Technical Report: Justifying a Dolev-Yao Model under Active Attacks (Long Version)*

Michael Backes, Birgit Pfitzmann, and Michael Waidner

IBM Zurich Research Lab

July 31, 2008

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.
To illustrate the usefulness of the cryptographic library, we present a cryptographically sound security proof of the well-known Needham-Schroeder-Lowe public-key protocol for entity authentication. This protocol was previously only proved over unfounded abstractions from cryptography. We show that the protocol is secure against arbitrary active attacks if it is implemented using standard provably secure cryptographic primitives. Conducting the proof by means of the idealized cryptographic library does not require us to deal with the probabilistic aspects of cryptography, hence the proof is in the scope of current automated proof tools. Besides establishing the cryptographic security of the Needham-Schroeder-Lowe protocol, this exemplifies the potential of this cryptographic library and paves the way for the cryptographically sound verification of security protocols by automated proof tools.

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 [4, 5], over carefully designed de-facto standards like SSL and PKCS [6, 7], up to current widely deployed products like Microsoft Passport [8]. 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 [9]. 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

* Parts of this work appeared in [1–3].

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., [10, 11].

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 [12], 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.

1.1 A Dolev-Yao Model that is Cryptographically Sound under Active Attacks

Three years ago, efforts started to get the best of both worlds. Essentially, [13, 14] started to define general cryptographic models that support idealization that is secure in arbitrary environments and under arbitrary active attacks, while [15] 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 [16–22].

The conference version [1, 23] underlying this paper is the first one 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. While [1] addressed the soundness of asymmetric cryptographic primitives such as public-key encryption and digital signatures, subsequent papers extended the soundness result to symmetric authentication [24] and

symmetric encryption [25]. Moreover, tailored tool support for this library was subsequently added [26, 27].[1] Combining security against active attacks and nesting cryptographic operations arbitrarily 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 [30, 16]. 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 [17, 19] combine encryption and signatures in a fixed way. Or they need immediate interaction with the adversary [18, 31], 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 [17] 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 [32, 17]. 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, i.e., local names. 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 1.4, and Section 1.5 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

[1] In more recent work, drawing upon insides gained from the proof of the cryptographic library, we showed that widely considered symbolic abstractions of hash functions and of the XOR operation cannot be proven computationally sound in general, hence indicating that their current symbolic representations might be overly simplistic [28, 29].

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 [23] and the extension to symmetric cryptographic operations [24, 25]. 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. In the following, we sketch the ideal cryptographic library in Section 3, the concrete cryptographic realization in Section 4, and the proof of soundness in Section 5. We restrict our attention to asymmetric cryptographic operations in this paper and refer the reader to [24, 25] for the cases of symmetric encryption and message authentication.

1.2 An Illustrating Example – A Cryptographically Sound Proof of the Needham-Schroeder-Lowe Protocol

To illustrate the usefulness of the ideal cryptographic library, we investigate the well-known Needham-Schroeder public-key authentication protocol [4, 33], which arguably constitutes the most prominent example demonstrating the usefulness of the formal-methods approach after Lowe used the FDR model checker to discover a man-in-the-middle attack against the protocol. Lowe later proposed a repaired version of the protocol [34] and used the model checker to prove that this modified protocol (henceforth known as the Needham-Schroeder-Lowe protocol) is secure in the Dolev-Yao model. The original and the repaired Needham-Schroeder public-key protocols are two of the most often investigated security protocols, e.g., [35–38]. Various new approaches and proof tools for the analysis of security protocols were validated by rediscovering the known flaw or proving the fixed protocol in the Dolev-Yao model.

It is well-known and easy to show that the security flaw of the original protocol in the Dolev-Yao model can be used to mount a successful attack against any cryptographic implementation of the protocol. However, all previous security proofs of the repaired protocol are in the Dolev-Yao model, and no theorem carried these results over to the cryptographic approach with its much more comprehensive adversary. We close this gap, i.e., we show that the Needham-Schroeder-Lowe protocol is secure in the cryptographic approach. More precisely, we show that it is secure against arbitrary active attacks, including arbitrary concurrent protocol runs and arbitrary manipulation of bitstrings within polynomial time. The underlying assumption is that the Dolev-Yao-style abstraction of public-key encryption is implemented using a chosen-ciphertext secure public-key encryption scheme with small additions like ciphertext tagging. Chosen-ciphertext security was introduced in [39] and formulated as IND-CCA2 in [40]. Efficient encryption systems secure in this sense exist under reasonable assumptions [41].

Our proof is built upon the ideal cryptographic library, and a composition theorem for the underlying security notion implies that protocol proofs can be made using the ideal library, and security then carries over automatically to the cryptographic realization. However, because of the extension to the Dolev-Yao model, no prior formal-methods proof carries over directly. Our paper therefore validates this approach by the first protocol proof over the new ideal cryptographic library, and cryptographic security follows as a corollary. Besides its value for the Needham-Schroeder-Lowe protocol, the proof shows that in spite of the extensions and differences in presentation with respect to prior Dolev-Yao models, a proof can be made over the new library that seems easily accessible to current automated proof tools. In particular, the proof contains neither probabilism nor computational restrictions. In the following, we express the Needham-Schroeder-Lowe protocol based on the ideal cryptographic library in Section 6 and 7. We formally capture the entity authentication requirement in Section 8, and we prove in Section 9 that entity authentication based on the ideal library implies (the cryptographic definition of) entity authentication based on the concrete realization of the library. Finally, we prove the entity authentication property based on the ideal library in Section 10.

1.3 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 [42, 43]. 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., [44, 45]. The early methods of automating proofs based on the Dolev-Yao model are summarized in [46]. More recently, such work concentrated on using existing general-purpose model checkers [34, 47, 48] and theorem provers [49, 50], and on treating larger protocols, e.g., [51].

Work intended to bridge the gap between the cryptographic approach and the use of automated tools started independently with [13, 14] and [30]. In [30], 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 [16] 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, [52] specifically considers strand spaces and information-theoretically secure authentication only. In [53] a deduction system for information flow is based on the same operations as in [30], still under passive attacks only.

The approach in [13, 14] 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 [54–57]; earlier extensions towards reactive systems were either without real abstraction [32] or for quite special cases [58]. The approach was extended from synchronous to asynchronous systems in [17, 18]. 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 [19, 59] using the theorem prover PVS [60]. 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, secrecy, liveness, and non-interference [59, 61–65].

Concurrently to [1], an extension to asymmetric encryption, but still under passive attacks only, has been presented in [66]. The underlying masters thesis [67] considers asymmetric encryption under active attacks, but in the random oracle model, which is itself an idealization of cryptography and not justifiable [68]. Laud [69] has subsequently presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work enjoys a direct connection with a formal proof tool, but it is specific to certain confidentiality properties, restricts the surrounding protocols to straight-line programs in a specific language, and does not address a connection to the remaining primitives of the Dolev-Yao model. Herzog et al. [66] and Micciancio and Warinschi [70] have subsequently also given a cryptographic underpinning under active attacks. Their results are narrower than that in [1] since they are specific for public-key encryption, but consider slightly simpler real implementations; moreover, the former relies on a stronger assumption whereas the latter severely restricts the classes of protocols and protocol properties that can be analyzed using this primitive. Section 6 of [70] further points out several possible extensions of their work which all already exist in the earlier work of [1]. Recently, Canetti and Herzog [71] have linked ideal functionalities for mutual authentication and key exchange protocols to corresponding representations in a formal language. They apply their techniques to the Needham-Schroeder-Lowe protocol by considering the exchanged nonces as secret keys. Their work is restricted to the mentioned functionalities and in contrast to the universally composable library [1] hence does not address soundness of Dolev-Yao models in their usual generality. The considered language does not allow loops and offers public-key encryption as the only cryptographic operation. Moreover, their approach to define a mapping between ideal and real traces following the ideas of [70] only captures trace-based properties (i.e., integrity properties); reasoning about secrecy properties additionally requires ad-hoc and functionality-specific arguments.

Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [72, 32, 73, 74] and, as a second step, to encode them into proof tools. This approach can not yet handle protocols with any degree of automation. It is complementary to the approach of proving simple deter-

ministic abstractions of cryptography and working with those wherever cryptography is only used in a blackbox way.

The first cryptographically sound security proofs of the Needham-Schroeder-Lowe protocol have been presented concurrently and independently in [2] and [75]. While the first paper conducts the proof by means of the ideal cryptographic library and hence within a deterministic framework that is accessible for machine-assisted verification, the proof in the second paper is done from scratch in the cryptographic approach and is hence vulnerable to the aforementioned problems. On the other hand, the second paper proves stronger properties; we discuss this in Section 8. It further shows that chosen-plaintext-secure encryption is insufficient for the security of the protocol. While certainly no full Dolev-Yao model would be needed to model just the Needham-Schroeder-Lowe protocol, there was no prior attempt to prove this or a similar cryptographic protocol based on a sound abstraction from cryptography in a way accessible to automated proof tools. After the Needham-Schroeder-Lowe protocol was soundly analyzed, a variety of additional protocols were proven to be secure in a computationally sound manner, e.g., [76–80]

1.4 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 $\{\mathsf{s}, \mathsf{r}, \mathsf{i}\}$, 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 subsequently added symmetric authentication [24] and symmetric encryption [25]).

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 [43] 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 [43, 81, 82].[2]
- We cannot (easily) allow participants to send secret keys over the network because then the simulation is not always possible.[3] 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.

1.5 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.

[2] Memory-less schemes exist with either lower efficiency or based on stronger assumptions (e.g., [83–85]). We could add them to the library as an additional primitive.

[3] The primitives become "committing". This is well-known from individual simulation proofs. It also explains why [30] is restricted to passive attacks.

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 [86].

- 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 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 [17]. A *system* consists of several possible *structures*. A structure consists of a set $\hat{M}$ 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 [17] 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 $(\hat{M}_1, S) \in Sys_{\mathsf{real}}$, every polynomial-time user H, and every polynomial-time adversary A_1, there exists a polynomial-time adversary A_2 on a corresponding ideal structure $(\hat{M}_2, S) \in Sys_{\mathsf{id}}$ such that the view of H is computationally indistinguishable in the two configurations. This is illustrated in Figure 1. Indistinguishability is a well-known cryptographic notion from [87].

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*,* $\exists k_0 \forall k \geq k_0 \colon\ g(k) \leq 1/Q(k)$. $\diamond$

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* $conf_1 = (\hat{M}_1, S, \mathsf{H}, \mathsf{A}_1) \in \mathsf{Conf}(Sys_{\mathsf{real}})$*, there exists a polynomial-time configuration* $conf_2 = (\hat{M}_2, S, \mathsf{H}, \mathsf{A}_2) \in \mathsf{Conf}(Sys_{\mathsf{id}})$ *(with the same* H*) such that* $view_{conf_1}(\mathsf{H}) \approx view_{conf_2}(\mathsf{H})$. $\diamond$

For the cryptographic library, we even show blackbox simulatability, i.e., A_2 consists of a simulator Sim that depends only on $(\hat{M}_1, S)$ and uses A_1 as a blackbox submachine. An essential feature of this definition of simulatability is a composition theorem [17], 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}.

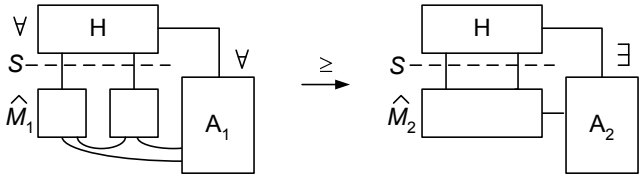

Fig. 1. Simulatability: The two views of H must be indistinguishable.

2.2 Notation

We write ":=" for deterministic and "$\leftarrow$" for probabilistic assignment, and "$\xleftarrow{\mathcal{R}}$" 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 $\mathsf{len}(m)$, 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 1.4, 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 [23].

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

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

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

- $ind \in \mathbb{N}_0$ is called the *index* of x. We write $D[i]$ instead of $D[ind = i]$.
- $type \in typeset := \{\mathsf{data}, \mathsf{list}, \mathsf{nonce}, \mathsf{ske}, \mathsf{pke}, \mathsf{enc}, \mathsf{sks}, \mathsf{pks}, \mathsf{sig}, \mathsf{garbage}\}$ identifies the *type* of x. Future extensions of the library can extend this set.
- $arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty *list of arguments*.
- $hnd_u \in \mathbb{N}_0 \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ identifies how u knows this entry. The value a represents the adversary, and $hnd_u = \downarrow$ indicates that u does not know this entry. A value $hnd_u \neq \downarrow$ 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}.
- $len \in \mathbb{N}_0$ 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 $ind := size$++, and $x.ind$ 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 $i^{\mathsf{hnd}} := curhnd_u$++.

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 $(cmd, x_1, \ldots, x_j)$. We usually write it $y \leftarrow cmd(x_1, \ldots, x_j)$ 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.

Basic Commands First, we have a set $basic\_cmds$ 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

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

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 [23] 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 nonce_len*, enc_len*, and 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 $n^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

$$D :\Leftarrow (ind := size\mathsf{++}, type := \mathsf{nonce}, arg := (),\\ hnd_u := n^{\mathsf{hnd}}, len := \mathsf{nonce\_len}^*(k)).$$

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 $D[pred]$) 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 $c^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

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

Honest Send Commands Secondly, we have a set $send\_cmds := \{\mathsf{send\_s}, \mathsf{send\_r}, \mathsf{send\_i}\}$ 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.

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.

Adversary Send Commands Fourthly, we have a set $adv\_send\_cmds := \{\mathsf{adv\_send\_s}, \mathsf{adv\_send\_r}, \mathsf{adv\_send\_i}\}$ 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

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

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 [43]. 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 [39] and formalized as "IND-CCA2" in [40]. It is the accepted definition for general-purpose encryption. An efficient encryption system secure in this sense is [41]. 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 $x \in \{\mathsf{s}, \mathsf{r}, \mathsf{i}\}$. 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 [17] 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.

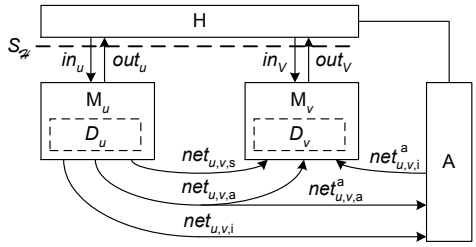

Fig. 2. Connections from a correct machine to another in the real system.

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.

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 1.5. 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 [23].

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 $n' \xleftarrow{R} \{0,1\}^{\mathsf{nonce\_len}(k)}$, $n := (\mathsf{nonce}, n')$, $n^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ 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 1.5.

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 $pk := D_u[pk^{\mathsf{hnd}}].word[2]$, $l := D_u[l^{\mathsf{hnd}}].word$, $r \xleftarrow{R} \{0,1\}^{\mathsf{nonce\_len}(k)}$, encrypt $c \leftarrow \mathsf{E}_{pk}((r,l))$, and set $c^* := (\mathsf{enc}, pk, c, r)$. If $c^* = \downarrow$ or $\mathsf{len}(c^*) > \mathsf{max\_len}(k)$ then return $\downarrow$, else set $c^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ 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.

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.

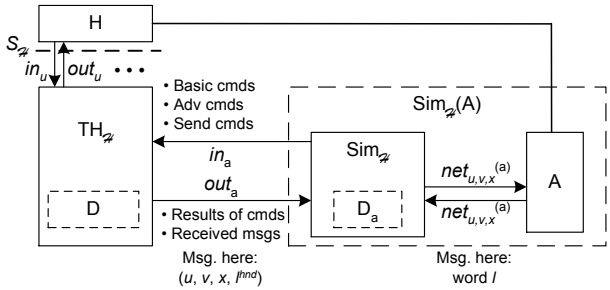

Fig. 3. Ports and in- and output types of the simulator.

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 $\mathsf{in}_{\mathsf{a}}!$.
  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 [23]).

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

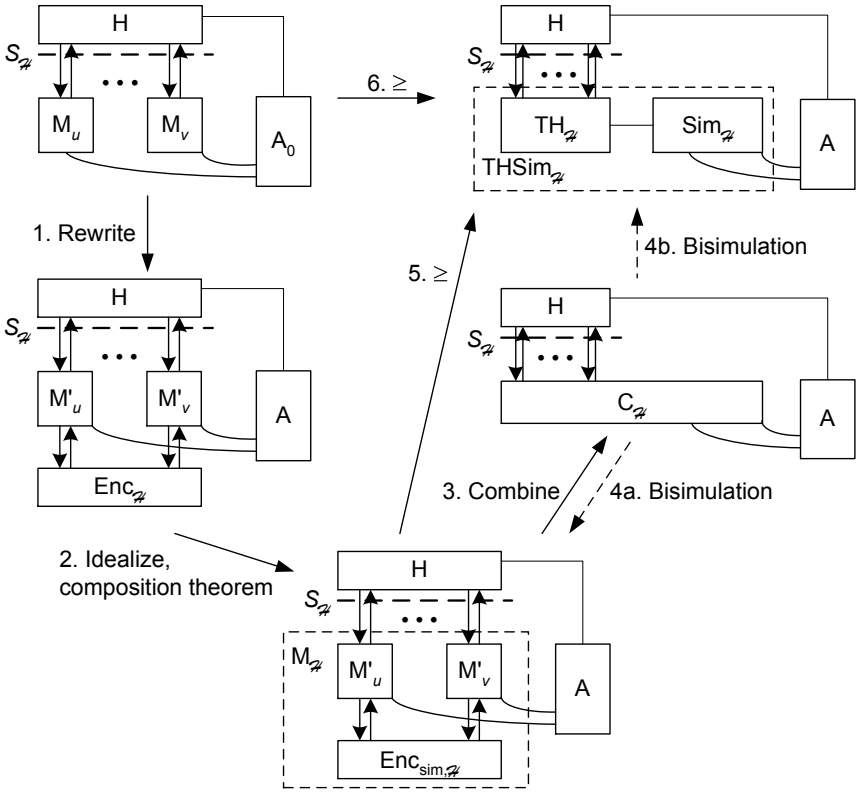

Fig. 4. Overview of the proof of correct simulation.

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 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 [43], 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 [23]; 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 [17] 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 $m_{sim}\mathsf{len}(m)$ and answers decryption requests for the resulting ciphertexts by table look-up. By [17], $\mathsf{Enc}_{\mathcal{H}}$ 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 $\hat{C}_{\mathcal{H}}$ 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 $\hat{C}_{\mathcal{H}}$ 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 $\hat{C}_{\mathcal{H}}$, we also define and prove an information-flow invariant on the variables of $\hat{C}_{\mathcal{H}}$.
- *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 $\hat{C}_{\mathcal{H}}$ 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 [17]. Hence the original real system is also as secure as the ideal system (Step 6 in Figure 4).

6 The Needham-Schroeder-Lowe Protocol

The original Needham-Schroeder public-key protocol and Lowe's variant consist of seven steps. Four steps deal with key generation and public-key distribution. They are usually omitted in a security analysis, and it is simply assumed that keys have already been generated and distributed. We do this as well to keep the proof short. However, the underlying cryptographic library offers commands for modeling these steps as well. The main part of the Needham-Schroeder-Lowe public-key protocol consists of the following three steps, expressed in the typical protocol notation, as in, e.g., [33].

1. $u \to v : E_{pk_v}(N_u, u)$
2. $v \to u : E_{pk_u}(N_u, N_v, v)$
3. $u \to v : E_{pk_v}(N_v)$.

Here, user u seeks to establish a session with user v. He generates a nonce N_u and sends it to v together with his identity, encrypted with v's public key (first message). Upon receiving this message, v decrypts it to obtain the nonce N_u. Then v generates a new nonce N_v and sends both nonces and her identity back to u, encrypted with u's public key (second message). Upon receiving this message, u decrypts it and tests whether the contained identity v equals the sender of the message and whether u earlier sent the first contained nonce to user v. If yes, u sends the second nonce back to v, encrypted with v's public key (third message). Finally, v decrypts this message; and if v had earlier sent the contained nonce to u, then v believes to speak with u.

7 The Needham-Schroeder-Lowe Protocol Using the Dolev-Yao-style Cryptographic Library

Almost all formal proof techniques for protocols such as Needham-Schroeder-Lowe first need a reformulation of the protocol into a more detailed version than the three steps above. These details include necessary tests on received messages, the types and generation rules for values like u and N_u, and a surrounding framework specifying the number of participants, the possibilities of multiple protocol runs, and the adversary capabilities. The same is true when using the Dolev-Yao-style cryptographic library

from [1], i.e., it plays a similar role in our proof as "the CSP Dolev-Yao model" or "the inductive-approach Dolev-Yao model" in other proofs. Our protocol formulation in this framework is given in Algorithms 1 and 2.[4] We first explain this formulation, and then consider general aspects of the surrounding framework as far as needed in our proofs.

7.1 Detailed Protocol Descriptions

Recall that the underlying framework is automata-based, i.e., protocols are executed by interacting machines, and event-based, i.e., machines react on received inputs. By $\mathsf{M}_i^{\mathsf{NS}}$ we denote the Needham-Schroeder machine for a participant i; it can act in the roles of both u and v above.

The first type of input that $\mathsf{M}_i^{\mathsf{NS}}$ can receive is a start message $(\mathsf{new\_prot}, v)$ from its user denoting that it should start a protocol run with user v. The number of users is called n. User inputs are distinguished from network inputs by arriving at a port $\mathsf{EA\_in}_u?$. The "EA" stands for entity authentication because the user interface is the same for all entity authentication protocols. The reaction on this input, i.e., the sending of the first message, is described in Algorithm 1.

Algorithm 1 Evaluation of User Inputs in $\mathsf{M}_u^{\mathsf{NS}}$

Input: $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$ with $v \in \{1, \ldots, n\} \setminus \{u\}$.
1: $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
2: $Nonce_{u,v} := Nonce_{u,v} \cup \{n_u^{\mathsf{hnd}}\}$.
3: $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
4: $l_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(n_u^{\mathsf{hnd}}, u^{\mathsf{hnd}})$.
5: $c_1^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_{v,u}^{\mathsf{hnd}}, l_1^{\mathsf{hnd}})$.
6: $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(c_1^{\mathsf{hnd}})$.
7: $\mathsf{send\_i}(v, m_1^{\mathsf{hnd}})$.

The command gen_nonce generates the nonce. $\mathsf{M}_u^{\mathsf{NS}}$ adds the result n_u^{hnd} to a set $Nonce_{u,v}$ for future comparison. The command store inputs arbitrary application data into the cryptographic library, here the user identity u. The command list forms a list and encrypt is encryption. The final command send_i means that $\mathsf{M}_u^{\mathsf{NS}}$ attempts to send the resulting term to v over an insecure channel. The list operation directly before sending is a technicality: recall that only lists are allowed to be sent in this library because the list operation concentrates verifications that no secret items are put into messages.

The behavior of the Needham-Schroeder machine of participant u upon receiving a network input is defined similarly in Algorithm 2. The input arrives at port $\mathsf{out}_u?$ and is of the form $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ where v is the supposed sender, i denotes that the channel is insecure, and m^{hnd} is a handle to a list. The port $\mathsf{out}_u?$ is connected to the cryptographic library, whose two implementations represent the obtained Dolev-Yao-style term or real bitstring, respectively, to the protocol in a unified way by a handle.

[4] For some frameworks there are compilers to generate these detailed protocol descriptions, e.g., [88]. This should be possible for this framework in a similar way.

Algorithm 2 Evaluation of Network Inputs in $\mathsf{M}_u^{\mathsf{NS}}$

Input: $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ at out_u? with $v \in \{1, \ldots, n\} \setminus \{u\}$.

$c^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 1)$
$l^{\mathsf{hnd}} \leftarrow \mathsf{decrypt}(ske_u^{\mathsf{hnd}}, c^{\mathsf{hnd}})$
$x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$ for $i = 1, 2, 3$.
if $x_1^{\mathsf{hnd}} \neq \downarrow \wedge\, x_2^{\mathsf{hnd}} \neq \downarrow \wedge\, x_3^{\mathsf{hnd}} = \downarrow$ **then** {First Message is input}
  $x_2 \leftarrow \mathsf{retrieve}(x_2^{\mathsf{hnd}})$.
  if $x_2 \neq v$ **then**
    Abort
  end if
  $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
  $Nonce_{u,v} := Nonce_{u,v} \cup \{n_u^{\mathsf{hnd}}\}$.
  $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
  $l_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(x_1^{\mathsf{hnd}}, n_u^{\mathsf{hnd}}, u^{\mathsf{hnd}})$.
  $c_2^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_{v,u}^{\mathsf{hnd}}, l_2^{\mathsf{hnd}})$.
  $m_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(c_2^{\mathsf{hnd}})$.
  $\mathsf{send\_i}(v, m_2^{\mathsf{hnd}})$.
else if $x_1^{\mathsf{hnd}} \neq \downarrow \wedge\, x_2^{\mathsf{hnd}} \neq \downarrow \wedge\, x_3^{\mathsf{hnd}} \neq \downarrow$ **then** {Second Message is input}
  $x_3 \leftarrow \mathsf{retrieve}(x_3^{\mathsf{hnd}})$.
  if $x_3 \neq v \vee x_1^{\mathsf{hnd}} \notin Nonce_{u,v}$ **then**
    Abort
  end if
  $l_3^{\mathsf{hnd}} \leftarrow \mathsf{list}(x_2^{\mathsf{hnd}})$.
  $c_3^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_{v,u}^{\mathsf{hnd}}, l_3^{\mathsf{hnd}})$.
  $m_3^{\mathsf{hnd}} \leftarrow \mathsf{list}(c_3^{\mathsf{hnd}})$.
  $\mathsf{send\_i}(v, m_3^{\mathsf{hnd}})$.
else if $x_1^{\mathsf{hnd}} \in Nonce_{u,v} \wedge\; x_2^{\mathsf{hnd}} = x_3^{\mathsf{hnd}} = \downarrow$ **then** {Third Message is input}
  Output (ok, v) at $\mathsf{EA\_out}_u$!.
end if

In this algorithm, the protocol machine first decrypts the list content using its secret key; this yields a handle l^{hnd} to an inner list. This list is parsed into at most three components using the command list_proj. If the list has two elements, i.e., it could correspond to the first message of the protocol, and if it contains the correct identity, the machine generates a new nonce and stores its handle in the set $Nonce_{u,v}$. Then it builds up a new list according to the protocol description, encrypts it and sends it to user v. If the list has three elements, i.e., it could correspond to the second message of the protocol, the machine tests whether the third list element equals v and the first list element is contained in the set $Nonce_{u,v}$. If one of these tests does not succeed, $\mathsf{M}_u^{\mathsf{NS}}$ aborts. Otherwise, it again builds up a term according to the protocol description and sends it to user v. Finally, if the list has only one element, i.e., it could correspond to the third message of the protocol, the machine tests if the handle of this element is contained in $Nonce_{u,v}$. If so, $\mathsf{M}_u^{\mathsf{NS}}$ outputs (ok, v) at $\mathsf{EA\_out}_u!$. This signals to user u that the protocol with user v has terminated successfully, i.e., u believes to speak with v.

Both algorithms should immediately abort the handling of the current input if a cryptographic command does not yield the desired result, e.g., if a decryption fails. For readability we omitted this in the algorithm descriptions; instead we impose the following convention on both algorithms.

Convention 1 *If* $\mathsf{M}_u^{\mathsf{NS}}$ *receives* $\downarrow$ *as the answer of the cryptographic library to a command, then* $\mathsf{M}_u^{\mathsf{NS}}$ *aborts the execution of the current algorithm, except for the command types* list_proj *or* send_i.

We refer to Step i of Algorithm j as Step $j.i$.

7.2 Overall Framework and Adversary Model

When protocol machines such as $\mathsf{M}_u^{\mathsf{NS}}$ for certain users $u \in \{1, \ldots, n\}$ are defined, there is no guarantee that all these machines are correct. A trust model determines for what subsets $\mathcal{H}$ of $\{1, \ldots, n\}$ we want to guarantee anything; in our case this is essentially for all subsets: We aim at entity authentication between u and v whenever $u, v \in \mathcal{H}$ and thus whenever $\mathsf{M}_u^{\mathsf{NS}}$ and $\mathsf{M}_v^{\mathsf{NS}}$ are correct. Incorrect machines disappear and are replaced by the adversary. Each set of potential correct machines together with its user interface constitute a structure, and the set of these structures is called the system, cf. Section 2.2. Recall further that when considering the security of a structure, an arbitrary probabilistic machine H is connected to the user interface to represent all users, and an arbitrary machine A is connected to the remaining free ports (typically the network) and to H to represent the adversary, see Fig. 5. In polynomial-time security proofs, H and A are polynomial-time.

This setting implies that any number of concurrent protocol runs with both honest participants and the adversary are considered because H and A can arbitrarily interleave protocol start inputs $(\mathsf{new\_prot}, v)$ with the delivery of network messages.

For a set $\mathcal{H}$ of honest participants, the user interface of the ideal and real cryptographic library is the port set $S_{\mathcal{H}}^{\mathsf{cry}} := \{\mathsf{in}_u?, \mathsf{out}_u! \mid u \in \mathcal{H}\}$. This is where the Needham-Schroeder machines input their cryptographic commands and obtain results

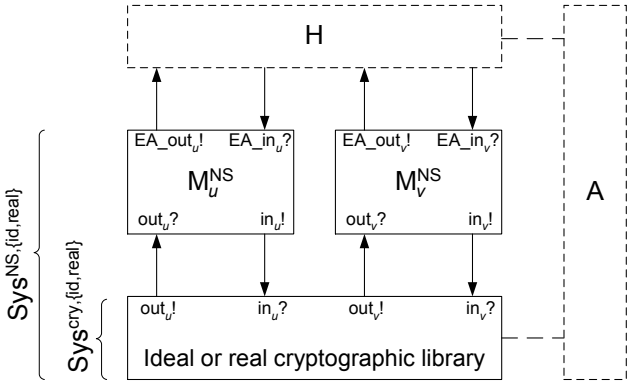

Fig. 5. Overview of the Needham-Schroeder-Lowe Ideal System

and received messages. In the ideal case this interface is served by just one machine $\mathsf{TH}_{\mathcal{H}}$ called trusted host which essentially administrates Dolev-Yao-style terms under the handles. In the real case, the same interface is served by a set $\hat{M}^{\mathsf{cry}}_{\mathcal{H}} := \{\mathsf{M}^{\mathsf{cry}}_{u,\mathcal{H}} \mid u \in \mathcal{H}\}$ of real cryptographic machines. The corresponding systems are called $Sys^{\mathsf{cry,id}} := \{(\{\mathsf{TH}_{\mathcal{H}}\}, S^{\mathsf{cry}}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1,\ldots,n\}\}$ and $Sys^{\mathsf{cry,real}} := \{(\hat{M}^{\mathsf{cry}}_{\mathcal{H}}, S^{\mathsf{cry}}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1,\ldots,n\}\}$.

The user interface of the Needham-Schroeder machines or any other entity authentication protocol is $S^{\mathsf{EA}}_{\mathcal{H}} := \{\mathsf{EA\_in}_u?, \mathsf{EA\_out}_u! \mid u \in \mathcal{H}\}$. The ideal and real Needham-Schroeder-Lowe systems serving this interface differ only in the cryptographic library. With $\hat{M}^{\mathsf{NS}}_{\mathcal{H}} := \{\mathsf{M}^{\mathsf{NS}}_u \mid u \in \mathcal{H}\}$, they are $Sys^{\mathsf{NS,id}} := \{(\hat{M}^{\mathsf{NS}}_{\mathcal{H}} \cup \{\mathsf{TH}_{\mathcal{H}}\}, S^{\mathsf{EA}}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1,\ldots,n\}\}$ and $Sys^{\mathsf{NS,real}} := \{(\hat{M}^{\mathsf{NS}}_{\mathcal{H}} \cup \hat{M}^{\mathsf{cry}}_{\mathcal{H}}, S^{\mathsf{EA}}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1,\ldots,n\}\}$.

7.3 Initial State

We have assumed in the algorithms that each Needham-Schroeder machine $\mathsf{M}^{\mathsf{NS}}_u$ already has a handle ske^{hnd}_u to its own secret encryption key and handles $pke^{\mathsf{hnd}}_{v,u}$ to the corresponding public keys of every participant v. The cryptographic library can also represent key generation and distribution by normal commands. Formally, this assumption means that for each participant u two entries of the following form are added to D where $\mathcal{H} = \{u_1,\ldots,u_m\}$:

$$(ske_u, type := \mathsf{ske}, arg := (), hnd_u := ske^{\mathsf{hnd}}_u, len := 0);$$
$$(pke_u, type := \mathsf{pke}, arg := (), hnd_{u_1} := pke^{\mathsf{hnd}}_{u,u_1}, \ldots,$$
$$hnd_{u_m} := pke^{\mathsf{hnd}}_{u,u_m}, hnd_{\mathsf{a}} := pke^{\mathsf{hnd}}_{u,\mathsf{a}}, len := \mathsf{pke\_len}^*(k)).$$

Here ske_u and pke_u are two consecutive natural numbers and $\mathsf{pke\_len}^*$ is the length function for public keys. Treating the secret key length as 0 is a technicality in [1] and will not matter here. Further, each machine $\mathsf{M}^{\mathsf{NS}}_u$ contains the bitstring u denoting its identity, and the family $(Nonce_{u,v})_{v\in\{1,\ldots,n\}}$ of initially empty sets of (nonce) handles.

7.4 On Polynomial Runtime

In order to be valid users of the real cryptographic library, the machines $\mathsf{M}_u^{\mathsf{NS}}$ have to be polynomial-time. We therefore define that each machine $\mathsf{M}_u^{\mathsf{NS}}$ maintains explicit polynomial bounds on the accepted message lengths and the number of inputs accepted at each port. As this is done exactly as in the cryptographic library, we omit the rigorous write-up.

8 The Security Property

Our security property states that an honest participant v only successfully terminates a protocol with an honest participant u if u has indeed started a protocol with v, i.e., an output (ok, u) at $\mathsf{EA\_out}_v!$ can only happen if there was a prior input $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$. This property and also the actual protocol does not consider replay attacks, i.e., a user v could successfully terminate a protocol with u multiple times while u started a protocol with v only once. However, this can easily be avoided as follows: If $\mathsf{M}_u^{\mathsf{NS}}$ receives a message from v containing one of its own nonces, it additionally removes this nonce from the corresponding set, i.e., it removes x_1^{hnd} from $Nonce_{u,v}$ after Steps 2.20 and 2.25. Proving freshness given this change and mutual authentication is useful future work, but better done once the proof has been automated. Warinschi proves these properties [75]. The even stronger property of matching conversations from [9] that he also proves makes constraints on events within the system, not only at the interface. We thus regard it as an overspecification in an approach based on abstraction.

Integrity properties in the underlying model are formally sets of traces at the user interfaces of a system, i.e., here at the port sets $S_{\mathcal{H}}^{\mathsf{EA}}$. Intuitively, an integrity property Req contains the "good" traces at these ports. A trace is a sequence of sets of events. We write an event $p?m$ or $p!m$, meaning that message m occurs at in- or output port p. The t-th step of a trace r is written r_t; we speak of the step at time t. The integrity requirement Req^{EA} for the Needham-Schroeder-Lowe protocol is defined as follows, meaning that if v believes to speak with u at time t_1, then there exists a past time t_0 where u started a protocol with v:

Definition 3. *(Entity Authentication Requirement) A trace r is contained in Req^{EA} if for all $u, v \in \mathcal{H}$:*

$$\forall t_1 \in \mathbb{N}\colon \mathsf{EA\_out}_v!(\mathsf{ok}, u) \in r_{t_1}$$
$$\Rightarrow \exists t_0 < t_1\colon \mathsf{EA\_in}_u?(\mathsf{new\_prot}, v) \in r_{t_0}.$$

$\diamond$

The notion of a system Sys fulfilling an integrity property Req essentially comes in two flavors [59]. *Perfect fulfillment*, $Sys \models^{\mathsf{perf}} Req$, means that the integrity property holds for all traces arising in runs of Sys (a well-defined notion from the underlying model [17]). *Computational fulfillment*, $Sys \models^{\mathsf{poly}} Req$, means that the property only holds for polynomially bounded users and adversaries, and that a negligible error probability is permitted. Perfect fulfillment implies computational fulfillment.

The following theorem captures the security of the Needham-Schroeder-Lowe protocol; we prove it in the rest of the paper.

Theorem 2. *(Security of the Needham-Schroeder-Lowe Protocol) For the Needham-Schroeder-Lowe systems from Section 7.2 and the integrity property of Definition 3, we have* $Sys^{\mathsf{NS,id}} \models^{\mathsf{perf}} Req^{\mathsf{EA}}$ *and* $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$. □

9 Proof of the Cryptographic Realization from the Idealization

As discussed in the introduction, the idea of our approach is to prove Theorem 2 for the protocol using the ideal Dolev-Yao-style cryptographic library. Then the result for the real system follows automatically. As this paper is the first instantiation of this argument, we describe it in detail.

The notion that a system Sys_1 securely implements another system Sys_2 reactive simulatability (recall the introduction), is written $Sys_1 \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys_2$ (in the computational case). The main result of [1] is therefore

$$Sys^{\mathsf{cry,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{cry,id}}. \tag{1}$$

Since $Sys^{\mathsf{NS,real}}$ and $Sys^{\mathsf{NS,id}}$ are compositions of the same protocol with $Sys^{\mathsf{cry,real}}$ and $Sys^{\mathsf{cry,id}}$, respectively, the composition theorem of [17] and (1) imply

$$Sys^{\mathsf{NS,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{NS,id}}. \tag{2}$$

Showing the theorem's preconditions is easy since the machines $\mathsf{M}_u^{\mathsf{NS}}$ are polynomial-time (see Section 7.4). Finally, the integrity preservation theorem from [59] and (2) imply for every integrity requirement Req that

$$(Sys^{\mathsf{NS,id}} \models^{\mathsf{poly}} Req) \Rightarrow (Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req). \tag{3}$$

Hence if we prove $Sys^{\mathsf{NS,id}} \models^{\mathsf{perf}} Req^{\mathsf{EA}}$, we immediately obtain $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$.

10 Proof in the Ideal Setting

This section contains the proof of the ideal part of Theorem 2: We prove that the Needham-Schroeder-Lowe protocol implemented with the ideal, Dolev-Yao-style cryptographic library perfectly fulfills the integrity requirement Req^{EA}. The proof idea is to go backwards in the protocol step by step, and to show that a specific output always requires a specific prior input. For instance, when user v successfully terminates a protocol with user u, then u has sent the third protocol message to v; thus v has sent the second protocol message to u; and so on. The main challenge in this proof was to find suitable invariants on the state of the ideal Needham-Schroeder-Lowe system.

We start by formulating the invariants and then prove the overall entity authentication requirement from the invariants. Finally we prove the invariants, after describing detailed state transitions of the ideal cryptographic library as needed in that proof.

10.1 Invariants

This section contains invariants of the system $Sys^{\mathsf{NS,id}}$, which are used in the proof of Theorem 2. The first invariants, *correct nonce owner* and *unique nonce use*, are easily proved and essentially state that handles contained in a set $Nonce_{u,v}$ indeed point to entries of type nonce, and that no nonce is in two such sets. The next two invariants, *nonce secrecy* and *nonce-list secrecy*, deal with the secrecy of certain terms. They are mainly needed to prove the last invariant, *correct list owner*, which establishes who created certain terms.

- *Correct Nonce Owner.* For all $u \in \mathcal{H}, v \in \{1,\ldots,n\}$ and $x^{\mathsf{hnd}} \in Nonce_{u,v}$, we have $D[hnd_u = x^{\mathsf{hnd}}].type = \mathsf{nonce}$.
- *Unique Nonce Use.* For all $u, v \in \mathcal{H}$, all $w, w' \in \{1,\ldots,n\}$, and all $j \leq size$: If $D[j].hnd_u \in Nonce_{u,w}$ and $D[j].hnd_v \in Nonce_{v,w'}$, then $(u, w) = (v, w')$.

Nonce secrecy states that the nonces exchanged between honest users u and v remain secret from all other users and from the adversary, i.e., that the other users and the adversary have no handles to such a nonce:

- *Nonce Secrecy.* For all $u, v \in \mathcal{H}$ and all $j \leq size$: If $D[j].hnd_u \in Nonce_{u,v}$ then $D[j].hnd_w = \downarrow$ for all $w \in (\mathcal{H} \cup \{\mathsf{a}\}) \setminus \{u, v\}$.

Similarly, the invariant *nonce-list secrecy* states that a list containing such a nonce can only be known to u and v. Further, it states that the identity fields in such lists are correct for Needham-Schroeder-Lowe messages. Moreover, if such a list is an argument of another entry, then this entry is an encryption with the public key of u or v.

- *Nonce-List Secrecy.* For all $u, v \in \mathcal{H}$ and all $j \leq size$ with $D[j].type = \mathsf{list}$: Let $x_i^{\mathsf{ind}} := D[j].arg[i]$ for $i = 1, 2, 3$. If $D[x_i^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$ then:
  a) $D[j].hnd_w = \downarrow$ for all $w \in (\mathcal{H} \cup \{\mathsf{a}\}) \setminus \{u, v\}$.
  b) If $D[x_{i+1}^{\mathsf{ind}}].type = \mathsf{data}$, then $D[x_{i+1}^{\mathsf{ind}}].arg = (u)$.
  c) For all $k \leq size$ we have $j \in D[k].arg$ only if $D[k].type = \mathsf{enc}$ and $D[k].arg[1] \in \{pke_u, pke_v\}$.

The invariant *correct list owner* states that certain protocol messages can only be constructed by the "intended" users. For instance, if a database entry is structured like the cleartext of a first protocol message, i.e., it is of type list, its first argument belongs to the set $Nonce_{u,v}$, and its second argument is non-cryptographic, i.e., of type data, then it has been created by user u. Similar statements exist for the second and third protocol message.

- *Correct List Owner.* For all $u, v \in \mathcal{H}$ and all $j \leq size$ with $D[j].type = \mathsf{list}$: Let $x_i^{\mathsf{ind}} := D[j].arg[i]$ and $x_{i,u}^{\mathsf{hnd}} := D[x_i^{\mathsf{ind}}].hnd_u$ for $i = 1, 2$.
  a) If $x_{1,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ and $D[x_2^{\mathsf{ind}}].type = \mathsf{data}$, then $D[j]$ was created by $\mathsf{M}_u^{\mathsf{NS}}$ in Step 1.4.
  b) If $D[x_1^{\mathsf{ind}}].type = \mathsf{nonce}$ and $x_{2,u}^{\mathsf{hnd}} \in Nonce_{u,v}$, then $D[j]$ was created by $\mathsf{M}_u^{\mathsf{NS}}$ in Step 2.12.
  c) If $x_{1,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ and $x_2^{\mathsf{ind}} = \downarrow$, then $D[j]$ was created by $\mathsf{M}_v^{\mathsf{NS}}$ in Step 2.21.

This invariant is key for proceeding backwards in the protocol. For instance, if v terminates a protocol with user u, then v must have received a third protocol message. *Correct list owner* implies that this message has been generated by u. Now u only constructs such a message if it received a second protocol message. Applying the invariant two more times shows that u indeed started a protocol with v. The proof described below will take care of the details. Formally, the invariance of the above statements is captured in the following lemma.

Lemma 1. *The statements* correct nonce owner, unique nonce use, nonce secrecy, nonce-list secrecy, *and* correct list owner *are invariants of* $Sys^{\mathsf{NS,id}}$*, i.e., they hold at all times in all runs of* $\{\mathsf{M}_u^{\mathsf{NS}} \mid u \in \mathcal{H}\} \cup \{\mathsf{TH}_{\mathcal{H}}\}$ *for all* $\mathcal{H} \subseteq \{1, \ldots, n\}$. □

The proof is postponed to Section 10.4.

10.2 Entity Authentication Proof

To increase readability, we partition the proof into several steps with explanations in between. Assume that $u, v \in \mathcal{H}$ and that $\mathsf{M}_v^{\mathsf{NS}}$ outputs (ok, u) to its user, i.e., a protocol between u and v has terminated successfully. We first show that this implies that $\mathsf{M}_v^{\mathsf{NS}}$ has received a message corresponding to the third protocol step, i.e., of the form that allows us to apply *correct list owner* to show that it was created by $\mathsf{M}_v^{\mathsf{NS}}$. The following property of $\mathsf{TH}_{\mathcal{H}}$ proven in [1] will be useful in this proof to show that properties proven for one time also hold at another time.

Lemma 2. *In the ideal cryptographic library* $Sys^{\mathsf{cry,id}}$*, the only modifications to existing entries* x *in* D *are assignments to previously undefined attributes* $x.hnd_u$ *(except for counter updates in entries for signature keys, which we do not have to consider here).* □

Proof. (Ideal part of Theorem 2) Assume that $\mathsf{M}_v^{\mathsf{NS}}$ outputs (ok, u) at $\mathsf{EA\_out}_v!$ for $u, v \in \mathcal{H}$ at time t_4. By definition of Algorithms 1 and 2, this can only happen if there was an input $(u, v, \mathsf{i}, m_v^{3\ \mathsf{hnd}})$ at $\mathsf{out}_v?$ at a time $t_3 < t_4$. Here and in the sequel we use the notation of Algorithm 2, but we distinguish the variables from its different executions by a superscript indicating the number of the (claimed) received protocol message, here ^{3}, and give handles an additional subscript for their owner, here v.

The execution of Algorithm 2 for this input must have given $l_v^{3\ \mathsf{hnd}} \neq \downarrow$ in Step 2.2, since it would otherwise abort by Convention 1 without creating an output. Let $l^{3\mathsf{ind}} := D[hnd_v = l_v^{3\ \mathsf{hnd}}].ind$. The algorithm further implies $D[l^{3\mathsf{ind}}].type = \mathsf{list}$. Let $x_i^{3\mathsf{ind}} := D[l^{3\mathsf{ind}}].arg[i]$ for $i = 1, 2$ at the time of Step 2.3. By definition of $\mathsf{list\_proj}$ and since the condition of Step 2.25 is true immediately after Step 2.3, we have

$$x_{1,v}^{3\ \mathsf{hnd}} = D[x_1^{3\mathsf{ind}}].hnd_v \text{ at time } t_4 \tag{4}$$

and

$$x_{1,v}^{3\ \mathsf{hnd}} \in Nonce_{v,u} \wedge x_2^{3\mathsf{ind}} = \downarrow \text{ at time } t_4, \tag{5}$$

since $x_{2,v}^{3\ \mathsf{hnd}} = \downarrow$ after Step 2.3 implies $x_2^{3\mathsf{ind}} = \downarrow$.

This first part of the proof shows that $\mathsf{M}_v^{\mathsf{NS}}$ has received a list corresponding to a third protocol message. Now we apply *correct list owner* to the list entry $D[l^{3^{\mathsf{ind}}}]$ to show that this entry was created by $\mathsf{M}_u^{\mathsf{NS}}$. Then we show that $\mathsf{M}_u^{\mathsf{NS}}$ only generates such an entry if it has received a second protocol message. To show that this message contains a nonce from v, as needed for the next application of *correct list owner*, we exploit the fact that v accepts the same value as its nonce in the third message, which we know from the first part of the proof.

Proof (cont'd with 3rd message). Equations (4) and (5) are the preconditions for Part c) of *correct list owner*. Hence the entry $D[l^{3^{\mathsf{ind}}}]$ was created by $\mathsf{M}_u^{\mathsf{NS}}$ in Step 2.21.

This algorithm execution must have started with an input $(w, u, \mathsf{i}, m_u^{2^{\mathsf{hnd}}})$ at out_u? at a time $t_2 < t_3$ with $w \neq u$. As above, we conclude $l_u^{2^{\mathsf{hnd}}} \neq \downarrow$ in Step 2.2, set $l^{2^{\mathsf{ind}}} := D[hnd_u = l_u^{2^{\mathsf{hnd}}}].ind$, and obtain $D[l^{2^{\mathsf{ind}}}].type = \mathsf{list}$. Let $x_i^{2^{\mathsf{ind}}} := D[l^{2^{\mathsf{ind}}}].arg[i]$ for $i = 1, 2, 3$ at the time of Step 2.3. As the condition of Step 2.16 is true immediately afterwards, we obtain $x_{i,u}^{2^{\mathsf{hnd}}} \neq \downarrow$ for $i \in \{1, 2, 3\}$. The definition of $\mathsf{list\_proj}$ and Lemma 2 imply

$$x_{i,u}^{2^{\mathsf{hnd}}} = D[x_i^{2^{\mathsf{ind}}}].hnd_u \text{ for } i \in \{1, 2, 3\} \text{ at time } t_4. \tag{6}$$

Step 2.18 ensures $x_3^2 = w$ and $x_{1,u}^{2^{\mathsf{hnd}}} \in Nonce_{u,w}$. Thus *correct nonce owner* implies

$$D[x_1^{2^{\mathsf{ind}}}].type = \mathsf{nonce}. \tag{7}$$

Now we exploit that $\mathsf{M}_u^{\mathsf{NS}}$ creates the entry $D[l^{3^{\mathsf{ind}}}]$ in Step 2.21 with the input $\mathsf{list}(x_{2,u}^{2^{\mathsf{hnd}}})$. With the definitions of list and $\mathsf{list\_proj}$ this implies $x_2^{2^{\mathsf{ind}}} = x_1^{3^{\mathsf{ind}}}$. Thus Equations (4) and (5) imply

$$D[x_2^{2^{\mathsf{ind}}}].hnd_v \in Nonce_{v,u} \text{ at time } t_4. \tag{8}$$

We have now shown that $\mathsf{M}_u^{\mathsf{NS}}$ has received a list corresponding to the second protocol message. We apply *correct list owner* to show that $\mathsf{M}_v^{\mathsf{NS}}$ created this list, and again we can show that this can only happen if $\mathsf{M}_v^{\mathsf{NS}}$ received a suitable first protocol message. Further, the next part of the proof shows that $w = v$ and thus $\mathsf{M}_u^{\mathsf{NS}}$ got the second protocol message from $\mathsf{M}_v^{\mathsf{NS}}$, which remained open in the previous proof part.

Proof (cont'd with 2nd message). Equations (6) to (8) are the preconditions for Part b) of *correct list owner*. Thus the entry $D[l^{2^{\mathsf{ind}}}]$ was created by $\mathsf{M}_v^{\mathsf{NS}}$ in Step 2.12. The construction of this entry in Steps 2.11 and 2.12 implies $x_3^2 = v$ and hence $w = v$ (using the definitions of store and $\mathsf{retrieve}$, and list and $\mathsf{list\_proj}$). With the results from before Equation (7) and Lemma 2 we therefore obtain

$$x_3^2 = v \wedge x_{1,u}^{2^{\mathsf{hnd}}} \in Nonce_{u,v} \text{ at time } t_4. \tag{9}$$

The algorithm execution where $\mathsf{M}_v^{\mathsf{NS}}$ creates the entry $D[l^{2^{\mathsf{ind}}}]$ must have started with an input $(w', v, \mathsf{i}, m_v^{1^{\mathsf{hnd}}})$ at out_v? at a time $t_1 < t_2$ with $w' \neq v$. As above, we conclude $l_v^{1^{\mathsf{hnd}}} \neq \downarrow$ in Step 2.2, set $l^{1^{\mathsf{ind}}} := D[hnd_v = l_v^{1^{\mathsf{hnd}}}].ind$, and obtain

$D[l^{1\,\mathsf{ind}}].type = \mathsf{list}$. Let $x_i^{1\,\mathsf{ind}} := D[l^{1\,\mathsf{ind}}].arg[i]$ for $i = 1, 2, 3$ at the time of Step 2.3. As the condition of Step 2.4 is true, we obtain $x_{i,v}^{1\,\mathsf{hnd}} \neq \downarrow$ for $i \in \{1, 2\}$. Then the definition of list_proj and Lemma 2 yield

$$x_{i,v}^{1\,\mathsf{hnd}} = D[x_i^{1\,\mathsf{ind}}].hnd_v \text{ for } i \in \{1, 2\} \text{ at time } t_4. \tag{10}$$

When $\mathsf{M}_v^{\mathsf{NS}}$ creates the entry $D[l^{2\,\mathsf{ind}}]$ in Step 2.12, its input is $\mathsf{list}(x_{1,v}^{1\,\mathsf{hnd}}, n_v^{\mathsf{hnd}}, v^{\mathsf{hnd}})$. This implies $x_1^{1\,\mathsf{ind}} = x_1^{2\,\mathsf{ind}}$ (as above). Thus Equations (6) and (9) imply

$$D[x_1^{1\,\mathsf{ind}}].hnd_u \in Nonce_{u,v} \text{ at time } t_4. \tag{11}$$

The test in Step 2.6 ensures that $x_2^1 = w' \neq \downarrow$. This implies $D[x_2^{1\,\mathsf{ind}}].type = \mathsf{data}$ by the definition of retrieve, and therefore with Lemma 2,

$$D[x_2^{1\,\mathsf{ind}}].type = \mathsf{data} \text{ at time } t_4. \tag{12}$$

We finally apply *correct list owner* again to show that $\mathsf{M}_u^{\mathsf{NS}}$ has generated this list corresponding to a first protocol message. We then show that this message must have been intended for user v, and thus user u has indeed started a protocol with user v.

Proof. (cont'd with 1st message) Equations (10) to (12) are the preconditions for Part a) of *correct list owner*. Thus the entry $D[l^{1\,\mathsf{ind}}]$ was created by $\mathsf{M}_u^{\mathsf{NS}}$ in Step 1.4. The construction of this entry in Steps 1.3 and 1.4 implies $x_2^1 = u$ and hence $w' = u$.

The execution of Algorithm 1 must have started with an input $(\mathsf{new\_prot}, w'')$ at $\mathsf{EA\_in}_u?$ at a time $t_0 < t_1$. We have to show $w'' = v$. When $\mathsf{M}_u^{\mathsf{NS}}$ creates the entry $D[l^{1\,\mathsf{ind}}]$ in Step 1.4, its input is $\mathsf{list}(n_u^{\mathsf{hnd}}, u^{\mathsf{hnd}})$ with $n_u^{\mathsf{hnd}} \neq \downarrow$. Hence the definition of list_proj implies $D[x_1^{1\,\mathsf{ind}}].hnd_u = n_u^{\mathsf{hnd}} \in Nonce_{u,w''}$. With Equation (11) and *unique nonce use* we conclude $w'' = v$.

In a nutshell, we have shown that for all times t_4 where $\mathsf{M}_v^{\mathsf{NS}}$ outputs (ok, u) at $\mathsf{EA\_out}_v!$, there exists a time $t_0 < t_4$ such that $\mathsf{M}_u^{\mathsf{NS}}$ receives an input $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$ at time t_0. This proves Theorem 2.

10.3 Command Evaluation by the Ideal Cryptographic Library

This section contains the definition of the cryptographic commands used for modeling the Needham-Schroeder-Lowe protocol, and the local adversary commands that model the extended capabilities of the adversary as far as needed to prove the invariants. Recall that we deal with top levels of Dolev-Yao-style terms, and that commands typically create a new term with its index, type, arguments, handles, and length functions, or parse an existing term. We present the full definitions of the commands, but the reader can ignore the length functions, which have names x_len. Note that we already defined the commands for generating a nonce and for public-key encryption in Section 3.2, hence we do not repeat them here.

Each input c at a port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$ should be a list $(cmd, x_1, \ldots, x_j)$ with cmd from a fixed list of commands and certain parameter domains. We usually

write it $y \leftarrow cmd(x_1, \ldots, x_j)$ with a variable y designating the result that $\mathsf{TH}_{\mathcal{H}}$ returns at $\mathsf{out}_u!$. The algorithm $i^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(i)$ (with side effect) denotes that $\mathsf{TH}_{\mathcal{H}}$ determines a handle i^{hnd} for user u to an entry $D[i]$: If $i^{\mathsf{hnd}} := D[i].hnd_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].hnd_u := curhnd_u\mathsf{++}$. On non-handles, it is the identity function. The function $\mathsf{ind2hnd}_u^*$ applies $\mathsf{ind2hnd}_u$ to each element of a list.

In the following definitions, we assume that a cryptographic command is input at port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$. First, we describe the commands for storing and retrieving data via handles.

- *Storing:* $m^{\mathsf{hnd}} \leftarrow \mathsf{store}(m)$, for $m \in \{0,1\}^{\mathsf{max\_len}(k)}$.
  If $i := D[type = \mathsf{data} \wedge arg = (m)].ind \neq \downarrow$ then return $m^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(i)$. Otherwise if $\mathsf{data\_len}^*(\mathsf{len}(m)) > \mathsf{max\_len}(k)$ return $\downarrow$. Else set $m^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

$$D :\Leftarrow (ind := size\mathsf{++}, type := \mathsf{data}, arg := (m), \\ hnd_u := m^{\mathsf{hnd}}, len := \mathsf{data\_len}^*(\mathsf{len}(m))).$$

- *Retrieval:* $m \leftarrow \mathsf{retrieve}(m^{\mathsf{hnd}})$.
  $m := D[hnd_u = m^{\mathsf{hnd}} \wedge type = \mathsf{data}].arg[1]$.

Next we describe list creation and projection. Lists cannot include secret keys of the public-key systems (entries of type ske, sks) because no information about those must be given away.

- *Generate a list:* $l^{\mathsf{hnd}} \leftarrow \mathsf{list}(x_1^{\mathsf{hnd}}, \ldots, x_j^{\mathsf{hnd}})$, for $0 \leq j \leq \mathsf{max\_len}(k)$.
  Let $x_i := D[hnd_u = x_i^{\mathsf{hnd}}].ind$ for $i = 1, \ldots, j$. If any $D[x_i].type \in \{\mathsf{sks}, \mathsf{ske}\}$, set $l^{\mathsf{hnd}} := \downarrow$.If $l := D[type = \mathsf{list} \wedge arg = (x_1, \ldots, x_j)].ind \neq \downarrow$, then return $l^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(l)$. Otherwise, set $length := \mathsf{list\_len}^*(D[x_1].len, \ldots, D[x_j].len)$ and return $\downarrow$ if $length > \mathsf{max\_len}(k)$. Else set $l^{\mathsf{hnd}} := curhnd_u\mathsf{++}$ and

$$D :\Leftarrow (ind := size\mathsf{++}, type := \mathsf{list}, arg := (x_1, \ldots, \\ x_j), hnd_u := l^{\mathsf{hnd}}, len := length).$$

- *i-th projection:* $x^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$, for $1 \leq i \leq \mathsf{max\_len}(k)$.
  If $D[hnd_u = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].arg = (x_1, \ldots, x_j)$ with $j \geq i$, then $x^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(x_i)$, otherwise $x^{\mathsf{hnd}} := \downarrow$.

Further, we used a command for decrypting a list.

- *Decryption:* $l^{\mathsf{hnd}} \leftarrow \mathsf{decrypt}(sk^{\mathsf{hnd}}, c^{\mathsf{hnd}})$.
  Let $sk := D[hnd_u = sk^{\mathsf{hnd}} \wedge type = \mathsf{ske}].ind$ and $c := D[hnd_u = c^{\mathsf{hnd}} \wedge type = \mathsf{enc}].ind$. Return $\downarrow$ if $c = \downarrow$ or $sk = \downarrow$ or $pk := D[c].arg[1] \neq sk + 1$ or $l := D[c].arg[2] = \downarrow$. Else return $l^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(l)$.

From the set of local adversary commands, which capture additional commands for the adversary at port $\mathsf{in}_\mathsf{a}?$, we only describe the command adv_parse. It allows the adversary to retrieve all information that we do not explicitly require to be hidden.

This command returns the type and usually all the abstract arguments of a value (with indices replaced by handles), except in the case of ciphertexts. About the remaining local adversary commands, we only need to know that they do not output handles to already existing entries of type list or nonce.

– *Parameter retrieval:* $(type, arg) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$.
Let $m := D[hnd_{\mathsf{a}} = m^{\mathsf{hnd}}].ind$ and $type := D[m].type$. In most cases, set $arg := \mathsf{ind2hnd}^*_{\mathsf{a}}(D[m].arg)$. (Recall that this only transforms arguments in $\mathcal{INDS}$.) The only exception is for $type = \mathsf{enc}$ and $D[m].arg$ of the form (pk, l) (a valid ciphertext) and $D[pk - 1].hnd_{\mathsf{a}} = \downarrow$ (the adversary does not know the secret key); then $arg := (\mathsf{ind2hnd}_{\mathsf{a}}(pk), D[l].len)$.

We finally describe the command that allows an adversary to send messages on insecure channels. In the command, the adversary sends list l to v, pretending to be u.

– $\mathsf{adv\_send\_i}(u, v, l^{\mathsf{hnd}})$, for $u \in \{1, \ldots, n\}$ and $v \in \mathcal{H}$ at port $\mathsf{in}_{\mathsf{a}}?$.
Let $l^{\mathsf{ind}} := D[hnd_{\mathsf{a}} = l^{\mathsf{hnd}} \wedge type = \mathsf{list}].ind$. If $l^{\mathsf{ind}} \neq \downarrow$, output $(u, v, \mathsf{i}, \mathsf{ind2hnd}_v(l^{\mathsf{ind}}))$ at $\mathsf{out}_v!$.

10.4 Proof of the Invariants

We start with the proof of *correct nonce owner*.

*Proof (*Correct nonce owner*).* Let $x^{\mathsf{hnd}} \in Nonce_{u,v}$ for $u \in \mathcal{H}$ and $v \in \{1, \ldots, n\}$. By construction, x^{hnd} has been added to $Nonce_{u,v}$ by $\mathsf{M}^{\mathsf{NS}}_u$ in Step 1.2 or Step 2.10. In both cases, x^{hnd} has been generated by the command $\mathsf{gen\_nonce}()$ at some time t, input at port $\mathsf{in}_u?$ of $\mathsf{TH}_{\mathcal{H}}$. Convention 1 implies $x^{\mathsf{hnd}} \neq \downarrow$, as $\mathsf{M}^{\mathsf{NS}}_u$ would abort otherwise and not add x^{hnd} to the set $Nonce_{u,v}$. The definition of $\mathsf{gen\_nonce}$ then implies $D[hnd_u = x^{\mathsf{hnd}}] \neq \downarrow$ and $D[hnd_u = x^{\mathsf{hnd}}].type = \mathsf{nonce}$ at time t. Because of Lemma 2 this also holds at all later times $t' > t$, which finishes the proof.

The following proof of *unique nonce use* is quite similar.

*Proof (*Unique Nonce Use*).* Assume for contradiction that both $D[j].hnd_u \in Nonce_{u,w}$ and $D[j].hnd_v \in Nonce_{v,w'}$ at some time t. Without loss of generality, let t be the first such time and let $D[j].hnd_v \notin Nonce_{v,w'}$ at time $t - 1$. By construction, $D[j].hnd_v$ is thus added to $Nonce_{v,w'}$ at time t by Step 1.2 or Step 2.10. In both cases, $D[j].hnd_v$ has been generated by the command $\mathsf{gen\_nonce}()$ at time $t - 1$. The definition of $\mathsf{gen\_nonce}$ implies that $D[j]$ is a new entry and $D[j].hnd_v$ its only handle at time $t - 1$, and thus also at time t. With *correct nonce owner* this implies $u = v$. Further, $Nonce_{v,w'}$ is the only set into which the new handle $D[j].hnd_v$ is put at times $t - 1$ and t. Thus also $w = w'$. This is a contradiction.

In the following, we prove *correct list owner*, *nonce secrecy*, and *nonce-list secrecy* by induction. Hence we assume that all three invariants hold at a particular time t in a run of the system, and show that they still hold at time $t + 1$.

*Proof (*Correct list owner*).* Let $u, v \in \mathcal{H}$, $j \leq size$ with $D[j].type = \mathsf{list}$. Let $x_i^{\mathsf{ind}} := D[j].arg[i]$ and $x_{i,u}^{\mathsf{hnd}} := D[x_i^{\mathsf{ind}}].hnd_u$ for $i = 1, 2$ and assume that $x_{i,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ for $i = 1$ or $i = 2$ at time $t + 1$.

The only possibilities to violate the invariant *correct list owner* are that (1) the entry $D[j]$ is created at time $t+1$ or that (2) the handle $D[j].hnd_u$ is created at time $t+1$ for an entry $D[j]$ that already exists at time t or that (3) the handle $x_{i,u}^{\mathsf{hnd}}$ is added to $Nonce_{u,v}$ at time $t + 1$. In all other cases the invariant holds by the induction hypothesis and Lemma 2.

We start with the third case. Assume that $x_{i,u}^{\mathsf{hnd}}$ is added to $Nonce_{u,v}$ at time $t + 1$. By construction, this only happens in a transition of $\mathsf{M}_u^{\mathsf{NS}}$ in Step 1.2 and Step 2.10. However, here the entry $D[x_i^{\mathsf{ind}}]$ has been generated by the command $\mathsf{gen\_nonce}$ input at $\mathsf{in}_u?$ at time t, hence x_i^{ind} cannot be contained as an argument of an entry $D[j]$ at time t. Formally, this corresponds to the fact that D is *well-formed*, i.e., index arguments of an entry are always smaller than the index of the entry itself; this has been shown in [1]. Since a transition of $\mathsf{M}_u^{\mathsf{NS}}$ does not modify entries in $\mathsf{TH}_{\mathcal{H}}$, this also holds at time $t + 1$.

For proving the remaining two cases, assume that $D[j].hnd_u$ is created at time $t+1$ for an already existing entry $D[j]$ or that $D[j]$ is generated at time $t + 1$. Because both can only happen in a transition of $\mathsf{TH}_{\mathcal{H}}$, this implies $x_{i,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ already at time t, since transitions of $\mathsf{TH}_{\mathcal{H}}$ cannot modify the set $Nonce_{u,v}$. Because of $u, v \in \mathcal{H}$, *nonce secrecy* implies $D[x_i^{\mathsf{ind}}].hnd_w \neq \downarrow$ only if $w \in \{u, v\}$. Lists can only be constructed by the basic command list, which requires handles to all its elements. More precisely, if $w \in \mathcal{H} \cup \{\mathsf{a}\}$ creates an entry $D[j']$ with $D[j'].type = \mathsf{list}$ and $(x'_1, \ldots, x'_k) := D[j].arg$ at time $t + 1$ then $D[x'_i].hnd_w \neq \downarrow$ for $i = 1, \ldots, k$ already at time t. Applied to the entry $D[j]$, this implies that either u or v have created the entry $D[j]$.

We now only have to show that the entry $D[j]$ has been created by u in the claimed steps. This can easily be seen by inspection of Algorithms 1 and 2. We only show it in detail for the first part of the invariant; it can be proven similarly for the remaining two parts.

Let $x_{1,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ and $D[x_2^{\mathsf{ind}}].type = \mathsf{data}$. By inspection of Algorithms 1 and 2 and because $D[j].type = \mathsf{list}$, we see that the entry $D[j]$ must have been created by either $\mathsf{M}_u^{\mathsf{NS}}$ or $\mathsf{M}_v^{\mathsf{NS}}$ in Step 1.4. (The remaining list generation commands either only have one element, which implies $x_2^{\mathsf{ind}} = \downarrow$ and hence $D[x_2^{\mathsf{ind}}].type \neq \mathsf{data}$, or we have $D[x_2^{\mathsf{ind}}].type = \mathsf{nonce}$ by construction.) Now assume for contradiction that the entry $D[j]$ has been generated by $\mathsf{M}_v^{\mathsf{NS}}$. This implies that also the entry $D[x_1^{\mathsf{ind}}]$ has been newly generated by the command $\mathsf{gen\_nonce}$ input at $\mathsf{in}_v?$. However, only $\mathsf{M}_u^{\mathsf{NS}}$ can add a handle to the set $Nonce_{u,v}$ (it is the local state of $\mathsf{M}_u^{\mathsf{NS}}$), but every nonce that $\mathsf{M}_u^{\mathsf{NS}}$ adds to the set $Nonce_{u,v}$ is newly generated by the command $\mathsf{gen\_nonce}$ input by $\mathsf{M}_u^{\mathsf{NS}}$ by construction. This implies $x_{1,u}^{\mathsf{hnd}} \notin Nonce_{u,v}$ at all times, which yields a contradiction to $x_{1,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ at time $t + 1$. Hence $D[j]$ has been created by user u.

*Proof (*Nonce secrecy*).* Let $u, v \in \mathcal{H}$, $j \leq size$ with $D[j].hnd_u \in Nonce_{u,v}$, and $w \in (\mathcal{H} \cup \{\mathsf{a}\}) \setminus \{u, v\}$ be given. Because of *correct nonce owner*, we know that $D[j].type = \mathsf{nonce}$. The invariant could only be affected if (1) the handle $D[j].hnd_u$ is put into the set $Nonce_{u,v}$ at time $t+1$ or (2) if a handle for w is added to the entry $D[j]$ at time $t + 1$.

For proving the first case, note that the set $Nonce_{u,v}$ is only extended by a handle n_u^{hnd} by $\mathsf{M}_u^{\mathsf{NS}}$ in Steps 1.2 and 2.10. In both cases, n_u^{hnd} has been generated by $\mathsf{TH}_{\mathcal{H}}$ at time t since the command gen_nonce was input at $\mathsf{in}_u?$ at time t. The definition of gen_nonce immediately implies that $D[j].hnd_w = \downarrow$ at time t if $w \neq u$. Moreover, this also holds at time $t+1$ since a transition of $\mathsf{M}_u^{\mathsf{NS}}$ does not modify handles in $\mathsf{TH}_{\mathcal{H}}$, which finishes the claim for this case.

For proving the second case, we only have to consider those commands that add handles for w to entries of type nonce. These are only the commands list_proj or adv_parse input at $\mathsf{in}_w?$, where adv_parse has to be applied to an entry of type list, since only entries of type list can have arguments which are indices to nonce entries. More precisely, if one of the commands violated the invariant there would exist an entry $D[i]$ at time t such that $D[i].type = \mathsf{list}$, $D[i].hnd_w \neq \downarrow$ and $j \in (x_1^{\mathsf{ind}}, \ldots, x_m^{\mathsf{ind}}) := D[i].arg$. However, both commands do not modify the set $Nonce_{u,v}$, hence we have $D[j].hnd_u \in Nonce_{u,v}$ already at time t. Now *nonce secrecy* yields $D[j].hnd_w = \downarrow$ at time t and hence also at all times $< t$ because of Lemma 2. This implies that the entry $D[i]$ must have been created by either u or v, since generating a list presupposes handles for all elements (cf. the previous proof). Assume without loss of generality that $D[i]$ has been generated by u. By inspection of Algorithms 1 and 2, this immediately implies $j \in (x_1^{\mathsf{ind}}, x_2^{\mathsf{ind}})$, since handles to nonces only occur as first or second element in a list generation by u. Because of $j \in D[i].arg[1,2]$ and $D[j].hnd_u \in Nonce_{u,v}$ at time t, *nonce-list secrecy* for the entry $D[i]$ implies that $D[i].hnd_w = \downarrow$ at time t. This yields a contradiction.

*Proof (*Nonce-list secrecy*).* Let $u, v \in \mathcal{H}$, $j \leq size$ with $D[j].type = \mathsf{list}$. Let $x_i^{\mathsf{ind}} := D[j].arg[i]$ and $x_{i,u}^{\mathsf{hnd}} := D[x_i^{\mathsf{ind}}].hnd_u$ for $i = 1, 2$, and $w \in (\mathcal{H} \cup \{\mathsf{a}\}) \setminus \{u, v\}$. Let $x_{i,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ for $i = 1$ or $i = 2$.

We first show that the invariant cannot be violated by adding the handle $x_{i,u}^{\mathsf{hnd}}$ to $Nonce_{u,v}$ at time $t+1$. This can only happen in a transition of $\mathsf{M}_u^{\mathsf{NS}}$ in Step 1.2 or 2.10. As shown in the proof of *correct list owner*, the entry $D[x_i^{\mathsf{ind}}]$ has been generated by $\mathsf{TH}_{\mathcal{H}}$ at time t. Since D is well-formed, this implies that $x_i^{\mathsf{ind}} \notin D[j].arg$ for all entries $D[j]$ that already exist at time t. This also holds for all entries at time $t+1$, since the transition of $\mathsf{M}_u^{\mathsf{NS}}$ does not modify entries of $\mathsf{TH}_{\mathcal{H}}$. This yields a contradiction to $x_i^{\mathsf{ind}} = D[j].arg[i]$. Hence we now know that $x_{i,u}^{\mathsf{hnd}} \in Nonce_{u,v}$ already holds at time t.

Part a) of the invariant can only be affected if a handle for w is added to an entry $D[j]$ that already exists at time t. (Creation of $D[j]$ at time t with a handle for w is impossible as above because that presupposes handles to all arguments, in contradiction to *nonce secrecy*.) The only commands that add new handles for w to existing entries of type list are list_proj, decrypt, adv_parse, send_i, and adv_send_i applied to an entry $D[k]$ with $j \in D[k].arg$. *Nonce-list secrecy* for the entry $D[j]$ at time t then yields $D[k].type = \mathsf{enc}$. Thus the commands list_proj, send_i, and adv_send_i do not have to be considered any further. Moreover, *nonce-list secrecy* also yields $D[k].arg[1] \in \{pke_u, pke_v\}$. The secret keys of u and v are not known to $w \notin \{u, v\}$, formally $D[hnd_w = ske_u^{\mathsf{hnd}}] = D[hnd_w = ske_v^{\mathsf{hnd}}] = \downarrow$; this corresponds to the invariant *key secrecy* of [1]. Hence the command decrypt does not violate the invariant. Finally, the command adv_parse applied to an entry of type enc with unknown secret key also does not give a handle to the cleartext list, i.e., to $D[k].arg[2]$, but only outputs its length.

Part b) of the invariant can only be affected if the list entry $D[j]$ is created at time $t+1$. (By well-formedness, the argument entry $D[x_{i+1}^{\mathsf{ind}}]$ cannot be created after $D[j]$.) As in Part a), it can only be created by a party $w \in \{u, v\}$ because other parties have no handle to the nonce argument. Inspection of Algorithms 1 and 2 shows that this can only happen in Steps 1.4 and 2.12, because all other commands list have only one argument, while our preconditions imply $x_2^{\mathsf{ind}} \neq \downarrow$.

- If the creation is in Step 1.4, the preceding Step 1.2 implies $D[x_1^{\mathsf{ind}}].hnd_w \in Nonce_{w,w'}$ for some w' and Step 1.3 implies $D[x_2^{\mathsf{ind}}].type = \mathsf{data}$. Thus the preconditions of Part b) of the invariant can only hold for $i = 1$, and thus $D[x_1^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$. Now *unique nonce use* implies $u = w$. Thus Steps 1.3 and 1.4 yield $D[x_2^{\mathsf{ind}}].arg = (u)$.
- If the creation is in Step 2.12, the preceding steps 2.10 and 2.11 imply that the preconditions of Part b) of the invariant can only hold for $i = 2$. Then the precondition, Step 2.10, and *unique nonce use* imply $u = w$. Finally, Steps 2.11 and 2.12 yield $D[x_3^{\mathsf{ind}}].arg = (u)$.

Part c) of the invariant can only be violated if a new entry $D[k]$ is created at time $t+1$ with $j \in D[k].arg$ (by Lemma 2 and well-formedness). As $D[j]$ already exists at time t, *nonce-list secrecy* for $D[j]$ implies $D[j].hnd_w = \downarrow$ for $w \notin \{u, v\}$ at time t. We can easily see by inspection of the commands that the new entry $D[k]$ must have been created by one of the commands list and encrypt (or by sign, which creates a signature), since entries newly created by other commands cannot have arguments that are indices of entries of type list. Since all these commands entered at a port $\mathsf{in}_z?$ presuppose $D[j].hnd_z \neq \downarrow$, the entry $D[k]$ is created by $w \in \{u, v\}$ at time $t+1$. However, the only steps that can create an entry $D[k]$ with $j \in D[k].arg$ (with the properties demanded for the entry $D[j]$) are Steps 1.5, 2.13, and 2.22. In all these cases, we have $D[k].type = \mathsf{enc}$. Further, we have $D[k].arg[1] = pke_{w'}$ where w' denotes w's current believed partner. We have to show that $w' \in \{u, v\}$.

- Case 1: $D[k]$ is created in Step 1.5. By inspection of Algorithm 1, we see that the precondition of this proof can only be fulfilled for $i = 1$. Then $D[x_1^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$ and $D[x_1^{\mathsf{ind}}].hnd_w \in Nonce_{w,w'}$ and *unique nonce use* imply $w' = v$.
- Case 2: $D[k]$ is created in Step 2.13, and $i = 2$. Then $D[x_2^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$ and $D[x_2^{\mathsf{ind}}].hnd_w \in Nonce_{w,w'}$ and *unique nonce use* imply $w' = v$.
- Case 3: $D[k]$ is created in Step 2.13, and $i = 1$. This execution of Algorithm 2 must give $l^{\mathsf{hnd}} \neq \downarrow$ in Step 2.2, since it would otherwise abort by Convention 1. Let $l^{\mathsf{ind}} := D[hnd_w = l^{\mathsf{hnd}}].ind$. The algorithm further implies $D[l^{\mathsf{ind}}].type = \mathsf{list}$. Let $x_i^{0\,\mathsf{ind}} := D[l^{\mathsf{ind}}].arg[i]$ for $i = 1, 2, 3$ at the time of Step 2.3, and let $x_{i,w}^{0\,\mathsf{hnd}}$ be the handles obtained in Step 2.3. As the algorithm does not abort in Steps 2.5 and 2.7, we have $D[x_2^{0\,\mathsf{ind}}].type = \mathsf{data}$ and $D[x_2^{0\,\mathsf{ind}}].arg = (w')$.
  Further, the reuse of $x_{1,w}^{0\,\mathsf{hnd}}$ in Step 2.12 implies $x_1^{0\,\mathsf{ind}} = x_1^{\mathsf{ind}}$. Together with the precondition $D[x_1^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$, the entry $D[l^{\mathsf{ind}}]$ therefore fulfills the conditions of Part b) of *nonce-list secrecy* with $i = 1$. This implies $D[x_2^{0\,\mathsf{ind}}].arg = (u)$, and thus $w' = u$.
- Case 4: $D[k]$ is created in Step 2.22. With Step 2.21, this implies $x_2^{\mathsf{ind}} = \downarrow$ and thus $i = 1$. As in Case 3, this execution of Algorithm 2 must give $l^{\mathsf{hnd}} \neq \downarrow$ in

Step 2.2, we set $l^{\mathsf{ind}} := D[hnd_w = l^{\mathsf{hnd}}].ind$, and we have $D[l^{\mathsf{ind}}].type = \mathsf{list}$. Let $x_i^{0\,\mathsf{ind}} := D[l^{\mathsf{ind}}].arg[i]$ for $i = 1, 2, 3$ at the time of Step 2.3, and let $x_{i,w}^{0\,\mathsf{hnd}}$ be the handles obtained in Step 2.3. As the algorithm does not abort in Steps 2.17 and 2.19, we have $D[x_3^{0\,\mathsf{ind}}].type = \mathsf{data}$ and $D[x_3^{0\,\mathsf{ind}}].arg = (w')$.
Further, the reuse of $x_{2,w}^{0\,\mathsf{hnd}}$ in Step 2.21 implies $x_2^{0\,\mathsf{ind}} = x_1^{\mathsf{ind}}$. Together with the precondition $D[x_1^{\mathsf{ind}}].hnd_u \in Nonce_{u,v}$, the entry $D[l^{\mathsf{ind}}]$ therefore fulfills the condition of Part b) of *nonce-list secrecy* with $i = 2$. This implies $D[x_3^{0\,\mathsf{ind}}].arg = (u)$, and thus $w' = u$.

Hence in all cases we obtained $w' = u$, i.e., the list containing the nonce was indeed encrypted with the key of an honest participant.

11 Conclusion

We have shown that an ideal cryptographic library, which constitutes a slightly extended Dolev-Yao model, is sound with respect to the commonly accepted cryptographic definitions under arbitrary active attacks and in arbitrary protocol environments. The abstraction is deterministic and does not contain any cryptographic objects, hence it is abstract in the sense needed for theorem provers. Sound means that we can implement the abstraction securely in the cryptographic sense, so that properties proved for the abstraction carry over to the implementation without any further work. We provided one possible implementation whose security is based on provably secure cryptographic systems. We already showed that the library can be extended in a modular way by adding symmetric authentication [24] and symmetric encryption [25].

This soundness of the cryptographic library now allows for a meaningful analysis of protocol properties on the abstract level. We demonstrated this with a proof of the well-known Needham-Schroeder-Lowe public-key protocol. Further, the abstractness of the library makes such an analysis accessible for formal verification techniques. As many protocols commonly analyzed in the literature can be expressed with our library, this enables the first formal, machine-aided verification of these protocols which is not only meaningful for Dolev-Yao-like abstractions, but whose security guarantees are equivalent to the security of the underlying cryptography. This bridges the up-to-now missing link between cryptography and formal methods for arbitrary attacks.

Acknowledgments

We thank *Anupam Datta, Martin Hirt, Dennis Hofheinz, Paul Karger, Ralf Küsters, John Mitchell, Jörn Müller-Quade, Andre Scedrov, Matthias Schunter, Victor Shoup, Michael Steiner, Rainer Steinwandt, Dominique Unruh* for interesting discussions.

References

1. M. Backes, B. Pfitzmann, M. Waidner, A composable cryptographic library with nested operations (extended abstract), in: Proc. 10th ACM Conference on Computer and Communications Security, 2003, pp. 220–230, full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

2. M. Backes, 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, pp. 1–12, full version in IACR Cryptology ePrint Archive 2003/121, Jun. 2003, http://eprint.iacr.org/.
3. M. Backes, B. Pfitzmann, M. Waidner, Justifying a Dolev-Yao Model under Active Attacks, Vol. 3655 of Lecture Notes in Computer Science, Springer, 2004, pp. 1–42.
4. R. Needham, M. Schroeder, Using encryption for authentication in large networks of computers, Communications of the ACM 12 (21) (1978) 993–999.
5. D. E. Denning, G. M. Sacco, Timestamps in key distribution protocols, Communications of the ACM 24 (8) (1981) 533–536.
6. D. Wagner, B. Schneier, Analysis of the SSL 3.0 protocol, in: Proc. 2nd USENIX Workshop on Electronic Commerce, 1996, pp. 29–40.
7. D. Bleichenbacher, Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS, in: Advances in Cryptology: CRYPTO '98, Vol. 1462 of Lecture Notes in Computer Science, Springer, 1998, pp. 1–12.
8. D. Fisher, Millions of .Net Passport accounts put at risk, eWeek(Flaw detected by Muhammad Faisal Rauf Danka).
URL http://www.eweek.com/article2/0,3959,1066242,00.asp
9. M. Bellare, P. Rogaway, Entity authentication and key distribution, in: Advances in Cryptology: CRYPTO '93, Vol. 773 of Lecture Notes in Computer Science, Springer, 1994, pp. 232–249.
10. B. Pfitzmann, M. Waidner, How to break and repair a "provably secure" untraceable payment system, in: Advances in Cryptology: CRYPTO '91, Vol. 576 of Lecture Notes in Computer Science, Springer, 1992, pp. 338–350.
11. Y. Desmedt, K. Kurosawa, How to break a practical mix and design a new one, in: Advances in Cryptology: EUROCRYPT 2000, Vol. 1807 of Lecture Notes in Computer Science, Springer, 2000, pp. 557–572.
12. D. Dolev, A. C. Yao, On the security of public key protocols, IEEE Transactions on Information Theory 29 (2) (1983) 198–208.
13. B. Pfitzmann, M. Schunter, 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.
14. B. Pfitzmann, M. Waidner, Composition and integrity preservation of secure reactive systems, in: Proc. 7th ACM Conference on Computer and Communications Security, 2000, pp. 245–254, extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz.
15. M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption), Journal of Cryptology 15 (2) (2002) 103–127.
16. M. Abadi, J. Jürjens, Formal eavesdropping and its computational interpretation, in: Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS), 2001, pp. 82–94.
17. B. Pfitzmann, M. Waidner, A model for asynchronous reactive systems and its application to secure message transmission, in: Proc. 22nd IEEE Symposium on Security & Privacy, 2001, pp. 184–200, extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, http://eprint.iacr.org/.
18. R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS), 2001, pp. 136–145, extended version in Cryptology ePrint Archive, Report 2000/67, http://eprint.iacr.org/.

19. M. Backes, C. Jacobi, B. Pfitzmann, Deriving cryptographically sound implementations using composition and formally verified bisimulation, in: Proc. 11th Symposium on Formal Methods Europe (FME 2002), Vol. 2391 of Lecture Notes in Computer Science, Springer, 2002, pp. 310–329.
20. M. Backes, B. Pfitzmann, M. Waidner, A general composition theorem for secure reactive system, in: Proceedings of 1st Theory of Cryptography Conference (TCC), Vol. 2951 of Lecture Notes in Computer Science, Springer, 2004, pp. 336–354.
21. M. Backes, B. Pfitzmann, M. Waidner, Secure asynchronous reactive systems, IACR Cryptology ePrint Archive 2004 (2004) 82.
URL http://eprint.iacr.org/2004/082
22. M. Backes, B. Pfitzmann, M. Waidner, The reactive simulatability framework for asynchronous systems, Information and Computation (2007) 1685–1720.
23. M. Backes, B. Pfitzmann, M. Waidner, A universally composable cryptographic library, IACR Cryptology ePrint Archive 2003/015, http://eprint.iacr.org/ (Jan. 2003).
24. M. Backes, B. Pfitzmann, M. Waidner, Symmetric authentication within a simulatable cryptographic library, in: Proc. 8th European Symposium on Research in Computer Security (ESORICS), Vol. 2808 of Lecture Notes in Computer Science, Springer, 2003, pp. 271–290, extended version in IACR Cryptology ePrint Archive 2003/145, Jul. 2003, http://eprint.iacr.org/.
25. M. Backes, B. Pfitzmann, Symmetric encryption in a simulatable Dolev-Yao style cryptographic library, in: Proc. 17th IEEE Computer Security Foundations Workshop (CSFW), 2004, full version in IACR Cryptology ePrint Archive 2004/059, Feb. 2004, http://eprint.iacr.org/.
26. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, M. Waidner, Cryptographically sound theorem proving, in: Proceedings of 19th IEEE Computer Security Foundations Workshop (CSFW), 2006, pp. 153–166.
27. M. Backes, P. Laud, Computationally sound secrecy proofs by mechanized flow analysis, in: Proceedings of 13th ACM Conference on Computer and Communications Security (CCS), 2006, pp. 370–379.
28. M. Backes, B. Pfitzmann, Limits of the cryptographic realization of Dolev-Yao-style XOR, in: Proceedings of 10th European Symposium on Research in Computer Security (ESORICS), Vol. 3679 of Lecture Notes in Computer Science, Springer, 2005, pp. 178–196.
29. M. Backes, B. Pfitzmann, M. Waidner, Limits of the reactive simulatability/UC of Dolev-Yao models with hashes, in: Proceedings of 11th European Symposium on Research in Computer Security(ESORICS), Vol. 4189 of Lecture Notes in Computer Science, Springer, 2006, pp. 404–423.
30. M. Abadi, P. Rogaway, Reconciling two views of cryptography: The computational soundness of formal encryption, in: Proc. 1st IFIP International Conference on Theoretical Computer Science, Vol. 1872 of Lecture Notes in Computer Science, Springer, 2000, pp. 3–22.
31. R. Canetti, A unified framework for analyzing security of protocols, IACR Cryptology ePrint Archive 2000/067, http://eprint.iacr.org/ (Dec. 2000).
32. P. Lincoln, J. Mitchell, M. Mitchell, A. Scedrov, A probabilistic poly-time framework for protocol analysis, in: Proc. 5th ACM Conference on Computer and Communications Security, 1998, pp. 112–121.
33. G. Lowe, An attack on the Needham-Schroeder public-key authentication protocol, Information Processing Letters 56 (3) (1995) 131–135.
34. 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), Vol. 1055 of Lecture Notes in Computer Science, Springer, 1996, pp. 147–166.

35. P. Syverson, A new look at an old protocol, Operation Systems Review 30 (3) (1996) 1–4.
36. C. Meadows, Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches, in: Proc. 4th European Symposium on Research in Computer Security (ESORICS), Vol. 1146 of Lecture Notes in Computer Science, Springer, 1996, pp. 351–364.
37. S. Schneider, Verifying authentication protocols with CSP, in: Proc. 10th IEEE Computer Security Foundations Workshop (CSFW), 1997, pp. 3–17.
38. F. J. Thayer Fabrega, J. C. Herzog, J. D. Guttman, Strand spaces: Why is a security protocol correct?, in: Proc. 19th IEEE Symposium on Security & Privacy, 1998, pp. 160–171.
39. C. Rackoff, D. R. Simon, Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack, in: Advances in Cryptology: CRYPTO '91, Vol. 576 of Lecture Notes in Computer Science, Springer, 1992, pp. 433–444.
40. M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, Relations among notions of security for public-key encryption schemes, in: Advances in Cryptology: CRYPTO '98, Vol. 1462 of Lecture Notes in Computer Science, Springer, 1998, pp. 26–45.
41. R. Cramer, V. Shoup, Practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, in: Advances in Cryptology: CRYPTO '98, Vol. 1462 of Lecture Notes in Computer Science, Springer, 1998, pp. 13–25.
42. S. Goldwasser, S. Micali, Probabilistic encryption, Journal of Computer and System Sciences 28 (1984) 270–299.
43. S. Goldwasser, S. Micali, R. L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, SIAM Journal on Computing 17 (2) (1988) 281–308.
44. M. Bellare, T. Kohno, C. Namprempre, Authenticated encryption in ssh: Provably fixing the ssh binary packet protocol, in: Proc. 9th ACM Conference on Computer and Communications Security, 2002, pp. 1–11.
45. P. Rogaway, Authenticated-encryption with associated-data, in: Proc. 9th ACM Conference on Computer and Communications Security, 2002, pp. 98–107.
46. R. Kemmerer, C. Meadows, J. Millen, Three systems for cryptographic protocol analysis, Journal of Cryptology 7 (2) (1994) 79–130.
47. J. Mitchell, M. Mitchell, U. Stern, Automated analysis of cryptographic protocols using murϕ, in: Proc. 18th IEEE Symposium on Security & Privacy, 1997, pp. 141–151.
48. Z. Dang, 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/`.
49. B. Dutertre, S. Schneider, Using a PVS embedding of CSP to verify authentication protocols, in: Proc. International Conference on Theorem Proving in Higher Order Logics (TPHOL), Vol. 1275 of Lecture Notes in Computer Science, Springer, 1997, pp. 121–136.
50. L. Paulson, The inductive approach to verifying cryptographic protocols, Journal of Cryptology 6 (1) (1998) 85–128.
51. G. Bella, F. Massacci, L. C. Paulson, The verification of an industrial payment protocol: The SET purchase phase, in: Proc. 9th ACM Conference on Computer and Communications Security, 2002, pp. 12–20.
52. J. D. Guttman, F. J. Thayer Fabrega, L. Zuck, The faithfulness of abstract protocol analysis: Message authentication, in: Proc. 8th ACM Conference on Computer and Communications Security, 2001, pp. 186–195.
53. P. Laud, Semantics and program analysis of computationally secure information flow, in: Proc. 10th European Symposium on Programming (ESOP), 2001, pp. 77–91.
54. S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, in: Advances in Cryptology: CRYPTO '90, Vol. 537 of Lecture Notes in Computer Science, Springer, 1990, pp. 77–93.
55. S. Micali, P. Rogaway, Secure computation, in: Advances in Cryptology: CRYPTO '91, Vol. 576 of Lecture Notes in Computer Science, Springer, 1991, pp. 392–404.

56. D. Beaver, Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority, Journal of Cryptology 4 (2) (1991) 75–122.
57. R. Canetti, Security and composition of multiparty cryptographic protocols, Journal of Cryptology 3 (1) (2000) 143–202.
58. M. Hirt, U. Maurer, Player simulation and general adversary structures in perfect multiparty computation, Journal of Cryptology 13 (1) (2000) 31–60.
59. M. Backes, C. Jacobi, Cryptographically sound and machine-assisted verification of security protocols, in: Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Vol. 2607 of Lecture Notes in Computer Science, Springer, 2003, pp. 675–686.
60. S. Owre, N. Shankar, J. M. Rushby, PVS: A prototype verification system, in: Proc. 11th International Conference on Automated Deduction (CADE), Vol. 607 of Lecture Notes in Computer Science, Springer, 1992, pp. 748–752.
61. M. Backes, B. Pfitzmann, M. Steiner, M. Waidner, Polynomial fairness and liveness, in: Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW), 2002, pp. 160–174.
62. M. Backes, B. Pfitzmann, Computational probabilistic non-interference, in: Proceedings of 7th European Symposium on Research in Computer Security (ESORICS), Vol. 2502 of Lecture Notes in Computer Science, Springer, 2002, pp. 1–23.
63. M. Backes, B. Pfitzmann, Intransitive non-interference for cryptographic purposes, in: Proc. 24th IEEE Symposium on Security & Privacy, 2003, pp. 140–152.
64. M. Backes, B. Pfitzmann, Relating symbolic and cryptographic secrecy, IEEE Transactions on Dependable and Secure Computing (TDSC) 2 (2) (2005) 109–123.
65. M. Backes, Quantifying probabilistic information flow in computational reactive systems, in: Proceedings of 10th European Symposium on Research in Computer Security (ESORICS), Vol. 3679 of Lecture Notes in Computer Science, Springer, 2005, pp. 336–354.
66. J. Herzog, M. Liskov, S. Micali, Plaintext awareness via key registration, in: Advances in Cryptology: CRYPTO 2003, Vol. 2729 of Lecture Notes in Computer Science, Springer, 2003, pp. 548–564.
67. J. Herzog, Computational soundness of formal adversaries, Ph.D. thesis, MIT (2002).
68. R. Canetti, O. Goldreich, S. Halevi, The random oracle methodology, revisited, in: Proc. 30th Annual ACM Symposium on Theory of Computing (STOC), 1998, pp. 209–218.
69. P. Laud, Symmetric encryption in automatic analyses for confidentiality against active adversaries, in: Proc. 25th IEEE Symposium on Security & Privacy, 2004, pp. 71–85.
70. D. Micciancio, B. Warinschi, Soundness of formal encryption in the presence of active adversaries, in: Proc. 1st Theory of Cryptography Conference (TCC), Vol. 2951 of Lecture Notes in Computer Science, Springer, 2004, pp. 133–151.
71. R. Canetti, J. Herzog, Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange), Cryptology ePrint Archive, Report 2004/334, `http://eprint.iacr.org/` (2004).
72. J. Mitchell, M. Mitchell, A. Scedrov, A linguistic characterization of bounded oracle computation and probabilistic polynomial time, in: Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 725–733.
73. J. Mitchell, M. Mitchell, A. Scedrov, V. Teague, A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report), Electronic Notes in Theoretical Computer Science 47 (2001) 1–31.
74. R. Impagliazzo, B. M. Kapron, Logics for reasoning about cryptographic constructions, in: Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS), 2003, pp. 372–381.
75. B. Warinschi, A computational analysis of the Needham-Schroeder-(Lowe) protocol, in: Proc. 16th IEEE Computer Security Foundations Workshop (CSFW), 2003, pp. 248–262.

76. M. Backes, A cryptographically sound dolev-yao style security proof of the Otway-Rees protocol, in: Proceedings of 9th European Symposium on Research in Computer Security (ESORICS), Vol. 3193 of Lecture Notes in Computer Science, Springer, 2004, pp. 89–108.
77. M. Backes, M. Duermuth, A cryptographically sound Dolev-Yao style security proof of an electronic payment system, in: Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW), 2005, pp. 78–93.
78. M. Backes, B. Pfitzmann, On the cryptographic key secrecy of the strengthened Yahalom protocol, in: Proceedings of 21st IFIP International Information Security Conference (SEC), 2006, pp. 233–245.
79. M. Backes, S. Moedersheim, B. Pfitzmann, L. Vigano, Symbolic and cryptographic analysis of the secure WS-ReliableMessaging Scenario, in: Proceedings of Foundations of Software Science and Computational Structures (FOSSACS), Vol. 3921 of Lecture Notes in Computer Science, Springer, 2006, pp. 428–445.
80. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, Cryptographically sound security proofs for basic and public-key kerberos, in: Proceedings of 11th European Symposium on Research in Computer Security(ESORICS), Vol. 4189 of Lecture Notes in Computer Science, Springer, 2006, pp. 362–383, preprint on IACR ePrint 2006/219.
81. R. Cramer, I. Damgård, Secure signature schemes based on interactive protocols, in: Advances in Cryptology: CRYPTO '95, Vol. 963 of Lecture Notes in Computer Science, Springer, 1995, pp. 297–310.
82. R. Cramer, I. Damgård, New generation of secure and practical RSA-based signatures, in: Advances in Cryptology: CRYPTO '96, Vol. 1109 of Lecture Notes in Computer Science, Springer, 1996, pp. 173–185.
83. O. Goldreich, Two remarks concerning the Goldwasser-Micali-Rivest signature scheme, in: Advances in Cryptology: CRYPTO '86, Vol. 263 of Lecture Notes in Computer Science, Springer, 1986, pp. 104–110.
84. R. Cramer, V. Shoup, Signature schemes based on the strong RSA assumption, in: Proc. 6th ACM Conference on Computer and Communications Security, 1999, pp. 46–51.
85. R. Gennaro, S. Halevi, T. Rubin, Secure hash-and-sign signatures without the random oracle, in: Advances in Cryptology: EUROCRYPT '99, Vol. 1592 of Lecture Notes in Computer Science, Springer, 1999, pp. 123–139.
86. R. Anderson, R. Needham, Robustness principles for public key protocols, in: Advances in Cryptology: CRYPTO '95, Vol. 963 of Lecture Notes in Computer Science, Springer, 1995, pp. 236–247.
87. A. C. Yao, Theory and applications of trapdoor functions, in: Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 80–91.
88. G. Lowe, Casper: A compiler for the analysis of security protocols, in: Proc. 10th IEEE Computer Security Foundations Workshop (CSFW), 1997, pp. 18–30.