization, for mechanically constructing composable low-level ideal systems for other cryptographic primitives that have “normal” cryptographic integrity definitions.

1 Introduction

Automated proofs of security protocols with model checkers or theorem provers typically abstract from cryptography by deterministic operations on abstract terms and by simple cancellation rules. An example term is $E_{pk_{e_w}}(E_{pk_{e_v}}(\text{sign}_{sk_{s_u}}(m, N_1), N_2))$, where m denotes an arbitrary message and N_1, N_2 two nonces. A typical cancellation rule is $D_{sk_e}(E_{pk_e}(m)) = m$ for corresponding keys. The proof tools handle these terms symbolically, i.e., they never evaluate them to bitstrings. In other words, they perform abstract algebraic manipulations on trees consisting of operators and base messages, using the cancellation rules, the transition rules of a particular protocol, and abstract models of networks and adversaries. Such abstractions, although different in details, are called the Dolev-Yao model after the first authors [17].

For many years there was no cryptographic justification for such abstractions. The problem lies in the assumption, implicit in the adversary model, that actions that cannot be expressed with the abstract operations are impossible, and that no relations hold between terms unless derivable by the cancellation rules. It is not hard to make artificial counterexamples to these assumptions. Nevertheless, no counterexamples against

the method for protocols proved in the literature were found so far. Further, the overall approach of abstracting from cryptographic primitives once with rigorous hand-proofs, and then using tools for proving protocols using such primitives, is highly attractive: Besides the cryptographic aspects, protocol proofs have many distributed-systems aspects, which make proofs tedious and error-prone even if they weren't interlinked with the cryptographic aspects. To use existing efficient automated proof tools for security protocols, cryptography must indeed be abstracted into simple, deterministic ideal systems. The closer one can stay to the Dolev-Yao model, the easier the adaptation of the proof tools will be.¹

Cryptographic underpinnings of a Dolev-Yao model were first addressed by Abadi and Rogaway in [2]. However, they only handled passive adversaries and symmetric encryption. The protocol language and security properties handled were extended in [1, 23], but still only for passive adversaries. This excludes most of the typical ways of attacking protocols, e.g., man-in-the-middle attacks and attacks by reusing a message part in a different place or concurrent protocol run. A full cryptographic justification for a Dolev-Yao model, i.e., for arbitrary active attacks and within the context of arbitrary surrounding interactive protocols, was first given recently in [5]. Based on the specific Dolev-Yao model whose soundness was proven in [5] and called the *cryptographic library* there, the well-known Needham-Schroeder-Lowe protocol was proved in [3]. This shows that in spite of adding certain operators and rules compared with simpler Dolev-Yao models (in order to be able to use arbitrary cryptographically secure primitives without too many changes in the cryptographic realization), such a proof is possible in the style already used in automated tools, only now with a sound cryptographic basis. It was also shown how the cryptographic library, in other words the term algebra and rules, can be modularly extended by additional cryptographic primitives, using the example of symmetric authentication [8] and symmetric encryption [4]. Subsequent to the work of [5], several papers presented cryptographic underpinnings of Dolev-Yao models under active attacks for specific primitives, e.g., [24] for symmetric encryption and [20, 21, 27] for public-key encryption.

The full version of [5] with its rigorous proofs is of considerable length. This is not too surprising compared with, e.g., the length of [2]. Nevertheless, it seems an interesting question whether the cryptographic library, in other words the precise Dolev-Yao model used, as well as its proof could not be presented in a more modular way. There are several aspects to this question. We will discuss easy ones first and then come to the question of a more modular proof, which is the main motivation for this paper.

The trivial answer is that we could have left out some operators and then added them again in a separate paper as in [8]. Clearly the text would be much shorter if only encryption, application data, and lists would be retained as a minimum repertoire for building nested encryption terms of the Dolev-Yao style, or similarly for signatures. This would

¹ Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [28, 25, 29, 22] and, as a second step, to encode them into proof tools. However, this approach can not yet handle protocols with any degree of automation. Generally this approach should be seen as complementary to the approach of getting simple deterministic abstractions of cryptography and working with those wherever cryptography is only used in a blackbox way.

be a simple textual deletion of the subsections dealing with the other operators in the ideal system, the real system, the simulator, and the proof. However, the scientific credibility of the overall framework is indeed much clearer if at least two really different cryptographic systems are present. The reason is that the main point of such term algebras is to define the grammar of correct nested terms and cancellation rules, and to guarantee that terms that cannot be transformed into each other by cancellation rules are always unequal in reality. The facts that terms are type-safe across different cryptographic systems and that no unwanted cancellation can occur must be established by the overall framework, e.g., by defining how the simulator parses received nested abstract messages from the ideal system and received nested concrete terms received from the adversary (including that it cannot always parse them completely). One could also define and name sublibraries of [5], in other terminologies sub-algebras or sub-functionalities, corresponding to textual subsets as described in the previous paragraph. However, this is not much use, because a protocol designer needing only a subset of the operators is not bothered by the presence of additional operators.

However, another version of the modularization question is of more interest. This question is why the proof would not become simpler by using a low-level ideal signature system similar to the low-level ideal encryption system that is used. In the following, we show the functionality that we used in prior versions of the proof and why we removed it again, although it was correct in itself. By “low-level” we mean that the interface of the ideal system is not yet abstract in the sense needed for current automatic tools, and as in Dolev-Yao models. For encryption, such low-level ideal functionalities were introduced in [33, Section 5.3.1] and [10]. For signatures, formalizing and proving an ideal version is actually easier because their security property is an integrity property. Their established cryptographic definition is from [19]. It was known since [30, 31] that such properties can be formulated abstractly, e.g., in temporal logic. A similar formulation for authentication is known from [35], 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. We will call this the integrity idealization paradigm. Such an idealization was first made in [26] for symmetric authentication. A somewhat similar ideal signature system was presented in [10], with variations in [13, 14]. However, the precise approach taken there cannot be used to construct nested Dolev-Yao style terms, because while a term $E_{pk_e}(\text{sign}_{sk_s}(m))$ 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 in these ideal functionalities.

In the following, we present an ideal low-level signature system that could be used as a submodule in the cryptographic implementation of the library from [5]. However, we also show that using it would make the overall proof of that library (or of the addition of signature schemes to that library if one first restricted it to encryption) more complicated instead of simpler. While this argument necessarily depends on the proof technique used in [5], an important aspect depends solely on the simulator, and not on the details of the cryptographic bisimulation, so that it does not seem easy to circumvent. A low-level signature system similar to the one presented here was also developed in [11] based on initial joint discussions.

2 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.

2.1 Underlying Cryptographic Definition

As cryptographic primitive, we use a signature system secure against adaptive chosen-message attacks [19]. Further, we assume that it uses memory about previously signed messages only in the form of a counter. Besides memory-less signature schemes, this class comprises important provably secure signature schemes such as [19, 34, 18, 15, 16]. While efficient implementations of such signature schemes also store a path in a tree at any time, in theory such a path or any other function of random values chosen during earlier applications of sign_{sk} is equivalent to just a counter, because the random values can be regarded as part of the secret key sk and thus as chosen during key generation. At the same time, this class of signature schemes excludes pathologic cases that could not be used safely in a normal Dolev-Yao style library, e.g., if every signature divulges the history of all previous signatures with the same key. A proof that secure signature systems with this pathologic property exist and that they can make applications insecure is given in [7]. We summarize the GMR definition for this subclass in the following two definitions.

Definition 1. (*Signature Schemes*) A signature scheme is a triple $(\text{gen}, \text{sign}, \text{test})$ of polynomial-time algorithms, where gen and sign are probabilistic. gen takes an input $(1^k, 1^s)$ with $k, s \in \mathbb{N}$, where k denotes a security parameter and s the desired maximum number of signatures, and outputs a pair (sk, pk) of a secret signing key and a public test key in $\{0, 1\}^+$. sign takes such a secret key, a counter $c \in \{1, \dots, s\}$, and a message $m \in \{0, 1\}^+$ as inputs and produces a signature in $\{0, 1\}^+$. We write this $\text{sig} \leftarrow \text{sign}_{sk,c}(m)$. Similarly, we write verification as $b := \text{test}_{pk}(m, \text{sig})$ with $b \in \{\text{true}, \text{false}\}$. 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 . \diamond

Security of a signature scheme is defined against existential forgery under adaptive chosen-message attacks:

Definition 2. (*Signature Security*) Given a signature scheme $(\text{gen}, \text{sign}, \text{test})$ and a polynomial $s \in \mathbb{N}[x]$, a signature oracle \mathcal{O}_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 $(sk, pk) \leftarrow \text{gen}(1^k, 1^{s(k)})$ and output pk .
- On input (sign, m) with $m \in \{0, 1\}^+$, and if $c < s(k)$, set $c := c + 1$ and return $\text{sig} \leftarrow \text{sign}_{sk,c}(m)$.

The signature scheme is called existentially unforgeable under adaptive chosen-message attack if for every polynomial s and every probabilistic polynomial-time machine A_{sig} that interacts with O_s the following holds: The probability is negligible (in k) that A_{sig} finally outputs two values m and sig (meant as a forged signature for the message m) with $\text{test}_{pk}(m, \text{sig}) = \text{true}$ and where m is not among the messages previously signed by the signature oracle. \diamond

2.2 The Low-level Ideal System

We now present an ideal signature system that, at a low level of abstraction, summarizes the functionality guaranteed by the cryptographic definition. It uses a list *keys* of key tuples belonging to honest users and a list *signed* of message tuples honestly signed with these keys. Using lookup in these lists, it never accepts forgeries, i.e., signatures on other messages that are supposedly made with one of these keys.

We define this by an ideal machine whose honest users are, without loss of generality, numbered $\{1, \dots, n\}$. Its ports from and to user u are $\text{in}_{\text{sig},u}?$ and $\text{out}_{\text{sig},u}!$.²

Definition 3 (Low-level Ideal Signature Machine). Let a signature scheme $(\text{gen}, \text{sign}, \text{test})$ and parameters $n \in \mathbb{N}$ and $s \in \mathbb{N}[x]$ be given. A corresponding ideal signature machine $\text{Sig}_{\text{low.id},n,s}$ is defined as follows:

It maintains two initially empty lists *keys* and *signed*. The transition function is given by the following rules. Let u be the index of the port $\text{in}_{\text{sig},u}?$ where the current input occurs; the resulting output goes to $\text{out}_{\text{sig},u}!$.

- On input (generate): Set $(sk, pk) \leftarrow \text{gen}(1^k, 1^{s(k)})$, add the tuple $(u, sk, pk, 0)$ to the list *keys*, and output pk .
- On input (sign, pk, m): Retrieve a tuple $(u, sk, pk, c) \in \text{keys}$ with the given u and pk . If none or more than one exist or if $c = s(k)$, return the error symbol \downarrow . Else set $c := c + 1$, i.e., increase the signature counter for this key in *keys*. Then set $\text{sig} \leftarrow \text{sign}_{sk,c}(m)$, add the pair (pk, m) to the list *signed*, and output sig .
- On input (test, pk, m, sig): Retrieve a tuple $(v, sk, pk, c) \in \text{keys}$ with the given pk . If none or more than one exist, output $\text{test}_{pk}(m, \text{sig})$. Else if the pair (pk, m) exists in *signed*, then output $\text{test}_{pk}(m, \text{sig})$, else false.

Other inputs are ignored. We omit the indices n, s of $\text{Sig}_{\text{low.id},n,s}$ where they are irrelevant. \diamond

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 internally 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

² The representation of the Dolev-Yao-style library in [5] is based on the system model from [33], containing details of the state-transition model used for abstract functionalities and its realization by interacting Turing machines. Ports correspond to individual input or output tapes in the Turing machine realization. We use the same model here, but omit some notation although it would allow a more compact presentation.

key, and then it additionally verifies that the signature presented is valid. In the long version of this paper [6], we consider several variants of the low-level ideal machine for capturing, e.g., memory-less signature schemes, schemes with fixed-length keys, schemes containing explicit polynomial bounds, or scheduling variants to model local sub-routine behavior.

The main difference to the signature functionality in [10] is that the adversary learns nothing about what honest users sign. In the notation from [33] used here, this would show up as outputs at an adversary port $\text{out}_{\text{sig},a}!$ during individual transitions, e.g., an output (m, sig) during signing.

The low-level ideal machine formally depends on the real system, like the first instantiation of this paradigm in [26]. This could be alleviated by the technique from [12] 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, e.g., like the algorithm-dependent low-level encryption idealization is used to prove the algorithm-independent cryptographic library in [5]. 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 [5].

2.3 Cryptographic Realization and Security

The claimed 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 4 (Real Signature Machines). *Let a signature scheme $(\text{gen}, \text{sign}, \text{test})$ and parameters $u \leq n \in \mathbb{N}$ and $s \in \mathbb{N}[x]$ be given. Then the low-level ideal signature machine $\text{Sig}_{u,s}$ is defined as follows. It has ports $\text{in}_{\text{sig},u}?$ and $\text{out}_{\text{sig},u}!$ and maintains an initially empty list keys_u . The transition function is given by the following rules.*

- On input (generate): Set $(sk, pk) \leftarrow \text{gen}(1^k, 1^{s(k)})$, add the tuple $(sk, pk, 0)$ to keys_u , and output pk .
- On input (sign, pk, m): Retrieve a tuple $(sk, pk, c) \in \text{keys}_u$ with the given pk . If none or more than one exist or if $c = s(k)$, return \downarrow . Else set $c := c + 1$ and output $\text{sig} \leftarrow \text{sign}_{sk,c}(m)$.
- On input (test, pk, m, sig), output $\text{test}_{pk}(m, \text{sig})$.

Other inputs are ignored. We denote the set of these machines by $\text{Sig}_{\text{real},n,s}$ and omit the indices n, s (also for $\text{Sig}_{u,s}$) where they are irrelevant. \diamond

We now claim that the real signature system is as secure as the low-level ideal system. In slight abuse of notation of [33], we can write this as follows.

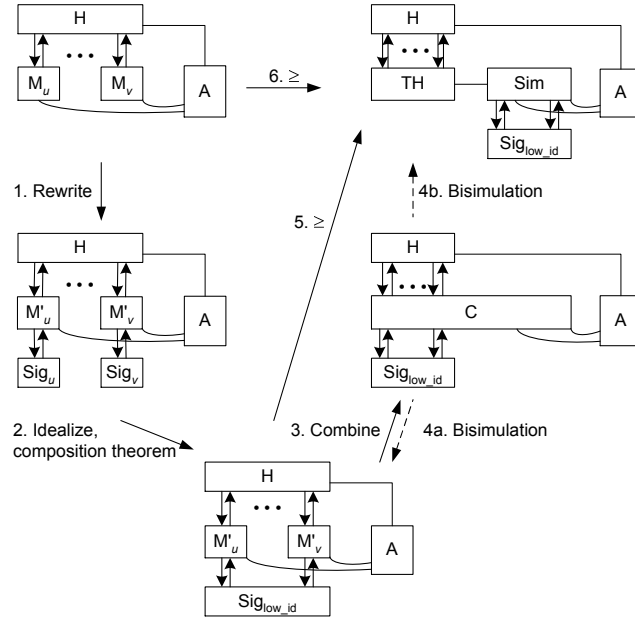


Fig. 1. Overview of a potential proof of the Dolev-Yao-style library with signature machines

Theorem 1. *Given a secure signature system according to Definitions 1 and 2, we have*

$$\forall n \in \mathbb{N} \forall s \in \mathbb{N}[x] : \text{Sig}_{\text{real},n,s} \geq^{\text{poly}} \text{Sig}_{\text{low_id},n,s}.$$

This holds with blackbox simulatability; actually no simulator is necessary which corresponds to the notion of observational equivalence as introduced in [25]. \square

We omit the proof due to space constraints and refer to the long version of this paper [6].

3 Using the Low-level Ideal Signature System in a Dolev-Yao-style Cryptographic Library

We now discuss why the use of low-level idealized signature machines in the proof of the Dolev-Yao style cryptographic library in [9] does not work quite as expected and actually makes the proof more complicated.

The proof of the Dolev-Yao-style library is based on a simulator. For all polynomial-time honest users and adversaries on the real system, the simulator achieves essentially the same effect in the ideal system. More precisely, it achieves that the views of the honest users in both systems are indistinguishable. This is the standard blackbox technique for showing that a system is as secure as another one in a sense that guarantees composability, as first formalized in detail for reactive, i.e., multi-step systems in [32].

A possible use of low-level ideal signature machines in the overall proof is shown in Figure 1. First the real cryptographic library is rewritten to use the real signature ma-

chines Sig_u (Step 1 in Figure 1). This happens after the step of rewriting with encryption machines from [9], which we omit in Figure 1.³ Next, the real signature machines Sig_u can jointly be replaced by their low-level ideal counterpart $\text{Sig}_{\text{low_id}}$ according to a composition theorem, in this case of [33] (Step 2 in Figure 1). The simulator can immediately be defined with the signature machine $\text{Sig}_{\text{low_id}}$ (upper right system in Figure 1).

The major part of the proof shows that the simulator is correct, both in general aspects like parsing, type determination, and handling of unparsable terms, and in handling the individual cryptographic operations. For this, we first define a combined system C that essentially has the combined ideal and real state space (Step 3 in Figure 1). Now it would also get the signature submachine. Then so-called cryptographic bisimulations are shown between the combined system and the real system with the signature and encryption machines, and with the ideal system with the simulator (Steps 4a and 4b in Figure 1). These bisimulations include a definition of error sets, containing runs where the simulation does not work. They also include an embedded information-flow analysis whose necessity will become clear in Section 3.2. Finally, cryptographic reduction proofs show that the error sets are negligible, based on the information-flow analysis. This yields that the rewritten real system is as secure as the ideal system (Step 5 in Figure 1), and by transitivity of “as secure as” this also holds for the original real system (Step 6 in Figure 1).

3.1 More Complicated Bisimulation by Diverging States

Our main argument for deciding to delete the signature machines again was that, surprisingly to us grown up with the idea that modularization simplifies things, they make the bisimulation more complicated. A long part of the proof in [5] is the standard aspects of the bisimulations. This comprises the definition of mappings from combined states to real and ideal states, respectively, as well as invariants of all three systems (combined, real and ideal). It is then shown that, given mapped states fulfilling the invariants, every input from the adversary or honest users leads to equal outputs and to mapped next states with equal probability distributions, and that the invariants are retained (except for the error sets).

Usually, the more machines one has in a bisimulation, the more complicated the state spaces and invariants get, i.e., modularization is typically not useful. Nevertheless, one might hope that introducing low-level ideal signature machines is useful because their states would simply be mapped identically, and their state transitions would trivially retain this mapping.⁴ However, this is not so. The individual signatures are not made in the same order in the simulator as in the real system. For instance, honest user u might make a signature sig on a message m , encrypt it as $E_{pk}(sig)$ with a key pk of another honest user, and send it over an insecure channel. Now the signature exists in $\text{Sig}_{\text{low_id}}$ in the real system, i.e., a counter has been updated and the message m is stored in $signed$. However, the simulator only gets an abstract ciphertext from the ideal

³ Both these steps are obsolete if one defines the real library directly with these functionalities, but we first presented an entirely real version in [5] for concreteness.

⁴ However, even in this case one would need invariants about the consistency between the state of the overall “term machines” and the signature machines.

system and simulates it by $E_{pk}(m_{sim})$ for a fixed message m_{sim} (using its low-level ideal encryption system). There is no way for the simulator to know at this point that it should simulate a signature. The signature will only be simulated if it is ever sent in a form readable for the adversary.

Hence although the signature subsystems in the overall systems to be compared are functionally equal, they are usually in different states. Hence they are just an additional burden on the bisimulation.⁵

3.2 Other Error Sets and Information-Flow Analysis Remain

The low-level idealization of signatures certainly avoids the error set corresponding to signature forgeries in the bisimulations, and the reduction proof showing that this set is negligible. However, this is a short and simple part of the overall proof.

Avoiding error sets would mainly be useful if one could get rid of all of them, and thus have “classical” probabilistic bisimulations. Towards this goal, one could introduce a low-level ideal nonce system that excludes nonce collisions, while still outputting real nonces. This system falls under the integrity-idealization paradigm: Equip a real nonce system with a virtual, global non-repetition test over the nonces of all honest parties.

However, one also has to demonstrate that the adversary cannot guess certain values. Here the low-level ideal systems do not help. For instance, with both real nonces and the low-level ideal nonce system, it is trivial that the adversary cannot guess a nonce of an honest participant if he obtained no information about it (except with exponentially small probability). But whether or not he obtained such information depends on all potential nested terms that were sent containing this nonce, i.e., this belongs to the overall proof and not to the proof of the subsystem. Concretely, this proof aspect is the static information-flow analysis embedded in the cryptographic bisimulation, a novel proof technique in [9].

In other words, idealizing signatures and nonces only eliminates the easier parts of the final reduction proofs and of the non-standard aspects of the bisimulations.

3.3 Simulator Needs Reordered Signatures

The example in Section 3.1 also shows that the simple low-level ideal system from Section 2.2, derived from the GMR definition by integrity idealization, is not quite the functionality we need in a reactive scenario.

If one only considers the real system, the following additional property is needed: If the adversary only sees a subset (adaptively chosen) of the signatures made by a signer, this divulges nothing about other signatures. This is already discussed in [7]. It is shown there that this property follows from the GMR definition as restricted in our Definition 1, but that the concrete security can be improved by additional randomization. In the cryptographic library of [5], such an additional randomization is present in the real system anyway.

⁵ Comparing Figure 1 with the corresponding figure in [5], one sees that encryption machines were not used in the simulator and the combined system. This is for the same reason of diverging states.

For proceeding as in Section 3, one also has to consider the simulator. The example in Section 3.1 shows that the simulator has to make signatures with an arbitrary non-repeating sequence of counter values. Thus, to define the simulator with a low-level ideal signature system, that system must take the counter value as an input, instead of incrementing it internally as done in reality (this problem disappears for memory-less underlying signature systems). Hence for the overall proof technique with signature submachines, we must prove that the underlying signature schemes are secure for this behavior. (The proof in [5] does not need this aspect of a signature scheme.) We are not aware that this follows from Definition 2, although we believe that it holds for all known systems (because the tree constructions are essentially symmetric if one considers all randomness as chosen in advance, e.g., the third leaf does not depend on the first leaf any more than vice versa). Nevertheless, this is not easy to point out in, say, the proof in [19].

4 Outlook on Integrity Idealization Paradigm

We have repeatedly mentioned integrity idealization as a common concept for defining certain low-level ideal functionalities. In the long version of this paper, we show that this concept can be formalized and used to mechanically construct many simple ideal systems and the corresponding proofs. Due to space constraints, we only give the basic ideas of this formulation here: Typically, a cryptographic primitive is defined as a tuple of algorithms (A_1, \dots, A_t) , such as (gen, sign, test) for signatures, with certain parameter domains. The security definition is typically given by an “oracle” (such as the signature oracle O_s) interacting with an adversary, where an oracle call corresponds to an algorithm invocation. A typical integrity property can be (re-)written as a property of the trace of in- and outputs of the oracle, where the first violation can only occur by an output of the oracle. This makes sense particularly for cases like signatures where the adversary only wins if he can successfully cheat an honest participant with a forgery. Other examples of such integrity properties are all other authentication properties, the collision-freeness of a nonce system, and the correctness properties of a payment system. For a system defined by such integrity requirements, we define the corresponding ideal system by one joint machine, corresponding closely to the oracle, that interacts with the honest users and the adversary and that, before every output, verifies that the output does not violate the integrity properties. Thus it fulfills the integrity properties perfectly by definition. This gives a general underpinning for using integrity properties in proofs that otherwise use a composition theorem, since an arbitrary real system fulfilling an arbitrary integrity property is automatically as secure as a mechanically derivable low-level idealization.

5 Conclusion

We have presented a low-level ideal signature functionality that can be realized by every secure signature scheme that uses memory only for a counter and random values, without additional techniques like padding or randomization.

However, we also showed multiple pitfalls when using such a low-level idealization for proving a larger protocol. While we showed this for the specific example of our cryptographic library, we believe that the problems are of a general nature. For instance, every protocol where some signatures are only sent in encrypted form has the problem from Section 3.1, and if such signatures become known to the adversary later in a different order than they were made, the problems from Section 3.3 are added. (Recall that prior low-level ideal signature systems cannot handle this case at all.) In many cases, it will be simpler to work either with a real abstraction, such as the cryptographic library from [5], or directly with the integrity properties.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.
3. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2003.
4. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.
5. 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. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003.
6. M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. Research Report RZ 3511, IBM Research, Nov. 2003.
7. M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. In *Proc. 6th Information Security Conference (ISC)*, pages 84–95, 2003.
8. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *LNCS*, pages 271–290. Springer, 2003.
9. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003.
10. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.
11. R. Canetti. On universally composable notions of security for signature, certification and authorization. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.
12. 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 *LNCS*, pages 90–106. Springer, 1999.
13. 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 *LNCS*, pages 337–351. Springer, 2002.
14. R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, 2003.

15. R. Cramer and I. Damgård. Secure signature schemes based on interactive protocols. In *Advances in Cryptology: CRYPTO '95*, volume 963 of *LNCS*, pages 297–310. Springer, 1995.
16. R. Cramer and I. Damgård. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology: CRYPTO '96*, volume 1109 of *LNCS*, pages 173–185. Springer, 1996.
17. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
18. C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *Journal of Cryptology*, 11(3):187–208, 1998.
19. 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.
20. J. Herzog. Computational soundness of formal adversaries. Master Thesis, MIT, 2002.
21. J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *LNCS*, pages 548–564. Springer, 2003.
22. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science*, pages 372–381, 2003.
23. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
24. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.
25. 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.
26. 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 *LNCS*, pages 776–793. Springer, 1999.
27. 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 *LNCS*, pages 133–151. Springer, 2004.
28. J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science*, pages 725–733, 1998.
29. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
30. B. Pfitzmann. Sorting out signature schemes. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 74–85, 1993.
31. B. Pfitzmann. *Digital Signature Schemes – General Framework and Fail-Stop Signatures*, volume 1100 of *LNCS*. Springer, 1996.
32. 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.
33. 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.
34. 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.
35. 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.