RZ 3511 (# 99529) 11/10/03
Computer Science 14 pages

Research Report

Low-level Ideal Signatures and General Integrity Idealization

Michael Backes, Birgit Pfitzmann and Michael Waidner

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Low-level Ideal Signatures and General Integrity Idealization

Abstract

Recently Backes, Pfitzmann and Waidner solved the long-standing open problem of justifying a Dolev-Yao type model of cryptography as used in virtually all automated protocol provers. They define a flexible toolbox for constructing abstract nested cryptographic terms and for using them in arbitrary protocols, together with a cryptographic realization provably secure under arbitrary active attacks in the standard model of cryptography. 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 could be used as a lower layer in this toolbox. 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 toolbox 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.

Keywords: Digital Signatures, Composability, Dolev-Yao, Formal Methods, Integrity Idealization

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 $\mathsf{E}_{pke_w}(\mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u}(m, N_1), N_2))$, where m denotes an arbitrary message and N_1, N_2 two nonces. A typical cancellation rule is $\mathsf{D}_{ske}(\mathsf{E}_{pke}(m)) = m$ 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 [14].

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]

[1]Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time consid-

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 where extended in [1, 18], 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], the well-known Needham-Schroeder-Lowe protocol was proved in [4]. 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. The authors also showed how the library, in other words the term algebra and rules, can be modularly extended by additional cryptographic primitives, using the example of symmetric authentication [7].

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.[2] 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 the authors could have left out some operators and then add them again in a separate paper as in [7]. 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 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. Although that might have been smarter (as least publishable units and given impatient readers), 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 subfunctionalities, 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, just as a programmer isn't bothered by additional systems in a real implementation of a cryptographic library she is using. Actually, the protocol

erations, in particular [21, 19, 22, 17] 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 it should be seen as complementary to, rather than competing with, the approach of getting simple deterministic abstractions of cryptography and working with those wherever cryptography is only used in a blackbox way.

[2]The non-anonymous version will contain acknowledgments to others who also asked this question. Let us only declare that we asked ourselves this question and had a written version of essentially the functionality presented here before any communication about it with others.

proof in [4] already shows this. It also shows how one can omit aspects like key distribution in such proofs if one does not care. Even if the designer of a theorem prover wanted to use only some of the operators, that would be easy, but the value of a theorem prover lies in its range of implemented theories, and we currently do not see that the tools would be easier to adapt for some operators in [5] than for others. Similarly, in coding the real (cryptographic) implementation of that term algebra, one could code only a subset of the operators, or bind part of the code to a protocol implementation, but usually a library with more primitives is more useful in practice.

However, another version of the modularization question is of more interest, and no answer to it can be derived from the text of [8] and its successor papers. 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, and possibly even using a pre-existing one.

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 [26, 9]. 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 [16]. It was known since [23, 24] that such properties can be formulated abstractly, e.g., in temporal logic. A similar formulation for authentication is known from [28], 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 [20] for symmetric authentication. A somewhat similar ideal signature system was presented in [9], with variations in [10, 11]. However, the precise approach taken in [9] cannot be used to construct nested Dolev-Yao style terms, because while a term $E(pke, S(pks, m))$ 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 [9]. This aspect was not changed in [10, 11].

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.

We further present a general technique "integrity idealization" to mechanically construct composable ideal systems for certain types of cryptographic primitives that have "normal" cryptographic integrity definitions together with a proof sketch that these idealizations are automatically secure.

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 [16]. 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 [16, 27, 15, 12, 13]. 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 [6]. We summarize the GMR definition for this subclass in the following two definitions.

Definition 2.1 *(Signature Schemes) A* signature scheme *is a triple* $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *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* $c \in \{1, \ldots, s\}$*, and a message* $m \in \{0,1\}^+$ *as inputs and produces a signature in* $\{0,1\}^+$*. 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*.* $\diamond$

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

Definition 2.2 *(Signature Security) Given a signature scheme* $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *and a polynomial* $s \in \mathbb{N}[x]$*, a* signature oracle 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* $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$ *and output* pk*.*
- *On input* (sign, m) *with* $m \in \{0,1\}^+$*, and if* $c < s(k)$*, set* $c := c + 1$ *and return* $sig \leftarrow \mathsf{sign}_{sk,c}(m)$*.*

The signature scheme is called existentially unforgeable under adaptive chosen-message attack *if for every polynomial* s *and every probabilistic polynomial-time machine* $\mathsf{A}_{\mathsf{sig}}$ *that interacts with* O_s *the following holds: The probability is negligible (in* k*) that* $\mathsf{A}_{\mathsf{sig}}$ *finally outputs two values* m *and* sig *(meant as a forged signature for the message* m*) with* $\mathsf{test}_{pk}(m, sig) = \mathsf{true}$ *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 $\{1, ..., n\}$. Its ports from and to user u are $\mathsf{in}_{\mathsf{sig},u}?$ and $\mathsf{out}_{\mathsf{sig},u}!$.[3]

[3]The representation of the Dolev-Yao-style library in [5] is based on the system model from [26], 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.

Definition 2.3 (Low-level Ideal Signature Machine) *Let a signature scheme* $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ *and parameters* $n \in \mathbb{N}$ *and* $s \in \mathbb{N}[x]$ *be given. A corresponding ideal signature machine* $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ *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* $\mathsf{in}_{\mathsf{sig},u}?$ *where the current input occurs; the resulting output goes to* $\mathsf{out}_{\mathsf{sig},u}!$*.*

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

Other inputs are ignored. We omit the indices n, s *of* $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ *where they are irrelevant.* ◇

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 key, and then it additionally verifies that the signature presented is valid.

The main difference to the signature functionality in [9] is that the adversary learns nothing about what honest users sign. In the notation from [26] used here, this would show up as outputs at an adversary $\mathsf{out}_{\mathsf{sig,a}}!$ during individual transitions, e.g., an output (m, sig) during signing.

2.3 Cryptographic Realization

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

- *On input* $(\mathsf{generate})$*: Set* $(sk, pk) \leftarrow \mathsf{gen}(1^k, 1^{s(k)})$*, add the tuple* $(sk, pk, 0)$ *to* $keys_u$*, and output* pk*.*
- *On input* (sign, pk, m)*: Retrieve a tuple* $(sk, pk, c) \in keys_u$ *with the given* pk*. If none is found or* $c = s(k)$*, return* $\downarrow$*. Else set* $c := c + 1$ *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 by $\mathsf{Sig}_{\mathsf{real},n,s}$*, and omit the indices* n, s *(also for* $\mathsf{Sig}_{u,s}$*) where they are irrelevant.* ◇

2.4 Security

We now claim that the real signature system is as secure as the low-level ideal system. In slight abuse of notation of [26] (explained in Section 3), we can write this as follows.

Theorem 2.1 *Given a secure signature system according to Definitions 2.1 and 2.2, we have*

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

This holds with blackbox simulatability; actually no simulator is necessary. □

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 as usual in the model of [26].)

The proof uses and shows the following invariant: at any time, each list $keys_u$ of a real machine $\mathsf{Sig}_{u,s}$ can be derived from the list *keys* of the ideal machine by restricting *keys* to entries with first parameter u and then deleting the first parameters. All entries of *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 systems generate keys and output pk in the same way. The way they add to their lists $keys_u$ 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 a tuple $(sk, pk, c) \in keys_u$ if and only if $\mathsf{Sig}_{\mathsf{low\_id},n,s}$ finds $(u, sk, pk, c) \in keys$ (with the same sk and c). Both return $\downarrow$ if none is found or $c = s(k)$. 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 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 is therefore negligible.

In more detail, the last step is proved in the following standard way: Let a probabilistic polynomial-time 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 system run) with non-negligible probability (over the possible runs of the system). Let maxkey be a polynomial bounding the number of inputs $\mathsf{generate}$ that H makes. We construct an adversary $\mathsf{A}_{\mathsf{sig}}$ against the signature oracle: It chooses $i \overset{\mathcal{R}}{\leftarrow} \{1, \ldots, \mathsf{maxkey}(k)\}$ and starts simulating H and all machines $\mathsf{Sig}_{u,s}$ and $\mathsf{Sig}_{\mathsf{low\_id},n,s}$. When H makes the i-th input $\mathsf{generate}$, then $\mathsf{A}_{\mathsf{sig}}$ uses the signature oracle. The element sk in the resulting tuple $(u, sk, pk, c) \in keys$ remains empty, 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 list *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 2.2. As this happens with not negligible probability, it contradicts the signature security. Hence the systems are indeed indistinguishable. ■

3 Other Versions of the Low-level Ideal System

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

Overall System Definition. We first explain in which sense Theorem 2.1 was simplified and why this is no problem. First, the model from [26] distinguishes so-called specified and unspecified ports of a set of machines. Honest users only link to the specified ports. In our case, all ports are specified, i.e., there are no special outputs to or inputs from the adversary. Secondly, simulatability is defined (and composition and property-preservation theorems are proved) for so-called systems. Our sets $\mathsf{Sig}_{\mathsf{real},n,s}$ and $\{\mathsf{Sig}_{\mathsf{low\_id},n,s}\}$, together with the set of specified ports, are each one possible structure in a system derived from an intended structure and a trust model in a standard way.

Scheduling. As the systems are currently described, outputs would be clocked by the adversary. Instead both systems can be equipped with clock ports by which they can schedule their own outputs. Similarly, putting input clock ports into the set of specified ports lets the users schedule the inputs. This is advantageous to keep the state space small in higher-level proofs if signature-related operations are only used as local subroutines.

Memory-less version. If we only want to consider memoryless signatures, we can omit the counter and the parameter s from Definition 2.1 and everything dealing with them in the following. This is simple text extraction.

Fixed-length schemes. In a cryptographic library that allows arbitrary (polynomial) long messages, encryption leaks length information. To make this manageable for the case of 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 original signature scheme as in [5].

Polynomial time. The machines described are not polynomial-time by themselves. In their typical intended application this does not matter. However, all machines can be made polynomial-time by equipping them with (arbitrary polynomial-time) bounds on the length and number of accepted inputs at each port.

Joint semi-real machine. If one intends to use a low-level idealization only once for proving a higher-level idealization with a given, completely real version as in [5], it may be simpler for the overall proof to also write the real system as one machine, because the fact that the overall real system can be rewritten with the low-level semi-real system implies that it is real enough for the given purpose. This would correspond more closely to the treatment of encryption in [5]. Conversely, the real version of 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.

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

The proof of the Dolev-Yao-style library in [8] 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 systems in [25].

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 machines Sig_u (Step 1 in

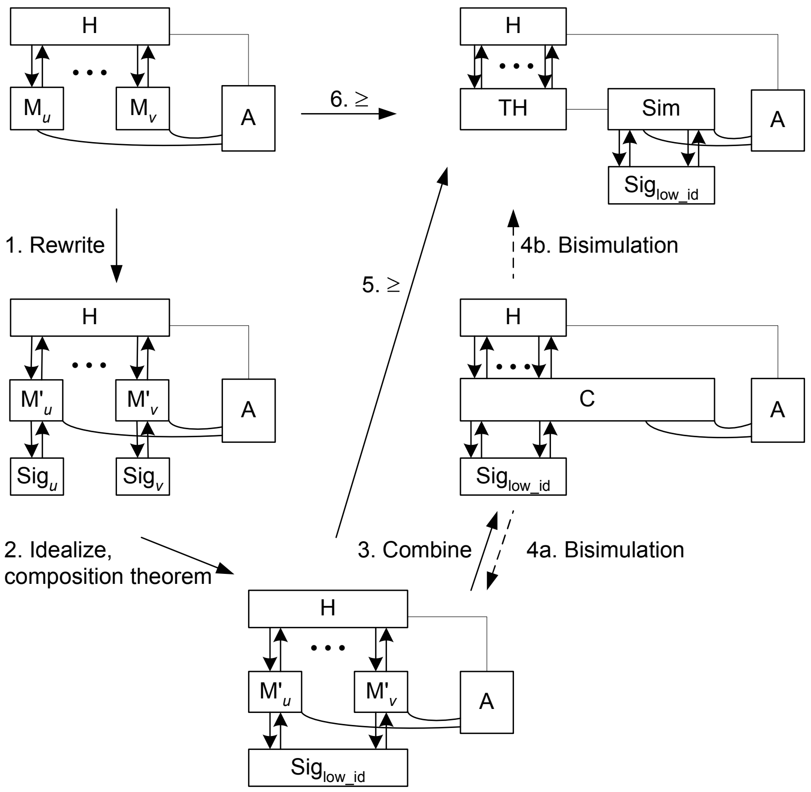

Figure 1: Overview of a potential proof of the Dolev-Yao-style library with signature machines

Figure 1). This happens after the step of rewriting with encryption machines from [8], which we omit in Figure 1.[4] Next, the real signature machines Sig_u can jointly be replaced by their low-level ideal counterpart $\mathsf{Sig}_{\mathsf{low\_id}}$ according to a composition theorem, in this case of [26] (Step 2 in Figure 1). The simulator can immediately be defined with the signature machines $\mathsf{Sig}_{\mathsf{low\_id}}$ (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, [8] first defines a combined system C that essentially has the combined ideal and real state space (Step 3 in Figure 1). Now it would also get signature submachines. 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 contain a definition of error sets, containing runs where the simulation does not work. It also contains an embedded information-flow analysis whose necessity will become clear in Section 5.1. 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).

[4]Both these steps are obsolete if one defines the real library based on these functionalities in the first place, but [8] first presents an entirely real version, presumably for concreteness.

5 Discussion

We now discuss why the use of low-level idealized signature machines in the proof of the Dolev-Yao style cryptographic library, as described in Section 4, does not work quite as expected and actually makes the proof more complicated.

5.1 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, an important novel proof technique in [8].

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

5.2 More Complicated Bisimulation by Diverging States

Another 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.[5] 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 $\mathsf{E}_{pk}(sig)$ with a key pk of another honest user, and send it over an insecure channel. Now the signature exists in $\mathsf{Sig}_{\mathsf{low\_id}}$ 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 system and simulates it by $\mathsf{E}_{pk}(m_{sim})$ 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.

[5]However, even in this case one would need invariants about the consistency between the state of the overall "term machines" and the signature machines.

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.[6]

5.3 Simulator Needs Reordered Signatures

The example in Section 5.2 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 [6]. It is shown there that this property follows from the GMR definition as restricted in our Definition 2.1, 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.

For proceeding as in Section 4, one also has to consider the simulator. The example in Section 5.2 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.[7] 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 2.2, 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 [16].

6 Integrity Idealization Theorem

We have repeatedly mentioned integrity idealization as a common concept for defining certain low-level ideal functionalities. We now show that this concept can be formalized and used to mechanically construct many simple ideal systems and the corresponding proofs.

Typically, a cryptographic primitive is defined as a tuple of algorithms $(A_1, \ldots, A_t)$, such $(\mathsf{gen}, \mathsf{sign}, \mathsf{test})$ 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. For instance, Definition 2.2 is not yet in this form because there is the "dangling" output (m, sig) of the adversary at the end, but it can be rewritten into this form by making that output an input to the oracle and by letting the oracle verify that sig is a valid signature and m a new message. 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.

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

[7]This problem disappears for memory-less underlying signature systems. However, as long as memory-less provably secure systems under standard assumptions are less efficient than their counterparts with memory, it is interesting to be able to handle systems with memory, too.

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 fulfils the integrity properties perfectly by definition.

To present this approach as a rigorous theorem, we assume that the real systems and the security properties are already described in a certain model. (In other words, we do not aim at mechanizing the translation of an arbitrary rigorous but textual definition of algorithms, parameters, oracles, and security properties into a particular model.) As in the rest of the paper, we use the model of [26]. Integrity properties and their cryptographic fulfillment were already defined for it in [3]. As that definition is geared towards a preservation theorem from ideal systems to real systems, it only deals with events at the specified ports of a system, i.e., those that honest users can connect to and that must be equal in the ideal and the real system. We generalize this to arbitrary ports of the machines, although integrity requirements involving only the specified ports will be the most common case.[8]

Briefly, an integrity requirement Req for a set M of machines is a set of traces (i.e., possible event sequences) over the set of ports of M. We say that Req is input-closed if with every trace $t \in \mathit{Req}$, also $\mathsf{append}(t, i) \in \mathit{Req}$ for all inputs i.

As multiple integrity requirements on one machine set can be combined into one by intersection, we only consider one.

Definition 6.1 *Let a set M of machines and an input-closed integrity property Req for M be given. Then we define the integrity-idealized machine M_{Req} as follows: Let M_{com} be the combination of the given machine set M (as defined in [26]).[9] Now M_{Req} acts like M_{com}, but before every output, it verifies that this output retains the integrity property Req. If not, it stops. For this verification, M_{Req} always keeps track of its prior trace t of all in- and outputs. Given the new potential output o, it evaluates whether $\mathsf{append}(t, o) \in \mathit{Req}$.* $\diamond$

If we want to make M_{Req} polynomial-time or weakly polynomial-time (i.e., polynomial-time in the length of its overall inputs), we must presuppose that membership in Req is decidable in polynomial time.

The definition can be canonically lifted to systems consisting of several structures (as mentioned in Section 3); formally each structure is a pair (M, S) of a set M of machines and a set S of specified ports. Conversely, the definition of a system fulfilling a requirement from [3] can immediately be specialized to one structure, and generalized to an arbitrary integrity requirement (i.e., not only involving specified ports). We denote this by $(M, S) \models \mathit{Req}$, where "$\models$" may get a superscript perf, SMALL, or poly for perfect, statistical or computational fulfillment, respectively. Essentially, the three notions mean that in all valid configurations $(M, S, \mathsf{H}, \mathsf{A})$ of (M, S), i.e., all combinations with a suitable honest user and adversary, the probability that the restriction of the resulting run r to the ports of M does not lie in Req, in formulas $r\lceil_{\mathsf{ports}(M)} \notin \mathit{Req}$, is zero, or in a class SMALL, or negligible (as a function of a security parameter k and in the probability space of runs defined for each k).

The following theorem says that if a structure (M, S) fulfils a requirement Req, then it is as secure as the corresponding integrity-idealized structure (M_{Req}, S). Here "$\geq$" denotes "as secure

[8]We can also generalize it to involve internal states without significant changes in the following definition and theorem. However, this requires more new notation. Further, one can also transform relevant internal state into events at ports in standard ways.

[9]The machines must be combined because the integrity condition is usually not locally verifiable. Otherwise one would not need a cryptographic implementation of this distributed functionality.

as", the simulatability definition guaranteeing composability, and can get the same superscripts as "$\models$" if one specifically wants to denote the perfect, statistical or computational case.

Theorem 6.1 *If $(M, S) \models Req$, and M_{Req} is constructed according to Definition 6.1, then $(M, S) \geq (M_{Req}, S)$. This holds for the perfect, statistical, and computational case, and with black-box simulatability. Actually, no simulator is needed.* □

In some sense, the theorem may seem trivial, just like the individual idealizations of signature systems, but as simulatability implies secrecy aspects, it is worth presenting a short proof.

Proof. (Sketch) We show that for every configuration $(M, S, \mathsf{H}, \mathsf{A})$ of (M, S), the corresponding configuration $(M_{Req}, S, \mathsf{H}, \mathsf{A})$ is indistinguishable. Each run r of $(M, S, \mathsf{H}, \mathsf{A})$ can trivially be mapped to the same run r of $(M_{Req}, S, \mathsf{H}, \mathsf{A})$, except if M_{Req} would suppress an output and stop in that run. Thus the view of H can differ at most for runs with $r \notin Req$, i.e., runs that do not fulfill the integrity requirement of (M, S). This happens with zero, small or negligible probability given perfect, statistical or computational integrity fulfillment.

Hence the view of H in the two configurations is perfectly, statistically or computationally indistinguishable, respectively. ■

An integrity-idealized system formally depends on the real system, like the first instantiation of this paradigm in [20]. In individual cases this could probably always be alleviated by the technique from [9] of letting the adversary choose the algorithms, so that the overall low-level ideal functionality comprises all possible instantiations. We have not worked out what this would mean at the level of generality of Theorem 6.1, and 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 Dolev-Yao style 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].

7 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 a cryptographic library enabling Dolev-Yao-style nested terms, 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 5.2, and if such signatures become known to the adversary later in a different order than they were made, the problems from Section 5.3 are added. (Recall that prior low-level ideal signature systems cannot handle this case at all.) In most 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.

As a general underpinning for using integrity properties in proofs that otherwise use a composition theorem, we showed that an arbitrary real system fulfilling an arbitrary integrity property is automatically as secure as a mechanically derivable low-level idealization.

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 *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[3] 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.

[4] 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)*, 2003. Extended version in IACR Cryptology ePrint Archive 2003/121, `http://eprint.iacr.org/`.

[5] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Extended version in IACR Cryptology ePrint Archive 2003/015, `http://eprint.iacr.org/`.

[6] M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. In *Proc. 6th Information Security Conference (ISC)*, pages 84–95, 2003.

[7] 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)*, pages 271–290, 2003. Extended version in IACR Cryptology ePrint Archive 2003/145, `http://eprint.iacr.org/`.

[8] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003. `http://eprint.iacr.org/`.

[9] 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.

[10] R. Canetti and H. Krawczyk. Universally composable key exchange and secure channels. In *Advances in Cryptology: EUROCRYPT 2002*, pages 337–351, 2002.

[11] R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology: CRYPTO 2003*, 2003.

[12] 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.

[13] 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.

[14] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[15] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *Journal of Cryptology*, 11(3):187–208, 1998.

[16] 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.

[17] R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[18] P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[19] 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.

[20] 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.

[21] 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 (FOCS)*, pages 725–733, 1998.

[22] J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.

[23] B. Pfitzmann. Sorting out signature schemes. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 74–85, 1993.

[24] B. Pfitzmann. *Digital Signature Schemes – General Framework and Fail-Stop Signatures*, volume 1100 of *Lecture Notes in Computer Science*. Springer, 1996.

[25] 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.

[26] 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.

[27] 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.

[28] 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.