A Composable Cryptographic Library with Nested Operations
(Extended Abstract)

Michael Backes
IBM Research Division
Rüschlikon, Switzerland
mbc@zurich.ibm.com

Birgit Pfitzmann
IBM Research Division
Rüschlikon, Switzerland
bpf@zurich.ibm.com

Michael Waidner
IBM Research Division
Rüschlikon, Switzerland
wmi@zurich.ibm.com

ABSTRACT

We present the first idealized cryptographic library that can be used like the Dolev-Yao model for automated proofs of cryptographic protocols that use nested cryptographic operations, while coming with a cryptographic implementation that is provably secure under active attacks.

Categories and Subject Descriptors

E.3 [**Data**]: Data Encryption; F.1.2 [**Theory of Computation**]: Computation by Abstract Devices, Modes of Computation

General Terms

Security, Theory, Verification

Keywords

Cryptography, Simulatability, Security Analysis of Protocols, Cryptographically Composable Operators

1. INTRODUCTION

Many practically relevant cryptographic protocols like SSL/TLS, S/MIME, IPSec, or SET use cryptographic primitives like signature schemes or encryption in a black-box way, while adding many non-cryptographic features. Vulnerabilities have accompanied the design of such protocols ever since early authentication protocols like Needham-Schroeder [43, 25], over carefully designed de-facto standards like SSL and PKCS [52, 16], up to current widely deployed products like Microsoft Passport [29]. However, proving the security of such protocols has been a very unsatisfactory task for a long time.

One possibility was to take the cryptographic approach. This means reduction proofs between the security of the overall system and the security of the cryptographic primitives, i.e., one shows that if one could break the overall system, one could also break one of the underlying cryptographic primitives with respect to their cryptographic definitions, e.g., adaptive chosen-message security for signature schemes. For authentication protocols, this approach was first used in [15]. In principle, proofs in this approach are as rigorous as typical proofs in mathematics. In practice, however, human beings are extremely fallible with this type of proofs. This is not due to the cryptography, but to the distributed-systems aspects of the protocols. It is well-known from non-cryptographic distributed systems that many wrong protocols have been published even for very small problems. Hand-made proofs are highly error-prone because following all the different cases how actions of different machines interleave is extremely tedious. Humans tend to take wrong shortcuts and do not want to proof-read such details in proofs by others. If the protocol contains cryptography, this obstacle is even much worse: Already a rigorous definition of the goals gets more complicated, and often not only trace properties (integrity) have to be proven but also secrecy. Further, in principle the complexity-theoretic reduction has to be carried out across all these cases, and it is not at all trivial to do this rigorously. In consequence, there is almost no real cryptographic proof of a larger protocol, and several times supposedly proven, relatively small systems were later broken, e.g., [47, 26].

The other possibility was to use formal methods. There one leaves the tedious parts of proofs to machines, i.e., model checkers or automatic theorem provers. This means to code the cryptographic protocols into the language of such tools, which may need more or less start-up work depending on whether the tool already supports distributed systems or whether interaction models have to be encoded first. None of these tools, however, is currently able to deal with reduction proofs. Nobody even thought about this for a long time, because one felt that protocol proofs could be based on simpler, idealized abstractions from cryptographic primitives. Almost all these abstractions are variants of the Dolev-Yao model [27], which represents all cryptographic primitives as operators of a term algebra with cancellation rules. For instance, public-key encryption is represented by operators E for encryption and D for decryption with one cancellation rule, $\mathsf{D}(\mathsf{E}(m)) = m$ for all m. Encrypting a message m twice in this model does not yield another message from the basic message space but the term $\mathsf{E}(\mathsf{E}(m))$. Further, the model assumes that two terms whose equality cannot be derived with the cancellation rules are not equal, and every term that cannot be derived is completely secret. However, originally there was no foundation at all for such assumptions about real cryptographic primitives, and thus no guarantee that protocols proved with these tools were still secure when implemented with real cryptography. Although no previously proved protocol has been broken when implemented with standard provably secure cryptosystems, this was clearly an unsatisfactory situation, and artificial counterexamples can be constructed.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'03, October 27–30, 2003, Washington, DC, USA.
Copyright 2003 ACM 1-58113-738-9/03/0010 ...$5.00.

Three years ago, efforts started to get the best of both worlds. Essentially, [46, 48] started to define general cryptographic models that support idealization that is secure in arbitrary environments and under arbitrary active attacks, while [2] started to justify the Dolev-Yao model as far as one could without such a model. Both directions were significantly extended in subsequent papers, in particular [1, 49, 19, 5].

Nevertheless, this paper is the first that offers a provably secure variant of the Dolev-Yao model for proofs that people typically make with the Dolev-Yao model, because for the first time we cover both active attacks and nested cryptographic operations. This new property combination is essential: First, most cryptographic protocols are broken by active attacks, e.g., man-in-the-middle attacks or attacks where an adversary reuses a message from one protocol step in a different protocol step where it suddenly gets a different semantics. Such attacks are not covered by [2, 1]. Secondly, the main use of the Dolev-Yao model is to represent nested protocol messages like $\mathsf{E}_{pke_v}(\mathsf{sign}_{sks_u}(m, N_1), N_2)$, where m denotes an arbitrary message and N_1, N_2 two nonces. No previous idealization proved in the reactive cryptographic models contains abstractions from cryptographic primitives (here mainly encryption and signatures, but also the nonces and the list operation) that can be used in such nested terms. Existing abstractions are either too high-level, e.g., the secure channels in [49, 5] combine encryption and signatures in a fixed way. Or they need immediate interaction with the adversary [19, 18], i.e., the adversary learns the structure of every term any honest party ever builds, and even every signed message. This abstraction is not usable for a term as above because one may want to show that m is secret because of the outer encryption, but the abstraction gives m to the adversary. (A similar immediate application of the model of [49] to such primitives would avoid this problem, but instead keep all signatures and ciphertexts in the system, so that nesting is also not possible.) Finally, there exist some semi-abstractions which still depend on cryptographic details [39, 49]. Thus they are not suitable for abstract protocol representations and proof tools, but we use such a semi-abstraction of public-key encryption as a submodule below.

The first decision in the design of an ideal library that supports both nesting and general active attacks was how we can represent an idealized cryptographic term and the corresponding real message in the *same* way to a higher protocol. This is necessary for using the reactive cryptographic models and their composition theorems. We do this by handles. In the ideal system, these handles essentially point to Dolev-Yao-like terms, while in the real system they point to real cryptographic messages. Our model for storing the terms belonging to the handles is stateful and in the ideal system comprises the knowledge of who knows which terms. Thus our overall ideal cryptographic library corresponds more to "the CSP Dolev-Yao model" or "the Strand-space Dolev-Yao model" than the pure algebraic Dolev-Yao model. Once one has the idea of handles, one has to consider whether one can put the exact Dolev-Yao terms under them or how one has to or wants to deviate from them in order to allow a provably secure cryptographic realization, based on a more or less general class of underlying primitives. An overview of these deviations is given in Section 1.2, and Section 1.3 surveys how the cryptographic primitives are augmented to give a secure implementation of the ideal library.

The vast majority of the work was to make a credible proof that the real cryptographic library securely implements the ideal one. This is a hand-made proof based on cryptographic primitives and with many distributed-systems aspects, and thus with all the problems mentioned above for cryptographic proofs of large protocols. Indeed we needed a novel proof technique consisting of a probabilistic, imperfect bisimulation with an embedded static information-flow analysis, followed by cryptographic reductions proofs for so-called error sets of traces where the bisimulation did not work. As this proof needs to be made only once, and is intended to be the justification for later basing many protocol proofs on the ideal cryptographic library and proving them with higher assurance using automatic tools, we carefully worked out all the tedious details, and we encourage some readers to double-check the 68-page full version of this paper [10]. Based on our experience with making this proof and the errors we found by making it, we strongly discourage the reader against accepting idealizations of cryptographic primitives where a similar security property, simulatability, is claimed but only the first step of the proof, the definition of a simulator, is made.

1.1 Further Related Literature

Both the cryptographic and the idealizing approach at proving cryptographic systems started in the early 80s. Early examples of cryptographic definitions and reduction proofs are [33, 34]. Applied to protocols, these techniques are at their best for relatively small protocols where there is still a certain interaction between cryptographic primitives, e.g., [14, 51]. The early methods of automating proofs based on the Dolev-Yao model are summarized in [37]. More recently, such work concentrated on using existing general-purpose model checkers [40, 42, 24] and theorem provers [28, 45], and on treating larger protocols, e.g., [12].

Work intended to bridge the gap between the cryptographic approach and the use of automated tools started independently with [46, 48] and [2]. In [2], Dolev-Yao terms, i.e., with nested operations, are considered specifically for symmetric encryption. However, the adversary is restricted to passive eavesdropping. Consequently, it was not necessary to define a reactive model of a system, its honest users, and an adversary, and the security goals were all formulated as indistinguishability of terms. This was extended in [1] from terms to more general programs, but the restriction to passive adversaries remains, which is not realistic in most practical applications. Further, there are no theorems about composition or property preservation from the abstract to the real system. Several papers extended this work for specific models or specific properties. For instance, [35] specifically considers strand spaces and information-theoretically secure authentication only. In [38] a deduction system for information flow is based on the same operations as in [2], still under passive attacks only.

The approach in [46, 48] was from the other end: It starts with a general reactive system model, a general definition of cryptographically secure implementation by simulatability, and a composition theorem for this notion of secure implementation. This work is based on definitions of secure *function* evaluation, i.e., the computation of one set of outputs from one set of inputs [32, 41, 11, 17]; earlier extensions towards reactive systems were either without real abstraction [39] or for quite special cases [36]. The approach was extended from synchronous to asynchronous systems in [49, 19]. All the reactive works come with more or less worked-out examples of abstractions of cryptographic systems, and first tool-supported proofs were made based on such an abstraction [5, 4] using the theorem prover PVS [44]. However, even with a composition theorem this does not automatically give a cryptographic library in the Dolev-Yao sense, i.e., with the possibility to nest abstract operations, as explained above. Our cryptographic library overcomes these problems. It supports nested operations in the intuitive sense; operations that are performed locally are not visible to the adversary. It is secure against arbitrary active attacks, and works in the context of arbitrary surrounding interactive protocols.

This holds independently of the goals that one wants to prove about the surrounding protocols; in particular, property preservation theorems for the simulatability definition we use have been proved for integrity, liveness, and non-interference [4, 9, 6, 8].

We have already exemplified the usefulness of the cryptographic library by conducting the first cryptographically sound security proof of the well-known Needham-Schroeder-Lowe protocol [7]. Since the proof relies on idealizations of cryptography it has all the advantages explained in the text; in particular, the proof is suited for formal proof tools. Simultaneously and independently to this work, another cryptographically sound security proof of this protocol was invented in [53]. This proof is done from scratch in the cryptographic setting and is hence vulnerable to the problems mentioned before. However, it is fair to mention that this proof establishes the security property of matching conversations whereas our proof currently only strives for a weaker authentication requirement.

1.2 Overview of the Ideal Cryptographic Library

The ideal cryptographic library offers its users abstract cryptographic operations, such as commands to encrypt or decrypt a message, to make or test a signature, and to generate a nonce. All these commands have a simple, deterministic semantics. In a reactive scenario, this semantics is based on state, e.g., of who already knows which terms. We store state in a "database". Each entry has a type, e.g., "signature", and pointers to its arguments, e.g., a key and a message. This corresponds to the top level of a Dolev-Yao term; an entire term can be found by following the pointers. Further, each entry contains handles for those participants who already know it. Thus the database index and these handles serve as an infinite, but efficiently constructible supply of global and local names for cryptographic objects. However, most libraries have export operations and leave message transport to their users ("token-based"). An actual implementation of the simulatable library might internally also be structured like this, but higher protocols are only automatically secure if they do not use this export function except via the special send operations.

The ideal cryptographic library does not allow cheating. For instance, if it receives a command to encrypt a message m with a certain key, it simply makes an abstract entry in a database for the ciphertext. Each entry further contains handles for those participants who already know it. Another user can only ask for decryption of this ciphertext if he has handles to both the ciphertext and the secret key. Similarly, if a user issues a command to sign a message, the ideal system looks up whether this user should have the secret key. If yes, it stores that this message has been signed with this key. Later tests are simply look-ups in this database. A send operation makes an entry known to other participants, i.e., it adds handles to the entry. Recall that our ideal library is an entire reactive system and therefore contains an abstract network model. We offer three types of send commands, corresponding to three channel types $\{\mathsf{s}, \mathsf{a}, \mathsf{i}\}$, meaning secure, authentic (but not private), and insecure. The types could be extended. Currently, our library contains public-key encryption and signatures, nonces, lists, and application data. We have recently added symmetric authentication (still unpublished).

The main differences between our ideal cryptographic library and the standard Dolev-Yao model are the following. Some of them already exist in prior extensions of the Dolev-Yao model.

- Signature schemes are not "inverses" of encryption schemes.

- Secure encryption schemes are necessarily probabilistic, and so are most secure signature schemes. Hence if the same message is signed or encrypted several times, we distinguish the versions by making different database entries.

- Secure signature schemes often have memory. The standard definition [34] does not even exclude that one signature divulges the entire history of messages signed before. We have to restrict this definition, but we allow a signature to divulge the number of previously signed messages, so that we include the most efficient provably secure schemes under classical assumptions like the hardness of factoring [34, 20, 21].[1]

- We cannot (easily) allow participants to send secret keys over the network because then the simulation is not always possible.[2] Fortunately, for public-key cryptosystems this is not needed in typical protocols.

- Encryption schemes cannot keep the length of arbitrary cleartexts entirely secret. Typically one can even see the length quite precisely because message expansion is minimized. Hence we also allow this in the ideal system. A fixed-length version would be an easy addition to the library, or can be implemented on top of the library by padding to a fixed length.

- Adversaries may include incorrect messages in encrypted parts of a message which the current recipient cannot decrypt, but may possibly forward to another recipient who can, and will thus notice the incorrect format. Hence we also allow certain "garbage" terms in the ideal system.

1.3 Overview of the Real Cryptographic Library

The real cryptographic library offers its users the same commands as the ideal one, i.e., honest users operate on cryptographic objects via handles. This is quite close to standard APIs for existing implementations of cryptographic libraries that include key storage. The database of the real system contains real cryptographic keys, ciphertexts, etc., and the commands are implemented by real cryptographic algorithms. Sending a term on an insecure channel releases the actual bitstring to the adversary, who can do with it what he likes. The adversary can also insert arbitrary bitstrings on non-authentic channels. The simulatability proof will show that nevertheless, everything a real adversary can achieve can also be achieved by an adversary in the ideal system, or otherwise the underlying cryptography can be broken.

We base the implementation of the commands on arbitrary secure encryption and signature systems according to standard cryptographic definitions. However, we "idealize" the cryptographic objects and operations by measures similar to robust protocol design [3].

- All objects are tagged with a type field so that, e.g., signatures cannot also be acceptable ciphertexts or keys.

- Several objects are also tagged with their parameters, e.g., signatures with the public key used.

- Randomized operations are randomized completely. For instance, as the ideal system represents several signatures under the same message with the same key as different, the real

[1]Memory-less schemes exist with either lower efficiency or based on stronger assumptions (e.g., [31, 23, 30]). We could add them to the library as an additional primitive.

[2]The primitives become "committing". This is well-known from individual simulation proofs. It also explains why [2] is restricted to passive attacks.

system has to guarantee that they *will* be different, except for small error probabilities. Even probabilistic encryptions are randomized additionally because they are not always sufficiently random for keys chosen by the adversary.

The reason to tag signatures with the public key needed to verify them is that the usual definition of a secure signature scheme does not exclude "signature stealing:" Let (sks_h, pks_h) denote the key pair of a correct participant. With ordinary signatures an adversary might be able to compute a valid key pair (sks_a, pks_a) such that signatures that pass the test with pks_h also pass the test with pks_a. Thus, if a correct participant receives an encrypted signature on m, it might accept m as being signed by the adversary, although the adversary never saw m. It is easy to see that this would result in protocols that could not be simulated. Our modification prevents this anomaly.

For the additional randomization of signatures, we include a random string r in the message to be signed. Alternatively we could replace r by a counter, and if a signature scheme is strongly randomized already we could omit r. Ciphertexts are randomized by including the same random string r in the message to be encrypted and in the ciphertext. The outer r prevents collisions among ciphertexts from honest participants, the inner r ensures continued non-malleability.

2. PRELIMINARY DEFINITIONS

We briefly sketch the definitions from [49]. A *system* consists of several possible *structures*. A structure consists of a set M of connected correct machines and a subset S of free ports, called *specified ports*. 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 measured in terms of a common security parameter k, given as the initial work-tape content of every machine. Readers only interested in using the ideal cryptographic library in larger protocols only need normal, deterministic IO automata.

In a *standard real cryptographic system*, the structures are derived from one intended structure and a trust model consisting of an access structure $\mathcal{ACC}$ and a channel model χ. Here $\mathcal{ACC}$ contains the possible sets $\mathcal{H}$ of indices of uncorrupted machines among the intended ones, and χ designates whether each channel is secure, authentic (but not private) or insecure. In a typical ideal system, each structure contains only one machine TH called *trusted host*.

Each structure is complemented to a *configuration* by an arbitrary *user* machine H and *adversary* machine A. 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 [49] 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 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.

2.1 Simulatability

Simulatability is the cryptographic notion of secure implementation. For reactive systems, it means that whatever might happen to an honest user in a real system Sys_{real} can also happen in the given ideal system Sys_{id}: For every structure $(M_1, S) \in Sys_{\mathsf{real}}$, 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 $(M_2, S) \in Sys_{\mathsf{id}}$ such that the view of H is computationally indistinguishable in the two configurations. This is illustrated in Figure 1. Indistinguishability is a well-known cryptographic notion from [54].

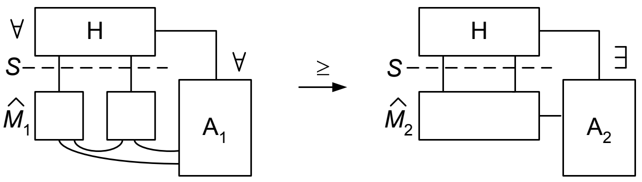

Figure 1: Simulatability: The two views of H must be indistinguishable.

Definition 1. (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* ("$\approx$") 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*, i.e., $g\colon \mathbb{N} \to \mathbb{R}_{\geq 0} \in NEGL$ iff for all positive polynomials Q, $\exists k_0 \forall k \geq k_0\colon\ g(k) \leq 1/Q(k)$.

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 2. (Simulatability) Let systems Sys_{real} and Sys_{id} be given. We say $Sys_{\mathsf{real}} \geq Sys_{\mathsf{id}}$ (*at least as secure as*) iff for every polynomial-time configuration $conf_1 = S\mathsf{HA}_1 \in \mathsf{Conf}(Sys_{\mathsf{real}})$, there exists a polynomial-time configuration $conf_2 = S\mathsf{HA}_2 \in \mathsf{Conf}(Sys_{\mathsf{id}})$ (with the same H) such that $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$.

For the cryptographic library, we even show blackbox simulatability, i.e., A_2 consists of a simulator Sim that depends only on (M_1, S) and uses A_1 as a blackbox submachine. An essential feature of this definition of simulatability is a composition theorem [49], which essentially says that one can design and prove a larger system based on the ideal system Sys_{id}, and then securely replace Sys_{id} by the real system Sys_{real}.

2.2 Notation

We write "$:=$" for deterministic and "$\leftarrow$" for probabilistic assignment, and "$\leftarrow_R$" for uniform random choice from a set. By $x := y$++ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $|m|$, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The list operation is denoted as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$. A database D is a set of functions, called entries, each over a finite domain called attributes. For an entry $x \in D$, the value at an attribute att is written $x.att$. For a predicate $pred$ involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill $pred$. If $D[pred]$ contains only one element, we use the same notation for this element. Adding an entry x to D is abbreviated $D :\Leftarrow x$.

3. IDEAL CRYPTOGRAPHIC LIBRARY

The ideal cryptographic library consists of a trusted host $\mathsf{TH}(\mathcal{H})$ for every subset $\mathcal{H}$ of a set $\{1, \ldots, n\}$ of users. It has a port $\mathsf{in}_u?$ for inputs from and a port $\mathsf{out}_u!$ for outputs to each user $u \in \mathcal{H}$ and for $u = \mathsf{a}$, denoting the adversary.

As mentioned in Section 1.2, we do not assume encryption systems to hide the length of the message. Furthermore, higher protocols may need to know the length of certain terms even for honest participants. Thus the trusted host is parameterized with certain length functions denoting the length of a corresponding value in the real system. The tuple of these functions is contained in a system parameter L.

For simulatability by a polynomial-time real system, the ideal cryptographic library has to be polynomial-time. It therefore contains explicit bounds on the message lengths, the number of signatures per key, and the number of accepted inputs at each port. They are also contained in the system parameter L. The underlying IO automata model guarantees that a machine can enforce such bounds without additional Turing steps even if an adversary tries to send more data. For all details, we refer to the full version [10].

3.1 States

The main data structure of $\mathsf{TH}(\mathcal{H})$ is a database D. The entries of D are abstract representations of the data produced during a system run, together with the information on who knows these data. Each entry $x \in D$ is of the form

$$(\mathit{ind}, \mathit{type}, \mathit{arg}, \mathit{hnd}_{u_1}, \ldots, \mathit{hnd}_{u_m}, \mathit{hnd}_{\mathsf{a}}, \mathit{len})$$

where $\mathcal{H} = \{u_1, \ldots, u_m\}$ and:

- $\mathit{ind} \in \mathbb{N}_0$ is called the *index* of x. We write $D[i]$ instead of $D[\mathit{ind} = i]$.
- $\mathit{type} \in \mathit{typeset} := \{\mathsf{data}, \mathsf{list}, \mathsf{nonce}, \mathsf{ske}, \mathsf{pke}, \mathsf{enc}, \mathsf{sks}, \mathsf{pks}, \mathsf{sig}, \mathsf{garbage}\}$ identifies the *type* of x. Future extensions of the library can extend this set.
- $\mathit{arg} = (a_1, a_2, \ldots, a_j)$ is a possibly empty *list of arguments*.
- $\mathit{hnd}_u \in \mathbb{N}_0 \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ identifies how u knows this entry. The value a represents the adversary, and $\mathit{hnd}_u = \downarrow$ indicates that u does not know this entry. A value $\mathit{hnd}_u \neq \downarrow$ is called the *handle* for u to entry x. We always use a superscript "hnd" for handles and usually denote a handle to an entry $D[i]$ by i^{hnd}.
- $\mathit{len} \in \mathbb{N}_0$ denotes the *length* of the abstract entry. It is computed by $\mathsf{TH}(\mathcal{H})$ using the given length functions from the system parameter L.

Initially, D is empty. $\mathsf{TH}(\mathcal{H})$ keeps a variable size denoting the current number of elements in D. New entries x always receive the index $\mathit{ind} := \mathit{size}$++, and $x.\mathit{ind}$ is never changed. For each $u \in \mathcal{H} \cup \{\mathsf{a}\}$, $\mathsf{TH}(\mathcal{H})$ maintains a counter curhnd_u (current handle) over $\mathbb{N}_0$ initialized with 0, and each new handle for u will be chosen as $i^{\mathsf{hnd}} := \mathit{curhnd}_u$++.

3.2 Inputs and their Evaluation

Each input c at a port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$ should be a list $(\mathit{cmd}, x_1, \ldots, x_j)$. We usually write it $y \leftarrow \mathit{cmd}(x_1, \ldots, x_j)$ with a variable y designating the result that $\mathsf{TH}(\mathcal{H})$ returns at $\mathsf{out}_u!$. The value cmd should be a command string, contained in one of the following four *command sets*. Commands in the first two sets are available for both the user and the adversary, while the last two sets model special adversary capabilities and are only accepted for $u = \mathsf{a}$. The command sets can be enlarged by future extensions of the library.

3.2.1 Basic Commands

First, we have a set $\mathit{basic\_cmds}$ of *basic commands*. Each basic command represents one cryptographic operation; arbitrary terms similar to the Dolev-Yao model are built up or decomposed by a sequence of commands. For instance there is a command gen_nonce to create a nonce, encrypt to encrypt a message, and list to combine several messages into a list. Moreover, there are commands store and retrieve to store real-world messages (bitstrings) in the library and to retrieve them by a handle. Thus other commands can assume that everything is addressed by handles. We only allow lists to be signed and transferred, because the list-operation is a convenient place to concentrate all verifications that no secret items are put into messages. Altogether, we have

$$\begin{gathered}\mathit{basic\_cmds} := \{\mathsf{get\_type}, \mathsf{get\_len}, \mathsf{store}, \mathsf{retrieve}, \mathsf{list}, \mathsf{list\_proj},\\ \mathsf{gen\_nonce}, \mathsf{gen\_sig\_keypair}, \mathsf{sign}, \mathsf{verify}, \mathsf{pk\_of\_sig}, \mathsf{msg\_of\_sig},\\ \mathsf{gen\_enc\_keypair}, \mathsf{encrypt}, \mathsf{decrypt}, \mathsf{pk\_of\_enc}\}.\end{gathered}$$

The commands not yet mentioned have the following meaning: get_type and get_len retrieve the type and abstract length of a message; list_proj retrieves a handle to the i-th element of a list; gen_sig_keypair and gen_enc_keypair generate key pairs for signatures and encryption, respectively, initially with handles for only the user u who input the command; sign, verify, and decrypt have the obvious purpose, and pk_of_sig, msg_of_sig; and pk_of_enc retrieve a public key or message, respectively, from a signature or ciphertext. (Retrieving public keys will be possible in the real cryptographic library because we tag signatures and ciphertexts with public keys as explained above.)

We only present the details of how $\mathsf{TH}(\mathcal{H})$ evaluates such basic commands based on its abstract state for two examples, nonce generation and encryption; see the full version [10] for the complete definition. We assume that the command is entered at a port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$. Basic commands are *local*, i.e., they produce a result at port $\mathsf{out}_u!$ and possibly update the database D, but do not produce outputs at other ports. They also do not touch handles for participants $v \neq u$. The functions $\mathsf{nonce\_len}^*$, $\mathsf{enc\_len}^*$, and $\mathsf{max\_len}$ are length functions and the message-length bound from the system parameter L.

For nonces, $\mathsf{TH}(\mathcal{H})$ just creates a new entry with type nonce, no arguments, a handle for user u, and the abstract nonce length. This models that in the real system nonces are randomly chosen bitstrings of a certain length, which should be all different and not guessable by anyone else than u initially. It outputs the handle to u.

- *Nonce Generation:* $n^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.

  Set $n^{\mathsf{hnd}} := \mathit{curhnd}_u$++ and

  $$\begin{aligned} D \;:\Leftarrow\; & (\mathit{ind} := \mathit{size}\text{++}, \mathit{type} := \mathsf{nonce}, \mathit{arg} := (),\\ & \mathit{hnd}_u := n^{\mathsf{hnd}}, \mathit{len} := \mathsf{nonce\_len}^*(k)).\end{aligned}$$

The inputs for public-key encryption are handles to the public key and the plaintext list. $\mathsf{TH}(\mathcal{H})$ verifies the types (recall the notation $D[\mathit{pred}]$) and verifies that the ciphertext will not exceed the maximum length. If everything is ok, it makes a new entry of type enc, with the indices of the public key and the plaintext as arguments, a handle for user u, and the computed length. The fact that each such entry is new models probabilistic encryption, and the arguments model the highest layer of the corresponding Dolev-Yao term.

- *Public-Key Encryption:* $c^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pk^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.

  Let $pk := D[hnd_u = pk^{\mathsf{hnd}} \wedge type = \mathsf{pke}].ind$ and $l := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$ and $length := \mathsf{enc\_len}^*(k, D[l].len)$. If $length > \mathsf{max\_len}(k)$ or $pk = \downarrow$ or $l = \downarrow$, then return $\downarrow$. Else set $c^{\mathsf{hnd}} := curhnd_u$++ and

  $$D \; :\Leftarrow \; (ind := size\text{++}, type := \mathsf{enc}, arg := (pk, l), \\ hnd_u := c^{\mathsf{hnd}}, len := length).$$

3.2.2 Honest Send Commands

Secondly, we have a set $send\_cmds := \{\mathsf{send\_s}, \mathsf{send\_a}, \mathsf{send\_i}\}$ of *honest send commands* for sending messages on channels of different degrees of security. As an example we present the details of the most important case, insecure channels.

- $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$, for $v \in \{1, \ldots, n\}$.

  Let $l^{\mathsf{ind}} := D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$, output $(u, v, \mathsf{i}, \mathsf{ind2hnd_a}(l^{\mathsf{ind}}))$ at $\mathsf{out_a}!$.

The used algorithm $\mathsf{ind2hnd}_u$ retrieves the handle for user u to the entry with the given index if there is one, otherwise it assigns a new such handle as $curhnd_u$++. Thus this command means that the database D now stores that this message is known to the adversary, and that the adversary learns by the output that user u wanted to send this message to user v.

Most protocols should only use the other two send commands, i.e., secret or authentic channels, for key distribution at the beginning. As the channel type is part of the send-command name, syntactic checks can ensure that a protocol designed with the ideal cryptographic library fulfills such requirements.

3.2.3 Local Adversary Commands

Thirdly, we have a set $adv\_local\_cmds := \{\mathsf{adv\_garbage}, \mathsf{adv\_invalid\_ciph}, \mathsf{adv\_transform\_sig}, \mathsf{adv\_parse}\}$ of *local adversary commands*. They model tolerable imperfections of the real system, i.e., actions that may be possible in real systems but that are not required. First, an adversary may create *invalid entries* of a certain length; they obtain the type garbage. Secondly, *invalid ciphertexts* are a special case because participants not knowing the secret key can reasonably ask for their type and query their public key, hence they cannot be of type garbage. Thirdly, the security definition of signature schemes does not exclude that the adversary *transforms signatures* by honest participants into other valid signatures on the same message with the same public key. Finally, we allow the adversary to retrieve all information that we do not explicitly require to be hidden, which is denoted by a command adv_parse. This command returns the type and usually all the abstract arguments of a value (with indices replaced by handles), e.g., parsing a signature yields the public key for testing this signature, the signed message, and the value of the signature counter used for this message. Only for ciphertexts where the adversary does not know the secret key, parsing only returns the length of the cleartext instead of the cleartext itself.

3.2.4 Adversary Send Commands

Fourthly, we have a set $adv\_send\_cmds := \{\mathsf{adv\_send\_s}, \mathsf{adv\_send\_a}, \mathsf{adv\_send\_i}\}$ of *adversary send commands*, again modeling different degrees of security of the channels used. In contrast to honest send commands, the sender of a message is an additional input parameter. Thus for insecure channels the adversary can pretend that a message is sent by an arbitrary honest user.

3.3 A Small Example

Assume that a cryptographic protocol has to perform the step

$$u \rightarrow v \colon \mathsf{enc}_{pke_v}(\mathsf{sign}_{sks_u}(m, N_1), N_2),$$

where m is an input message and N_1, N_2 are two fresh nonces. Given our library, this is represented by the following sequence of commands input at port $\mathsf{in}_u?$. We assume that u has already received a handle pke_v^{hnd} to the public encryption key of v, and created signature keys, which gave him a handle sks_u^{hnd}.

1. $m^{\mathsf{hnd}} \leftarrow \mathsf{store}(m)$.
2. $N_1^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
3. $l_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(m^{\mathsf{hnd}}, N_1^{\mathsf{hnd}})$.
4. $sig^{\mathsf{hnd}} \leftarrow \mathsf{sign}(sks_u^{\mathsf{hnd}}, l_1^{\mathsf{hnd}})$.
5. $N_2^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
6. $l_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(sig^{\mathsf{hnd}}, N_2^{\mathsf{hnd}})$.
7. $enc^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_v^{\mathsf{hnd}}, l_2^{\mathsf{hnd}})$.
8. $m^{\mathsf{hnd}} \leftarrow \mathsf{list}(enc^{\mathsf{hnd}})$.
9. $\mathsf{send\_i}(v, m^{\mathsf{hnd}})$.

Note that the entire term is constructed by a local interaction of user u and the ideal library, i.e., the adversary does not learn anything about this interaction until Step 8. In Step 9, the adversary gets an output $(u, v, \mathsf{i}, m_{\mathsf{a}}^{\mathsf{hnd}})$ with a handle $m_{\mathsf{a}}^{\mathsf{hnd}}$ for him to the resulting entry. In the real system described below, the sequence of inputs for constructing and sending this term is identical, but real cryptographic operations are used to build up a bitstring m until Step 8, and m is sent to v via a real insecure channel in Step 9.

4. REAL CRYPTOGRAPHIC LIBRARY

The real system is parameterized by a digital signature scheme $\mathcal{S}$ and a public-key encryption scheme $\mathcal{E}$. The ranges of all functions are $\{0,1\}^+ \cup \{\downarrow\}$. The signature scheme has to be secure against existential forgery under adaptive chosen-message attacks [34]. This is the accepted security definition for general-purpose signing. The encryption scheme has to fulfill that two equal-length messages are indistinguishable even in adaptive chosen-ciphertext attacks. Chosen-ciphertext security has been introduced in [50] and formalized as "IND-CCA2" in [13]. It is the accepted definition for general-purpose encryption. An efficient encryption system secure in this sense is [22]. Just like the ideal system, the real system is parameterized by a tuple L' of length functions and bounds.

4.1 Structures

The intended structure of the real cryptographic library consists of n machines $\{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$. Each M_u has ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$, so that the same honest users can connect to the ideal and the real library. Each M_u has three connections $\mathsf{net}_{u,v,x}$ to each M_v for $x \in \{\mathsf{s}, \mathsf{a}, \mathsf{i}\}$. They are called network connections and the corresponding ports network ports. Network connections are scheduled by the adversary.

The actual system is a standard cryptographic system as defined in [49] and sketched in Section 2. Any subset of the machines may be corrupted, i.e., any set $\mathcal{H} \subseteq \{1, \ldots, n\}$ can denote the indices of correct machines. The channel model means that in an actual structure, an honest intended recipient gets all messages output at network ports of type s(secret) and a (authentic) and the adversary gets all messages output at ports of type a and i (insecure). Furthermore, the adversary makes all inputs to a network port of type i. This is shown in Figure 2.

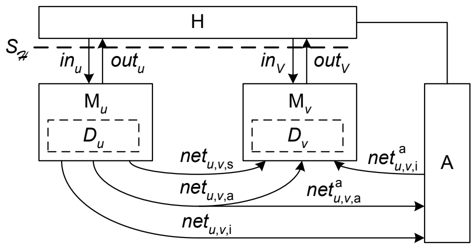

Figure 2: Connections from a correct machine to another in the real system.

4.2 States of a Machine

The main data structure of M_u is a database D_u that contains implementation-specific data such as ciphertexts and signatures produced during a system run, together with the handles for u and the type as in the ideal system, and possibly additional internal attributes. Thus each entry $x \in D_u$ is of the form

$$(hnd_u, word, type, add\_arg).$$

- $hnd_u \in \mathbb{N}_0$ is the *handle* of x and consecutively numbers all entries in D_u.
- $word \in \{0,1\}^+$, called *word*, is the real cryptographic representation of x.
- $type \in typeset \cup \{\mathsf{null}\}$ is the *type* of x, where null denotes that the entry has not yet been parsed.
- add_arg is a list of *additional arguments*. Typically it is $()$, only for signing keys it contains the signature counter.

Similar to the ideal system, M_u maintains a counter $curhnd_u$ over $\mathbb{N}_0$ denoting the current number of elements in D_u. New entries x always receive $hnd_u := curhnd_u$++, and $x.hnd_u$ is never changed.

4.3 Inputs and their Evaluation

Now we describe how M_u evaluates individual inputs. Inputs at port $\mathsf{in}_u?$ should be basic commands and honest send commands as in the ideal system, while network inputs can be arbitrary bitstrings. Often a bitstrings has to be parsed. This is captured by a functional algorithm parse, which outputs a pair $(type, arg)$ of a type $\in typeset$ and a list of real arguments, i.e., of bitstrings. This corresponds to the top level of a term, similar to the abstract arguments in the ideal database D. By "parse m^{hnd}" we abbreviate that M_u calls $(type, arg) \leftarrow \mathsf{parse}(D_u[m^{\mathsf{hnd}}].word)$, assigns $D_u[m^{\mathsf{hnd}}].type := type$ if it was still null, and may then use arg.

4.3.1 Basic Commands

Basic commands are again *local*, i.e., they do not produce outputs at network ports. The basic commands are implemented by the underlying cryptographic operations with the modifications motivated in Section 1.3. For general unambiguousness, not only all cryptographic objects are tagged, but also data and lists. Similar to the ideal system, we only show two examples of the evaluation of basic commands, and additionally how ciphertexts are parsed. All other commands can be found in the full version [10].

In nonce generation, a real nonce n is generated by tagging a random bitstring n' of a given length with its type nonce. Further, a new handle for u is assigned and the handle, the word n, and the type are stored without additional arguments.

- *Nonce Generation:* $n^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.

  Let $n' \leftarrow_R \{0,1\}^{\mathsf{nonce\_len}(k)}$, $n := (\mathsf{nonce}, n')$, $n^{\mathsf{hnd}} := curhnd_u$++ and $D_u :\Leftarrow (n^{\mathsf{hnd}}, n, \mathsf{nonce}, ())$.

For the encryption command, let $\mathsf{E}_{pk}(m)$ denote probabilistic encryption of a string m with the public key pk in the underlying encryption system $\mathcal{E}$. The parameters are first parsed in case they have been received over the network, and their types are verified. Then the second component of the (tagged) public-key word is the actual public key pk, while the message l is used as it is. Further, a fresh random value r is generated for additional randomization as explained in Section 1.3.

Recall that r has to be included both inside the encryption and in the final tagged ciphertext c^*.

- *Encryption:* $c^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pk^{\mathsf{hnd}}, l^{\mathsf{hnd}})$.

  Parse pk^{hnd} and l^{hnd}. If $D_u[pk^{\mathsf{hnd}}].type \neq \mathsf{pke}$ or $D_u[l^{\mathsf{hnd}}].type \neq \mathsf{list}$, return $\downarrow$. Else set $pk := D_u[pk^{\mathsf{hnd}}].word[2]$, $l := D_u[l^{\mathsf{hnd}}].word$, $r \leftarrow_R \{0,1\}^{\mathsf{nonce\_len}(k)}$, encrypt $c \leftarrow \mathsf{E}_{pk}((r,l))$, and set $c^* := (\mathsf{enc}, pk, c, r)$. If $c^* = \downarrow$ or $|c^*| > \mathsf{max\_len}(k)$ then return $\downarrow$, else set $c^{\mathsf{hnd}} := curhnd_u$++ and $D_u :\Leftarrow (c^{\mathsf{hnd}}, c^*, \mathsf{enc}, ())$.

Parsing a ciphertext verifies that the components and lengths are as in c^* above, and outputs the corresponding tagged public key, whereas the message is only retrieved by a decryption command.

4.3.2 Send Commands and Network Inputs

Send commands simply output real messages at the appropriate network ports. We show this for an insecure channel.

- $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$ for $v \in \{1,\ldots,n\}$.

  Parse l^{hnd} if necessary. If $D_u[l^{\mathsf{hnd}}].type = \mathsf{list}$, output $D_u[l^{\mathsf{hnd}}].word$ at port $\mathsf{net}_{u,v,\mathsf{i}}!$.

Upon receiving a bitstring l at a network port $\mathsf{net}_{w,u,x}?$, machine M_u parses it and verifies that it is a list. If yes, and if l is new, M_u stores it in D_u using a new handle l^{hnd}, else it retrieves the existing handle l^{hnd}. Then it outputs (w, x, l^{hnd}) at port $\mathsf{out}_u!$.

5. SECURITY PROOF

The security claim is that the real cryptographic library is as secure as the ideal cryptographic library, so that protocols proved on the basis of the deterministic, Dolev-Yao-like ideal library can be safely implemented with the real cryptographic library. To formulate the theorem, we need additional notation: Let $Sys^{\mathsf{cry,id}}_{n,L}$ denote the ideal cryptographic library for n participants and with length functions and bounds L, and $Sys^{\mathsf{cry,real}}_{n,\mathcal{S},\mathcal{E},L'}$ the real cryptographic library for n participants, based on a secure signature scheme $\mathcal{S}$ and a secure encryption scheme $\mathcal{E}$, and with length functions and bounds L'. Let $RPar$ be the set of valid parameter tuples for the real system, consisting of the number $n \in \mathbb{N}$ of participants, secure signature and encryption schemes $\mathcal{S}$ and $\mathcal{E}$, and length functions and bounds L'. For $(n, \mathcal{S}, \mathcal{E}, L') \in RPar$, let $Sys^{\mathsf{cry,real}}_{n,\mathcal{S},\mathcal{E},L'}$ be the resulting real cryptographic library. Further, let the corresponding length functions and bounds of the ideal system be formalized by a function $L := \mathsf{R2Ipar}(\mathcal{S}, \mathcal{E}, L')$, and let $Sys^{\mathsf{cry,id}}_{n,L}$ be the ideal cryptographic library with parameters n and L. Using the notation of Definition 2, we then have

THEOREM 1. *(Security of Cryptographic Library) For all parameters $(n, \mathcal{S}, \mathcal{E}, L') \in RPar$, we have*

$$Sys^{\mathsf{cry,real}}_{n,\mathcal{S},\mathcal{E},L'} \geq Sys^{\mathsf{cry,id}}_{n,L},$$

where $L := \mathsf{R2Ipar}(\mathcal{S}, \mathcal{E}, L')$.

For proving this theorem, we define a simulator $\mathsf{Sim}_{\mathcal{H}}$ such that even the combination of arbitrary polynomial-time users H and an arbitrary polynomial-time adversary A cannot distinguish the combination of the real machines M_u from the combination $\mathsf{TH}(\mathcal{H})$ and $\mathsf{Sim}_{\mathcal{H}}$ (for all sets $\mathcal{H}$ indicating the correct machines). We first sketch the simulator and then the proof of correct simulation.

5.1 Simulator

Basically $\mathsf{Sim}_{\mathcal{H}}$ has to translate real messages from the real adversary A into handles as $\mathsf{TH}(\mathcal{H})$ expects them at its adversary input port $\mathsf{in_a}?$ and vice versa; see Figure 3. In both directions, $\mathsf{Sim}_{\mathcal{H}}$ has to parse an incoming messages completely because it can only construct the other version (abstract or real) bottom-up. This is done by recursive algorithms. In some cases, the simulator cannot produce any corresponding message. We collect these cases in so-called *error sets* and show later that they cannot occur at all or only with negligible probability.

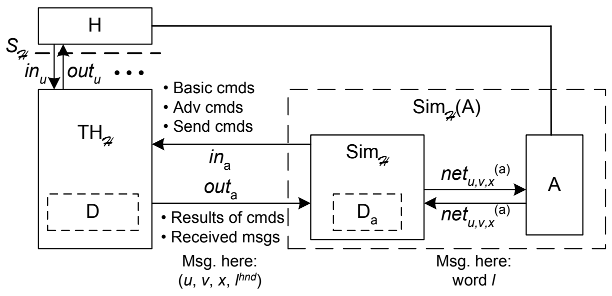

Figure 3: Ports and in- and output types of the simulator.

The state of $\mathsf{Sim}_{\mathcal{H}}$ mainly consists of a database D_a, similar to the databases D_u, but storing the knowledge of the adversary. The behavior of $\mathsf{Sim}_{\mathcal{H}}$ is sketched as follows.

- *Inputs from* $\mathsf{TH}(\mathcal{H})$. Assume that $\mathsf{Sim}_{\mathcal{H}}$ receives an input $(u, v, x, l^{\mathsf{hnd}})$ from $\mathsf{TH}(\mathcal{H})$. If a bitstring l for l^{hnd} already exists in D_a, i.e., this message is already known to the adversary, the simulator immediately outputs l at port $\mathsf{net}_{u,v,x}!$. Otherwise, it first constructs such a bitstring l with a recursive algorithm id2real. This algorithm decomposes the abstract term using basic commands and the adversary command adv_parse. At the same time, id2real builds up a corresponding real bitstring using real cryptographic operations and enters all new message parts into D_a to recognize them when they are reused, both by $\mathsf{TH}(\mathcal{H})$ and by A.

  Mostly, the simulator can construct subterms exactly like the correct machines would do in the real system. Only for encryptions with a public key of a correct machine, adv_parse does not yield the plaintext; thus there the simulator encrypts a fixed message of equal length. This simulation presupposes that all new message parts are of the standard formats, not those resulting from local adversary commands; this is proven correct in the bisimulation.

- *Inputs from* A. Now assume that $\mathsf{Sim}_{\mathcal{H}}$ receives a bitstring l from A at a port $\mathsf{net}_{u,v,x}?$. If l is not a valid list, $\mathsf{Sim}_{\mathcal{H}}$ aborts the transition. Otherwise it translates l into a corresponding handle l^{hnd} by an algorithm real2id, and outputs the abstract sending command $\mathsf{adv\_send\_}x(w, u, l^{\mathsf{hnd}})$ at port $\mathsf{in_a}!$.

  If a handle l^{hnd} for l already exists in D_a, then real2id reuses that. Otherwise it recursively parses a real bitstring using the functional parsing algorithm. At the same time, it builds up a corresponding abstract term in the database of $\mathsf{TH}(\mathcal{H})$. This finally yields the handle l^{hnd}. Furthermore, real2id enters all new subterms into D_a. For building up the abstract term, real2id makes extensive use of the special capabilities of the adversary modeled in $\mathsf{TH}(\mathcal{H})$. In the real system, the bitstring may, e.g., contain a transformed signature, i.e., a new signature for a message for which the correct user has already created another signature. Such a transformation of a signature is not excluded by the definition of secure signature schemes, hence it might occur in the real system. Therefore the simulator also has to be able to insert such a transformed signature into the database of $\mathsf{TH}(\mathcal{H})$, which explains the need for the command adv_transform_signature. Similarly, the adversary might send invalid ciphertexts or simply bitstrings that do not yield a valid type when being parsed. All these cases can be covered by using the special capabilities.

  The only case for which no command exists is a forged signature under a new message. This leads the simulator to abort. (Such runs fall into an error set which is later shown to be negligible.)

As all the commands used by id2real and real2id are local, these algorithms give uninterrupted dialogues between $\mathsf{Sim}_{\mathcal{H}}$ and $\mathsf{TH}(\mathcal{H})$, which do not show up in the views of A and H.

Two important properties have to be shown about the simulator before the bisimulation. First, the simulator has to be polynomial-time. Otherwise, the joint machine $\mathsf{Sim}_{\mathcal{H}}(\mathsf{A})$ of $\mathsf{Sim}_{\mathcal{H}}$ and A might not be a valid polynomial-time adversary on the ideal system. Secondly, it has to be shown that the interaction between $\mathsf{TH}(\mathcal{H})$ and $\mathsf{Sim}_{\mathcal{H}}$ in the recursive algorithms cannot fail because one of the machines reaches its runtime bound. The proof of both properties is quite involved, using an analysis of possible recursion depths depending on the number of existing handles (see [10]).

5.2 Proof of Correct Simulation

Given the simulator, we show that arbitrary polynomial-time users H and an arbitrary polynomial-time adversary A cannot distinguish the combination of the real machine M_u from the combination of $\mathsf{TH}(\mathcal{H})$ and $\mathsf{Sim}_{\mathcal{H}}$. The standard technique in non-cryptographic distributed systems for rigorously proving that two systems have identical visible behaviors is a bisimulation, i.e., one defines a mapping between the respective states and shows that identical inputs in mapped states retain the mapping and produce identical outputs. We need a probabilistic bisimulation because the real system and the simulator are probabilistic, i.e., identical inputs should yield mapped states with the correct probabilities and identically distributed outputs. (For the former, we indeed use mappings, not arbitrary relations for the bisimulation.) In the presence of cryptography and active attacks however, a normal probabilistic bisimulation is still insufficient for three crucial reasons. First, the adversary might succeed in attacking the real system with a very small probability, while this is impossible in the ideal system. This means that we have to cope with *error probabilities*. Secondly, encryption only gives computational indistinguishability, which cannot be captured by a bisimulation, because the actual values in the two systems may be quite different. Thirdly, the adversary might guess a random value, e.g., a nonce that has already been created by some machine but that the adversary has ideally not yet seen. (Formally, "ideally not yet seen" just means that the bisimulation fails if the adversary sends a certain value which already exists in the databases but for which there is no command to give the adversary a handle.) In order to perform a rigorous reduction proof in this case, we have to show that no *partial information* about this

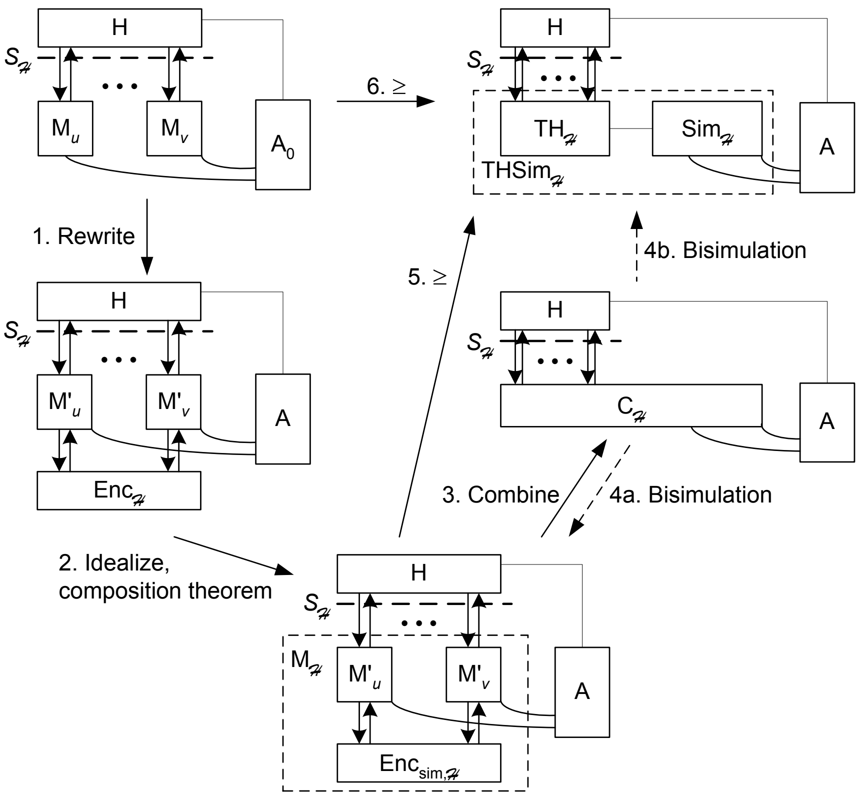

Figure 4: Overview of the proof of correct simulation.

value has already leaked to the adversary because the value was contained in a nested term, or because certain operations would leak partial information. For instance, here the proof would fail if we allowed arbitrary signatures according to the definition of [34], which might divulge previously signed messages, or if we did not additionally randomize probabilistic ciphertexts made with keys of the adversary.

We meet these challenges by first factoring out the computational aspects by a special treatment of ciphertexts. Then we use a new bisimulation technique that includes a static information-flow analysis, and is followed by the remaining cryptographic reductions. The rigorous proof takes 30 pages [10]; hence we can only give a very brief overview here, see also Figure 4.

- *Introducing encryption machines.* We use the two encryption machines $\mathsf{Enc}_{\mathcal{H}}$ and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ from [49] to handle the encryption and decryption needs of the system. Roughly, the first machine calculates the correct encryption of every message m, whereas the second one always encrypts the fixed message $1^{|m|}$ and answers decryption requests for the resulting ciphertexts by table look-up. By [49], $\mathsf{Enc}_{\mathcal{H}}$ is at least as secure as $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$. We rewrite the machines M_u such that they use $\mathsf{Enc}_{\mathcal{H}}$ (Step 1 in Figure 4); this yields modified machines M'_u. We then replace $\mathsf{Enc}_{\mathcal{H}}$ by its idealized counterpart $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ (Step 2 in Figure 4) and use the composition theorem to show that the original system is at least as secure as the resulting system.

- *Combined system.* We now want to compare the combination $\mathsf{M}_{\mathcal{H}}$ of the machines M'_u and $\mathsf{Enc}_{\mathsf{sim},\mathcal{H}}$ with the combination $\mathsf{THSim}_{\mathcal{H}}$ of the machines $\mathsf{TH}(\mathcal{H})$ and $\mathsf{Sim}_{\mathcal{H}}$. However, there is no direct invariant mapping between the states of these two joint machines. Hence we defining an intermediate system $\mathsf{C}_{\mathcal{H}}$ with a state space combined from both these systems (Step 3 in Figure 4).

- *Bisimulations with error sets and information-flow analysis.* We show that the joint view of H and A is equal in interaction with the combined machine $\mathsf{C}_{\mathcal{H}}$ and the two machines $\mathsf{THSim}_{\mathcal{H}}$ and $\mathsf{M}_{\mathcal{H}}$, except for certain runs, which we collect in *error sets*. We show this by performing two bisimulations simultaneously (Step 4 in Figure 4). Transitivity and symmetry of indistinguishability then yield the desired result for $\mathsf{THSim}_{\mathcal{H}}$ and $\mathsf{M}_{\mathcal{H}}$. Besides several normal state invariants of $\mathsf{C}_{\mathcal{H}}$, we also define and prove an information-flow invariant on the variables of $\mathsf{C}_{\mathcal{H}}$.

- *Reduction proofs.* We show that the aggregated probability of the runs in error sets is negligible, as we could otherwise break the underlying cryptography. I.e., we perform reduction proofs against the security definitions of the primitives. For signature forgeries and collisions of nonces or ciphertexts, these are relatively straightforward proofs. For the fact that the adversary cannot guess “official” nonces as well as additional randomizers in signatures and ciphertext, we use the information-flow invariant on the variables of $\mathsf{C}_{\mathcal{H}}$ to show that the adversary has no partial information about such values in situations where correct guessing would put the run in an error set. This proves that $\mathsf{M}_{\mathcal{H}}$ is computationally at least as secure as the ideal system (Step 5 in Figure 4).

Finally, simulatability is transitive [49]. Hence the original real system is also as secure as the ideal system (Step 6 in Figure 4).

6. 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] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Advances in Cryptology: CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 1995.

[4] 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.

[5] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. 11th Symposium on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.

[6] 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.

[7] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. To appear in *Proc. of 23rd Conference on foundations of software technology and theoretical computer science (FSTTCS)*. Preliminary version available from IACR Cryptology ePrint Archive 2003/121, 2003.

[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, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.

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

[11] D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[12] G. Bella, F. Massacci, and L. C. Paulson. The verification of an industrial payment protocol: The set purchase phase. In *Proc. 9th ACM Conference on Computer and Communications Security*, pages 12–20, 2002.

[13] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.

[14] M. Bellare, T. Kohno, and C. Namprempre. Authenticated encryption in ssh: Provably fixing the ssh binary packet protocol. In *Proc. 9th ACM Conference on Computer and Communications Security*, pages 1–11, 2002.

[15] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.

[16] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[17] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[18] R. Canetti. A unified framework for analyzing security of protocols. IACR Cryptology ePrint Archive 2000/067, Dec. 2001. `http://eprint.iacr.org/`.

[19] 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.

[20] 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.

[21] 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.

[22] R. Cramer and V. Shoup. Practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[23] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In *Proc. 6th ACM Conference on Computer and Communications Security*, pages 46–51, 1999.

[24] Z. Dang and R. Kemmerer. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. `http://dimacs.rutgers.edu/Workshops/Security/`.

[25] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[26] Y. Desmedt and K. Kurosawa. How to break a practical mix and design a new one. In *Advances in Cryptology: EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer, 2000.

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

[28] B. Dutertre and S. Schneider. Using a PVS embedding of CSP to verify authentication protocols. In *Proc. International Conference on Theorem Proving in Higher Order Logics (TPHOL)*, volume 1275 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1997.

[29] D. Fisher. Millions of .Net Passport accounts put at risk. *eWeek*, May 2003. (Flaw detected by Muhammad Faisal Rauf Danka).

[30] R. Gennaro, S. Halevi, and T. Rubin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 1999.

[31] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In *Advances in Cryptology: CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 104–110. Springer, 1986.

[32] 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.

[33] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

[34] 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.

[35] J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.

[36] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

[37] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

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

[39] 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.

[40] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[41] 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.

[42] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proc. 18th IEEE Symposium on Security & Privacy*, pages 141–151, 1997.

[43] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.

[44] S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *Proc. 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer, 1992.

[45] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[46] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. Presented at the DERA/RHUL Workshop on Secure Architectures and Information Flow, 1999, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000. `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`.

[47] B. Pfitzmann and M. Waidner. How to break and repair a "provably secure" untraceable payment system. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 1992.

[48] 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.

[49] 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.

[50] C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1992.

[51] P. Rogaway. Authenticated-encryption with associated-data. In *Proc. 9th ACM Conference on Computer and Communications Security*, pages 98–107, 2002.

[52] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.

[53] B. Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 248–262, 2003.

[54] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.