

A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol*

Michael Backes and Birgit Pfitzmann

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

Abstract. We prove the Needham-Schroeder-Lowe public-key protocol secure under real, active cryptographic attacks including concurrent protocol runs. This proof is based on an abstract cryptographic library, which is a provably secure abstraction of a real cryptographic library. Together with composition and integrity preservation theorems from the underlying model, this allows us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model.

Our proof is one of the two first independent cryptographically sound security proofs of this protocol.

It is the first protocol proof over an abstract Dolev-Yao-style library that is in the scope of formal proof tools and that automatically yields cryptographic soundness. We hope that it paves the way for the actual use of automatic proof tools for this and many similar cryptographically sound proofs of security protocols.

1 Introduction

In recent times, the analysis of cryptographic protocols has been getting more and more attention, and the demand for rigorous proofs of cryptographic protocols has been rising.

One way to conduct such proofs is the cryptographic approach, whose security definitions are based on complexity theory, e.g., [13, 12, 14, 7]. The security of a cryptographic protocol is proved by reduction, i.e., by showing that breaking the protocol implies breaking one of the underlying cryptographic primitives with respect to its cryptographic definition and thus finally a computational assumption such as the hardness of integer factoring. This approach captures a very comprehensive adversary model and allows for mathematically rigorous and precise proofs. However, because of probabilism and complexity-theoretic restrictions, these proofs have to be done by hand so far, which yields proofs with faults and imperfections. Moreover, such proofs rapidly become too complex for larger protocols.

The alternative is the formal-methods approach, which is concerned with the automation of proofs using model checkers and theorem provers. As these tools currently cannot deal with cryptographic details like error probabilities and computational restrictions, abstractions of cryptography are used. They are almost always based on the

* An extended version of this paper is available as [5].

so-called Dolev-Yao model [11]. This model simplifies proofs of larger protocols considerably and gave rise to a large body of literature on analyzing the security of protocols using various techniques for formal verification, e.g., [20, 18, 15, 9, 22, 1].

A prominent example demonstrating the usefulness of the formal-methods approach is the work of Lowe [16], where he found a man-in-the-middle attack on the well-known Needham-Schroeder public-key protocol [21]. Lowe later proposed a repaired version of the protocol [17] and used the model checker FDR 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., [26, 19, 25, 27]. Various new approaches and formal proof tools for the analysis of security protocols were validated by showing that they can discover the known flaw or prove the repaired protocol in the Dolev-Yao model.

It is well-known and easy to show that the security flaw of the original protocol in the formal-methods approach can as well be used to mount a successful attack against any cryptographic implementation of the protocol. However, all prior security proofs of the repaired protocol are restricted to the Dolev-Yao model, i.e., no theorem exists that allows for carrying over the results of an existing proof to the cryptographic approach with its much more comprehensive adversary. Although recent research focused on moving towards such a theorem, i.e., a cryptographically sound foundation of the formal-methods approach, the results are either specific for passive adversaries [3, 2] or they do not capture the local evaluation of nested cryptographic terms [10, 24], which is needed to model many usual cryptographic protocols. A recently proposed cryptographic library [6] allows for such nesting, but has not been applied to any security protocols yet. Thus, it is still an open problem to conduct a formal protocol proof that an actual cryptographic implementation is secure under active attacks with respect to cryptographic security definitions.

We close this gap by providing a security proof of the Needham-Schroeder-Lowe public-key protocol in the cryptographic approach. Our proof is based on the cryptographic library from [6], which is abstract in the sense needed for theorem provers but nevertheless has a provably secure implementation. Together with composition and integrity preservation theorems from the underlying model, this allows us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model.

Independently and concurrently to this work, another cryptographically sound proof of the Needham-Schroeder-Lowe public-key protocol has been invented in [28]. The proof is conducted from scratch in the cryptographic approach. It establishes a stronger security property. The benefit of our proof is that it is sufficient to prove the security of the Needham-Schroeder-Lowe protocol based on the deterministic abstractions offered by the cryptographic library; then the result automatically carries over to the cryptographic setting. As the proof is both deterministic and rigorous, it should be easily expressible in formal proof tools, in particular theorem provers. Even done by hand, our proof is much less prone to error than a reduction proof conducted from scratch in the cryptographic approach.

We hope that our proof paves the way for the actual use of automatic proof tools for this and many similar cryptographically sound proofs of security protocols. In particular, we are confident that stronger properties of the Needham-Schroeder-Lowe protocol can be proved in the same way, but this should become much simpler once the transition to automatic proof tools has been made based on this first, hand-proved example.

2 Preliminaries

In this section, we give an overview of the ideal cryptographic library of [6] and briefly sketch its provably secure implementation. We start by introducing notation.

2.1 Notation

We write “:=” for deterministic and “ \leftarrow ” for probabilistic assignment, and \downarrow is an error element added to the domains and ranges of all functions and algorithms. The list operation is denoted as $l := (x_1, \dots, x_j)$, 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 $D := x$.

2.2 Overview of the Ideal and Real Cryptographic Library

The ideal (abstract) cryptographic library of [6] 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. To allow a reactive scenario, this semantics is based on state, e.g., of who already knows which terms; the state is represented as a database. Each entry has a type (e.g., “ciphertext”), and pointers to its arguments (e.g., a key and a message). Further, each entry contains handles for those participants who already know it. A send operation makes an entry known to other participants, i.e., it adds handles to the entry. 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 database entry for the ciphertext. Another user can only ask for decryption of this ciphertext if he has obtained handles to both the ciphertext and the secret key.

To allow for the proof of cryptographic faithfulness, the library is based on a detailed model of asynchronous reactive systems introduced in [24] and represented as a deterministic machine $\text{TH}(\mathcal{H})$, called *trusted host*. The parameter $\mathcal{H} \subseteq \{1 \dots, n\}$ denotes the honest participants, where n is a parameter of the library denoting the overall number of participants. Depending on the considered set \mathcal{H} , the trusted host offers slightly extended capabilities for the adversary. However, for current purposes, the trusted host can be seen as a slightly modified Dolev-Yao model together with a network and intruder model, similar to “the CSP Dolev-Yao model” or “the inductive-approach Dolev-Yao model”.

The real cryptographic library offers its users the same commands as the ideal one, i.e., honest users operate on cryptographic objects via handles. The objects are now real cryptographic keys, ciphertexts, etc., handled by real distributed machines. 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 implementation of the commands is based on arbitrary secure encryption and signature systems according to standard cryptographic definitions, e.g., adaptive chosen-ciphertext security in case of public-key encryption, with certain additions like type tagging and additional randomizations.

The security proof of [6] states that the real library is *at least as secure* as the ideal library. This is captured using the notion of *simulatability*, which states that whatever an adversary can achieve in the real implementation, another adversary can achieve given the ideal library, or otherwise the underlying cryptography can be broken [24]. This is the strongest possible cryptographic relationship between a real and an ideal system. In particular it covers active attacks. Moreover, a composition theorem exists in the underlying model [24], which states that one can securely replace the ideal library in larger systems with the real library, i.e., without destroying the already established simulatability relation.

3 The Needham-Schroeder-Lowe Public-Key Protocol

The original Needham-Schroeder protocol and Lowe's variant consist of seven steps, where four steps deal with key generation and public-key distribution. These steps 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 the remaining 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, e.g., in [16].

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

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 that she spoke with u .

3.1 The Needham-Schroeder-Lowe Protocol Using the Abstract Library

We now show how to model the Needham-Schroeder-Lowe protocol in the framework of [24] and using the ideal cryptographic library. For each user $u \in \{1, \dots, n\}$, we de-

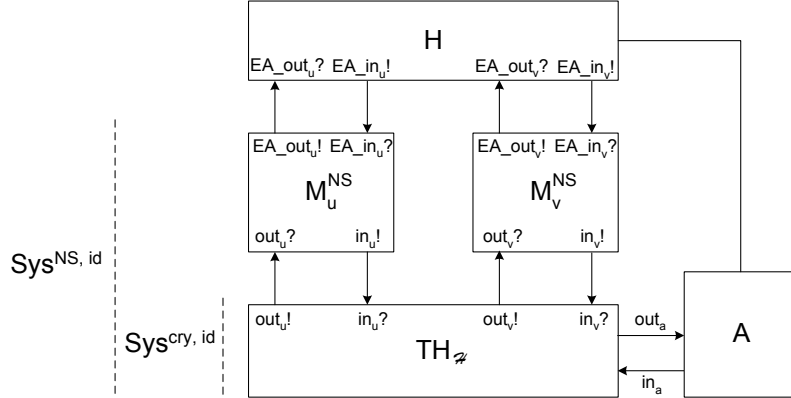


Fig. 1. Overview of the Needham-Schroeder-Lowe Ideal System.

fine a machine M_u^{NS} , called a *protocol machine*, which executes the protocol sketched above for participant identity u , allowing an arbitrary (polynomial) number of concurrent executions.¹ This machine is connected to its user via ports $EA_out_u!$, $EA_in_u?$ (“EA” for “Entity Authentication”, because the behavior at these ports is the same for all entity authentication protocols) and to the cryptographic library via ports $in_u!$, $out_u?$. The notation follows the CSP convention, e.g., the cryptographic library has a port $in_u?$ where it obtains messages output at $in_u!$. The combination of the protocol machines M_u^{NS} and the trusted host $TH(\mathcal{H})$ is the *ideal Needham-Schroeder-Lowe system* $Sys^{\text{NS}, id}$. It is shown in Figure 1; H and A model the arbitrary joint honest users and the adversary, respectively.

Using the notation of [24], the system $Sys^{\text{NS}, id}$ consists of several *structures* $(M(\mathcal{H}), S(\mathcal{H}))$, one for each value of the parameter \mathcal{H} . Each structure consists of a set $M(\mathcal{H}) := \{TH(\mathcal{H})\} \cup \{M_u^{\text{NS}} \mid u \in \mathcal{H}\}$ of machines, i.e., for a given set \mathcal{H} of honest users, only the machines M_u^{NS} with $u \in \mathcal{H}$ are actually present in a system run. The others are subsumed in the adversary. $S(\mathcal{H})$ denotes those ports of $M(\mathcal{H})$ that the honest users connect to, i.e., $S(\mathcal{H}) := \{EA_in_u?, EA_out_u! \mid u \in \mathcal{H}\}$. Formally, we obtain $Sys^{\text{NS}, id} := \{(M(\mathcal{H}), S(\mathcal{H})) \mid \mathcal{H} \subseteq \{1, \dots, n\}\}$.

In order to capture that keys have been generated and distributed, we assume that suitable entries for the keys already exist in the database of $TH(\mathcal{H})$. We denote the handle of u_1 to the public key of u as pk_{u, u_1}^{hnd} and the handle of u to its secret key as sk_u^{hnd} .

¹ We could define local submachines per protocol execution. However, one also needs a dispatcher submachine for user u to dispatch incoming protocol messages to the submachines by the nonces, and user inputs to new submachines. We made such a construction once for a case with complicated submachines [23], and the dispatcher could be reused for all protocols with a fixed style of distinguishing protocol runs. However, the Needham-Schroeder-Lowe protocol does not use an existing dispatcher, and for such an almost stateless protocol splitting machines into submachines rather complicates the invariants.

The state of the machine M_u^{NS} consists of the bitstring u and a family $(\text{Nonce}_{u,v})_{v \in \{1, \dots, n\}}$ of sets of handles. Each set $\text{Nonce}_{u,v}$ is initially empty. We now define how the machine M_u^{NS} evaluates inputs. They either come from user u at port $\text{EA_in}_u?$ or from $\text{TH}(\mathcal{H})$ at port $\text{out}_u?$. The behavior of M_u^{NS} in both cases is described in Algorithm 1 and 2 respectively, which we will describe below. We refer to Step i of Algorithm j as Step $j.i$. Both algorithms should immediately abort if a command to the cryptographic library does not yield the desired result, e.g., if a decryption request fails. For readability we omit these abort checks in the algorithm descriptions; instead we impose the following convention.

Convention 1 *If M_u^{NS} enters a command at port $\text{in}_u!$ and receives \downarrow at port $\text{out}_u?$ as the immediate answer of the cryptographic library, then M_u^{NS} aborts the execution of the current algorithm, except if the command was of the form `list_proj` or `send_i`.*

At any time, the user of the machine M_u^{NS} can start a new protocol execution with any user $v \in \{1, \dots, n\} \setminus \{u\}$ by inputting `(new_prot, v)` at port $\text{EA_in}_u?$. Our security proof holds for all adversaries and all honest users, i.e., especially those that start protocols with the adversary (respectively a malicious user) concurrently with protocols with honest users. Upon such an input, M_u^{NS} builds up the term corresponding to the first protocol message using the ideal cryptographic library $\text{TH}(\mathcal{H})$ according to Algorithm 1. The command `gen_nonce` generates the ideal nonce. M_u^{NS} stores the resulting handle n_u^{hnd} in $\text{Nonce}_{u,v}$ 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. Since only lists are allowed to be transferred in $\text{TH}(\mathcal{H})$ (because the list-operation is a convenient place to concentrate all verifications that no secret items are put into messages), the encryption is packed as a list again. The final command `send_i` means that M_u^{NS} sends the resulting term to v over an insecure channel (called channel type i). The effect is that the adversary obtains a handle to the term and can decide what to do with it, e.g., forward it to M_v^{NS} , delay it compared with concurrent protocol executions, or modify it.

The behavior of M_u^{NS} upon receiving an input from the cryptographic library at port $\text{out}_u?$ corresponding to a message that arrives over the network is defined similarly in Algorithm 2. By construction of $\text{TH}(\mathcal{H})$, such an input is always of the form $(v, u, i, m^{\text{hnd}})$, where v is the supposed sender, u the recipient, i the channel type “insecure” (the only type used here), and m^{hnd} the handle to the received message, which is always a list. M_u^{NS} first decrypts the list content using the secret key of user u , which 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, M_u^{NS} generates a new nonce and stores its handle in $\text{Nonce}_{u,v}$. After that, M_u^{NS} builds up a new list according to the protocol description, encrypts the list and sends it to user v . If the list has three elements, i.e., it could correspond to the second message of the protocol, then M_u^{NS} tests whether the third list element equals v and whether the first list element is already contained in the set $\text{Nonce}_{u,v}$. If one of these tests does not succeed, M_u^{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, then M_u^{NS} tests if the handle of this element is already contained in the set $\text{Nonce}_{u,v}$.

Algorithm 1 Evaluation of Inputs from the User (Protocol Start)

Input: $(\text{new_prot}, v)$ at $\text{EA_in}_u?$ with $v \in \{1, \dots, n\} \setminus \{u\}$.

- 1: $n_u^{\text{hnd}} \leftarrow \text{gen_nonce}()$.
- 2: $\text{Nonce}_{u,v} := \text{Nonce}_{u,v} \cup \{n_u^{\text{hnd}}\}$.
- 3: $u^{\text{hnd}} \leftarrow \text{store}(u)$.
- 4: $l_1^{\text{hnd}} \leftarrow \text{list}(n_u^{\text{hnd}}, u^{\text{hnd}})$.
- 5: $c_1^{\text{hnd}} \leftarrow \text{encrypt}(pk_{v,u}^{\text{hnd}}, l_1^{\text{hnd}})$.
- 6: $m_1^{\text{hnd}} \leftarrow \text{list}(c_1^{\text{hnd}})$.
- 7: $\text{send}_i(v, m_1^{\text{hnd}})$.

If so, M_u^{NS} outputs (ok, v) at $\text{EA_out}_u!$. This signals that the protocol with user v has terminated successfully, i.e., u believes that he spoke with v .

3.2 On Polynomial Runtime

In order to use existing composition results of the underlying model, the machines M_u^{NS} have to be polynomial-time. Similar to the cryptographic library, we hence define that each machine M_u^{NS} maintains explicit polynomial bounds on the message lengths and the number of inputs accepted at each port.

4 Formalizing the Security Property

The security property that we prove is entity authentication of the initiator. It 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 $\text{EA_out}_v!$ can only happen if there was a prior input $(\text{new_prot}, v)$ at $\text{EA_in}_u?$. This property and the protocol as defined above do not consider replay attacks. This can be added to the protocol as follows: If M_u^{NS} receives a message from v containing a nonce and M_u^{NS} created this nonce, then it additionally removes this nonce from the set $\text{Nonce}_{u,v}$, i.e., after Steps 2.20 and 2.25, the handle x_1^{hnd} is removed from $\text{Nonce}_{u,v}$.²

Integrity properties in the underlying model are formally sets of traces at the in- and output ports connecting the system to the honest users, i.e., here traces at the port set $S(\mathcal{H}) = \{\text{EA_out}_u!, \text{EA_in}_u? \mid u \in \mathcal{H}\}$. Intuitively, such an integrity property Req states which are 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 input or output port p . The t -th step of a trace r is written r_t ; we also speak of the step at time t . Thus the integrity requirement Req^{EA} of entity authentication of the initiator is formally defined as follows:

² Proving freshness and two-sided authentication is certainly useful future work, in particular once the proof has been automated. We do not intend to prove the property of matching conversation from [8]. It makes constraints on events within the system; this cannot be expressed in an approach based on abstraction and modularization. We see it as a sufficient, but not necessary condition for the desired abstract properties. Some additional properties associated with matching conversations only become meaningful at an abstract level if one goes beyond entity authentication to session establishment.

Algorithm 2 Evaluation of Inputs from $\text{TH}(\mathcal{H})$ (Network Inputs)

Input: $(v, u, i, m^{\text{hnd}})$ at $\text{out}_u?$ with $v \in \{1, \dots, n\} \setminus \{u\}$.

- 1: $c^{\text{hnd}} \leftarrow \text{list_proj}(m^{\text{hnd}}, 1)$
- 2: $l^{\text{hnd}} \leftarrow \text{decrypt}(sk_e^{\text{hnd}}, c^{\text{hnd}})$
- 3: $x_i^{\text{hnd}} \leftarrow \text{list_proj}(l^{\text{hnd}}, i)$ for $i = 1, 2, 3$.
- 4: **if** $x_1^{\text{hnd}} \neq \downarrow \wedge x_2^{\text{hnd}} \neq \downarrow \wedge x_3^{\text{hnd}} = \downarrow$ **then** {First Message is input}
- 5: $x_2 \leftarrow \text{retrieve}(x_2^{\text{hnd}})$.
- 6: **if** $x_2 \neq v$ **then**
- 7: Abort
- 8: **end if**
- 9: $n_u^{\text{hnd}} \leftarrow \text{gen_nonce}()$.
- 10: $\text{Nonce}_{u,v} := \text{Nonce}_{u,v} \cup \{n_u^{\text{hnd}}\}$.
- 11: $u^{\text{hnd}} \leftarrow \text{store}(u)$.
- 12: $l_2^{\text{hnd}} \leftarrow \text{list}(x_1^{\text{hnd}}, n_u^{\text{hnd}}, u^{\text{hnd}})$.
- 13: $c_2^{\text{hnd}} \leftarrow \text{encrypt}(pk_{e_{v,u}}^{\text{hnd}}, l_2^{\text{hnd}})$.
- 14: $m_2^{\text{hnd}} \leftarrow \text{list}(c_2^{\text{hnd}})$.
- 15: $\text{send}_i(v, m_2^{\text{hnd}})$.
- 16: **else if** $x_1^{\text{hnd}} \neq \downarrow \wedge x_2^{\text{hnd}} \neq \downarrow \wedge x_3^{\text{hnd}} \neq \downarrow$ **then** {Second Message is input}
- 17: $x_3 \leftarrow \text{retrieve}(x_3^{\text{hnd}})$.
- 18: **if** $x_3 \neq v \vee x_1^{\text{hnd}} \notin \text{Nonce}_{u,v}$ **then**
- 19: Abort
- 20: **end if**
- 21: $l_3^{\text{hnd}} \leftarrow \text{list}(x_2^{\text{hnd}})$.
- 22: $c_3^{\text{hnd}} \leftarrow \text{encrypt}(pk_{e_{v,u}}^{\text{hnd}}, l_3^{\text{hnd}})$.
- 23: $m_3^{\text{hnd}} \leftarrow \text{list}(c_3^{\text{hnd}})$.
- 24: $\text{send}_i(v, m_3^{\text{hnd}})$.
- 25: **else if** $x_1^{\text{hnd}} \in \text{Nonce}_{u,v} \wedge x_2^{\text{hnd}} = x_3^{\text{hnd}} = \downarrow$ **then** {Third Message is input}
- 26: Output (ok, v) at $\text{EA_out}_u!$.
- 27: **end if**

Definition 1. (*Entity Authentication Requirement*) A trace r is contained in Req^{EA} if for all $u, v \in \mathcal{H}$:

$$\begin{aligned} \exists t_1 \in \mathbb{N}: \text{EA_out}_v!(\text{ok}, u) \in r_{t_1} & \quad \# \text{ If } v \text{ believes she speaks with } u \text{ at time } t_1 \\ \Rightarrow \exists t_0 < t_1: & \quad \# \text{ then there exists a past time } t_0 \\ \text{EA_in}_u?(\text{new_prot}, v) \in r_{t_0} & \quad \# \text{ in which } u \text{ started a protocol with } v \end{aligned}$$

The notion of a system Sys fulfilling an integrity property Req essentially comes in two flavors [4]. *Perfect fulfillment*, $\text{Sys} \models^{\text{perf}} \text{Req}$, means that the integrity property holds for all traces arising in runs of Sys (a well-defined notion from the underlying model [24]). *Computational fulfillment*, $\text{Sys} \models^{\text{poly}} \text{Req}$, means that the property only holds for polynomially bounded users and adversaries, and only with negligible error probability. Perfect fulfillment implies computational fulfillment.

The following theorem captures the security of the ideal Needham-Schroeder-Lowe protocol.

Theorem 1. (Security of the Needham-Schroeder-Lowe Protocol based on the Ideal Cryptographic Library) Let $Sys^{NS,id}$ be the ideal Needham-Schroeder-Lowe system defined in Section 3, and Req^{EA} the integrity property of Definition 1. Then $Sys^{NS,id} \models^{perf} Req^{EA}$.

5 Proof of the Cryptographic Realization

If Theorem 1 has been proven, it follows that the Needham-Schroeder-Lowe protocol based on the real cryptographic library computationally fulfills the integrity requirement Req^{EA} . The main tool is the following *preservation theorem* from [4].

Theorem 2. (Preservation of Integrity Properties (Sketch)) Let two systems Sys_1, Sys_2 be given such that Sys_1 is computationally at least as secure as Sys_2 (written $Sys_1 \geq_{sec}^{poly} Sys_2$). Let Req be an integrity requirement for both Sys_1 and Sys_2 , and let $Sys_2 \models^{poly} Req$. Then also $Sys_1 \models^{poly} Req$.

Let $Sys^{cry,id}$ and $Sys^{cry,real}$ denote the ideal and the real cryptographic library from [6], and $Sys^{NS,real}$ the Needham-Schroeder-Lowe protocol based on the real cryptographic library. This is well-defined given the formalization with the ideal library because the real library has the same user ports and offers the same commands.

Theorem 3. (Security of the Real Needham-Schroeder-Lowe Protocol) Let Req^{EA} denote the integrity property of Definition 1. Then $Sys^{NS,real} \models^{poly} Req^{EA}$.

Proof. In [6] it was shown that $Sys^{cry,real} \geq_{sec}^{poly} Sys^{cry,id}$ holds for suitable parameters in the ideal system. Since $Sys^{NS,real}$ is derived from $Sys^{NS,id}$ by replacing the ideal with the real cryptographic library, $Sys^{NS,real} \geq_{sec}^{poly} Sys^{NS,id}$ follows from the composition theorem of [24]. We only have to show that the theorem's preconditions are fulfilled. This is straightforward, since the machines M_u^{NS} are polynomial-time (cf. Section 3.2). Now Theorem 1 implies $Sys^{NS,id} \models^{poly} Req^{EA}$, hence Theorem 2 yields $Sys^{NS,real} \models^{poly} Req^{EA}$.

6 Proof in the Ideal Setting

This section sketches the proof of Theorem 1, i.e., the proof of the Needham-Schroeder-Lowe protocol using the ideal, deterministic cryptographic library. A complete proof can be found in the long version of this paper [5], together with a short version of the notation of the cryptographic library. 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. This is somewhat similar to formal proofs using the Dolev-Yao model; indeed the similarity supports our hope that the new, sound cryptographic library can be used in the place of the Dolev-Yao model in automated tools.

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 and are mainly needed to prove the last invariant, *correct list owner*, which establishes who created certain terms.

Invariant 1 (*Correct Nonce Owner*) For all $u \in \mathcal{H}, v \in \{1, \dots, n\}$ and for all $x^{\text{hnd}} \in Nonce_{u,v}$, we have $D[\text{hnd}_u = x^{\text{hnd}}] \neq \downarrow$ and $D[\text{hnd}_u = x^{\text{hnd}}].\text{type} = \text{nonce}$.

Invariant 2 (*Unique Nonce Use*) For all $u, v \in \mathcal{H}$, all $w, w' \in \{1, \dots, n\}$, and all $j \leq \text{size}$: If $D[j].\text{hnd}_u \in Nonce_{u,w}$ and $D[j].\text{hnd}_v \in Nonce_{v,w'}$, 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. For the formalization, note that the handles to these nonces form the sets $Nonce_{u,v}$. The claim is that the other users and the adversary have no handles to such a nonce in the database D of $\text{TH}(\mathcal{H})$:

Invariant 3 (*Nonce Secrecy*) For all $u, v \in \mathcal{H}$ and for all $j \leq \text{size}$: If $D[j].\text{hnd}_u \in Nonce_{u,v}$ then $D[j].\text{hnd}_w = \downarrow$ for all $w \in (\mathcal{H} \cup \{\mathbf{a}\}) \setminus \{u, v\}$.

Similarly, the invariant *nonce-list secrecy* states that a list containing such a handle can only be known to u and v . Further, it states that the identity fields in such lists are correct. 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 .

Invariant 4 (*Nonce-List Secrecy*) For all $u, v \in \mathcal{H}$ and for all $j \leq \text{size}$ with $D[j].\text{type} = \text{list}$: Let $x_i^{\text{ind}} := D[j].\text{arg}[i]$ for $i = 1, 2, 3$. If $D[x_i^{\text{ind}}].\text{hnd}_u \in Nonce_{u,v}$ then

- $D[j].\text{hnd}_w = \downarrow$ for all $w \in (\mathcal{H} \cup \{\mathbf{a}\}) \setminus \{u, v\}$.
- if $D[x_{i+1}^{\text{ind}}].\text{type} = \text{data}$, then $D[x_{i+1}^{\text{ind}}].\text{arg} = (u)$.
- for all $k \leq \text{size}$ we have $j \in D[k].\text{arg}$ only if $D[k].\text{type} = \text{enc}$ and $D[k].\text{arg}[1] \in \{pk_{e_u}, pk_{e_v}\}$.

The invariant *correct list owner* states that certain protocol messages can only be constructed by the “intended” users. For example, 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 a non-cryptographic construct (formally of type data) then it must have been created by user u . Similar statements exist for the second and third protocol message.

Invariant 5 (*Correct List Owner*) For all $u, v \in \mathcal{H}$ and for all $j \leq \text{size}$ with $D[j].\text{type} = \text{list}$: Let $x_i^{\text{ind}} := D[j].\text{arg}[i]$ and $x_{i,u}^{\text{hnd}} := D[x_i^{\text{ind}}].\text{hnd}_u$ for $i = 1, 2$.

- If $x_{1,u}^{\text{hnd}} \in Nonce_{u,v}$ and $D[x_2^{\text{ind}}].\text{type} = \text{data}$, then $D[j]$ was created by M_u^{NS} in Step 1.4 (in one of its protocol executions).
- If $D[x_1^{\text{ind}}].\text{type} = \text{nonce}$ and $x_{2,u}^{\text{hnd}} \in Nonce_{u,v}$, then $D[j]$ was created by M_u^{NS} in Step 2.12.

- If $x_{1,u}^{\text{hd}} \in \text{Nonce}_{u,v}$ and $x_2^{\text{ind}} = \downarrow$, then $D[j]$ was created by M_v^{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 .

Acknowledgments

We thank Michael Waidner and the anonymous reviewers for interesting comments.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
2. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
3. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
4. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.
5. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. IACR Cryptology ePrint Archive 2003/121, June 2003. <http://eprint.iacr.org/>.
6. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. IACR Cryptology ePrint Archive 2003/015, Jan. 2003. <http://eprint.iacr.org/>.
7. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.
8. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1994.
9. M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.
10. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.
11. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
13. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.

14. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.
15. R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.
16. G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–135, 1995.
17. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
18. C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
19. 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)*, volume 1146 of *Lecture Notes in Computer Science*, pages 351–364. Springer, 1996.
20. J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
21. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.
22. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
23. B. Pfizmann, M. Schunter, and M. Waidner. Provably secure certified mail. Research Report RZ 3207, IBM Research, 2000. <http://www.zurich.ibm.com/security/publications/>.
24. B. Pfizmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.
25. S. Schneider. Verifying authentication protocols with CSP. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 3–17, 1997.
26. P. Syverson. A new look at an old protocol. *Operation Systems Review*, 30(3):1–4, 1996.
27. F. J. Thayer Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 160–171, 1998.
28. B. Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 248–262, 2003.