Technical Report: A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol (Long Version) *

Michael Backes and Birgit Pfitzmann
IBM Zurich Research Lab, Rüschlikon, Switzerland
{mbc,bpf}@zurich.ibm.com

October 29, 2006

Abstract

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 it is secure against arbitrary active attacks if it is implemented using standard provably secure cryptographic primitives. Nevertheless, our proof does not have to deal with the probabilistic aspects of cryptography and is hence in the scope of current automated proof tools. We achieve this by exploiting a recently proposed Dolev-Yao-style cryptographic library with a provably secure cryptographic implementation. Besides establishing the cryptographic security of the Needham-Schroeder-Lowe protocol, our result 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

Cryptographic protocols for authentication and key establishment are an established technology. Nevertheless, most new networking and messaging stacks come with new protocols for such tasks. As the design of cryptographic protocols is very error-prone, the demand for rigorous proofs has been rising.

One way to conduct such proofs is the cryptographic approach. Its security definitions are based on complexity theory, e.g., [45, 43, 34]. 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 proofs. However, because of probabilism and computational restrictions, these proofs have to be done by hand so far, which often yields proofs with faults or gaps. Moreover, such proofs rapidly become too complex for larger protocols.

*Parts of this work were published in [16] and [19].

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.[1] They are almost always based on the so-called Dolev-Yao model [41], which represents cryptography as term algebras. The original Dolev-Yao model was extended in many papers, e.g., [42, 63]. The use of term algebras simplifies proofs of larger protocols considerably and led to a large body of literature on analyzing protocol security using various techniques for formal verification, e.g., [66, 61, 50, 35, 70, 1].

A prominent example of the usefulness of the formal-methods approach is Lowe's discovery of a man-in-the-middle attack on the well-known Needham-Schroeder public-key authentication protocol [69, 53]. Lowe later proposed a repaired version of the protocol [54] 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., [75, 62, 74, 76]. 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 [73] and formulated as IND-CCA2 in [33]. Efficient encryption systems secure in this sense exist under reasonable assumptions [39].

Our proof relies on a recent general result that a so-called ideal cryptographic library, which implements a slightly extended Dolev-Yao model, can be securely realized by a specific cryptographic implementation. 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 library (i.e., the extended Dolev-Yao model), 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.

1.1 Related Work

Concurrently to this work, Warinschi also gave a cryptographically sound security proof for the Needham-Schroeder-Lowe protocol [77]. This proof is done from scratch in the cryptographic approach. Hence even if it had preceded ours, we would be giving the first example of a

[1]Efforts exist to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [67, 68, 48]. However, this approach cannot handle protocols with any degree of automation yet.

cryptographically sound proof of a cryptographic protocol via a deterministic, Dolev-Yao-style idealization of cryptography. On the other hand, Warinschi proves stronger properties; we discuss this in Section 4. He further shows that chosen-plaintext-secure encryption is insufficient for the security of the protocol.

Work on justifying Dolev-Yao-style models under cryptographic definitions prior to [25] was restricted to passive adversaries and symmetric encryption [3, 2, 51]. Concurrently to [25], an extension to asymmetric encryption, but still under passive attacks only, was presented in [47]. The underlying masters thesis [46] considers asymmetric encryption under active attacks, but in the random oracle model, which is itself an idealization of cryptography and not justifiable [38]. The recent work [65] gives a slightly more efficient implementation of asymmetric encryption than [25] (no additional tagging and randomization) at the cost of a much less general library and a weaker security notion – the outlook in [65] would essentially give [25] again. Since the original publication of our results in [16], computational soundness has become a highly active line of research, see, e.g., line of research, see e.g., [4, 26, 13, 31, 28, 20, 22, 8, 14].

The security notion used for the relation between the ideal Dolev-Yao-style library and its cryptographic implementation, reactive simulatability, and its composition properties were introduced in [71] and extended to asynchronous systems in [72, 37, 29, 27]. It extends the security notions of multi-party (one-step) function evaluation [78, 43, 44, 64, 32, 36] and the observational equivalence of [52]. One of the overarching goals of the reactive notions was to offer composable security guarantees, which constitutes a long-standing open problem for many security notions in the present and past, see, e.g., [57, 49, 58, 59, 60, 56, 40, 17, 24, 18, 30, 5, 10, 28, 9, 21, 11]). There are multiple possible layers of sound abstraction from cryptography in the sense of reactive simulatability besides Dolev-Yao-style cryptographic libraries. They reach from low-level idealizations that still have real cryptographic in- and outputs to high-level abstractions like secure channels. The specific aspects of a Dolev-Yao-style abstraction are simple operator-tree abstractions from nested cryptographic terms, the restriction of adversary capabilities to algebraic operations on such terms, and the assumption that terms whose equality cannot be derived explicitly are always unequal.

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.

2 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., [53].

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

Here, user u seeks to establish a session with user v. He generates a nonce N_u and sends it to v together with hisidentity, 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.

3 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 [25], 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.[2] We first explain this formulation, and then explain general aspects of the surrounding framework as far as needed in our proofs.

3.1 Detailed Protocol Descriptions

We write ":=" for deterministic and "←" for probabilistic assignment, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The 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 so-called port $\mathsf{EA\_in}_u?$. The "?" for input ports follows the CSP convention, and "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.

The command $\mathsf{gen\_nonce}$ 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 $\mathsf{encrypt}$ is encryption. The final command $\mathsf{send\_i}$ means that $\mathsf{M}_u^{\mathsf{NS}}$ attempts to send the resulting term to v over an insecure channel. The superscript $^{\mathsf{hnd}}$ on most parameters denotes that these are so-called handles, i.e., local names that this machine has for the corresponding terms. This is an important aspect of [25] because it allows the same protocol description to be implemented once with Dolev-Yao-style idealized cryptography and once with real cryptography. More precisely, the five

[2]For some frameworks there are compilers to generate these detailed protocol descriptions, e.g., [55]. This should be possible for this framework in a similar way.

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

Input: $(\mathsf{new\_prot}, v)$ at $\mathsf{EA\_in}_u?$ with $v \in \{1, \ldots, n\} \setminus \{u\}$.

$n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.

$Nonce_{u,v} := Nonce_{u,v} \cup \{n_u^{\mathsf{hnd}}\}$.

$u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.

$l_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(n_u^{\mathsf{hnd}}, u^{\mathsf{hnd}})$.

$c_1^{\mathsf{hnd}} \leftarrow \mathsf{encrypt}(pke_{v,u}^{\mathsf{hnd}}, l_1^{\mathsf{hnd}})$.

$m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(c_1^{\mathsf{hnd}})$.

$\mathsf{send\_i}(v, m_1^{\mathsf{hnd}})$.

commands we saw so far and their input and output domains belong to the interface (in the same sense as, e.g., a Java interface) of the underlying cryptographic library. This interface is implemented by both the idealized and the real version. In the first case, the handles are local names of Dolev-Yao-style terms, in the second case of real cryptographic bitstrings. We say more about these two implementations below. The effect of $\mathsf{send\_i}$ in the ideal implementation is that the adversary obtains a handle to the Dolev-Yao-style term and can decide what to do with it (such as forwarding it to $\mathsf{M}_v^{\mathsf{NS}}$ or performing Dolev-Yao-style algebraic operations on the term); the effect in the real implementation is that the adversary obtains the real bitstring and can perform arbitrary bit manipulations on it. The list operation directly before sending is a technicality: 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.

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 $\mathsf{list\_proj}$. 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* $\mathsf{list\_proj}$

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

Input: $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ at $\mathsf{out}_u?$ 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

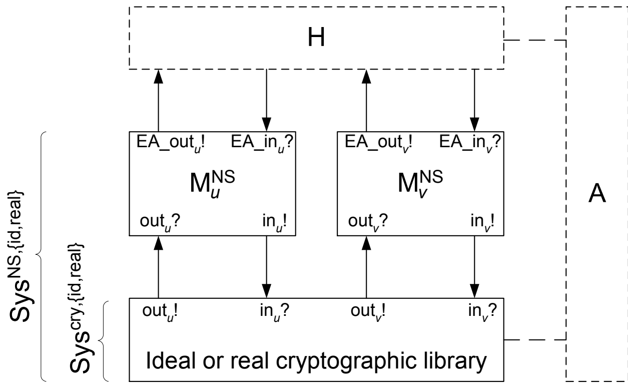

Figure 1: Overview of the Needham-Schroeder-Lowe Ideal System

or send_i.

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

3.2 Initial State

We have assumed in the algorithms that each Needham-Schroeder machine $\mathsf{M}_u^{\mathsf{NS}}$ already has a handle ske_u^{hnd} to its own secret encryption key and handles $pke_{v,u}^{\mathsf{hnd}}$ to the corresponding public keys of every participant v. The cryptographic library can also represent key generation and distribution by normal commands. Further, each machine $\mathsf{M}_u^{\mathsf{NS}}$ contains the bitstring u denoting its identity, and the family $(Nonce_{u,v})_{v\in\{1,\dots,n\}}$ of initially empty sets of (nonce) handles.

3.3 Overall Framework and Adversary Model

The framework that determines how machines such as our Needham-Schroeder machines and the machines of the idealized or real cryptographic library execute is taken from [72]. The basis is an asynchronous probabilistic execution model with distributed scheduling. We already used implicitly above that for term construction and parsing commands to the cryptographic library, so-called local scheduling is defined, i.e., a result is returned immediately. The idealized or real network sending via this library, however, is scheduled by the adversary.

When protocol machines such as $\mathsf{M}_u^{\mathsf{NS}}$ for certain users $u \in \{1,\dots,n\}$ are defined, there is no guarantee that all these machines are correct. A trust model determines for what subsets $\mathcal{H}$ of $\{1,\dots,n\}$ 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 is called a structure, and the set of these structures is called the system. 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. 1. 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 $S^{\mathsf{cry}}_{\mathcal{H}} := \{\mathsf{in}_u?, \mathsf{out}_u! \mid u \in \mathcal{H}\}$. This is where the Needham-Schroeder machines input their cryptographic commands and obtain results 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 $Sys^{\mathsf{NS,real}} := \{(\hat{M}^{\mathsf{NS}}_{\mathcal{H}} \cup \hat{M}^{\mathsf{cry}}_{\mathcal{H}}, S^{\mathsf{EA}}_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1, \ldots, n\}\}$.

3.4 On Polynomial Runtime

In order to be valid users of the real cryptographic library, the machines $\mathsf{M}^{\mathsf{NS}}_u$ have to be polynomial-time. We therefore define that each machine $\mathsf{M}^{\mathsf{NS}}_u$ 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.

4 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 $\mathsf{M}^{\mathsf{NS}}_u$ 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 [77]. The even stronger property of matching conversations from [34] 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 $S^{\mathsf{EA}}_{\mathcal{H}}$. 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 4.1 *(Entity Authentication Requirement) A trace r is contained in Req^{EA} if for all $u, v \in \mathcal{H}$:*

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

$\diamond$

The notion of a system *Sys* fulfilling an integrity property *Req* essentially comes in two flavors [12]. *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 [72]). *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 4.1 *(Security of the Needham-Schroeder-Lowe Protocol) For the Needham-Schroeder-Lowe systems from Section 3.3 and the integrity property of Definition 4.1, we have* $Sys^{\mathsf{NS,id}} \models^{\mathsf{perf}} Req^{\mathsf{EA}}$ *and* $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$. $\square$

5 Proof of the Cryptographic Realization from the Idealization

As discussed in the introduction, the idea of our approach is to prove Theorem 4.1 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 $Sys_1 \geq^{\mathsf{poly}}_{\mathsf{sec}} Sys_2$ (in the computational case). The main result of [25] is therefore

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

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 [72] and (1) imply

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

Showing the theorem's preconditions is easy since the machines $\mathsf{M}_u^{\mathsf{NS}}$ are polynomial-time (see Section 3.4). Finally, the integrity preservation theorem from [12] 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 $Sys^{\mathsf{NS,real}} \models^{\mathsf{poly}} Req^{\mathsf{EA}}$.

6 Proof in the Ideal Setting

This section contains the proof of the ideal part of Theorem 4.1: We prove that the Needham-Schroeder-Lowe protocol implemented with the ideal, Dolev-Yao-style cryptographic library perfectly fulfils 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 this section with a rigorous definition of the possible states of the ideal cryptographic library as needed for formulating the invariants. We then define the invariants and prove the overall entity authentication requirement from the invariants. Finally we prove the invariants, after describing the detailed state transitions of the ideal cryptographic library as needed in that proof.

6.1 Overview and States of the Ideal Cryptographic Library

The ideal cryptographic library administrates Dolev-Yao-style terms and allows each user to operate on them via handles, i.e., via local names specific to this user. The handles also contain the information that knowledge sets give in other Dolev-Yao formalizations: The set of terms that a participant u knows, including $u = \mathsf{a}$ for the adversary, is the set of terms with a handle for u. As we saw in the Needham-Schroeder-Lowe algorithms, the library offers its user (and the adversary) the typical operations on terms to which they have handles, e.g., encryption with a public key and decryption with a secret key. The terms are typed; for instance, decryption only succeeds on ciphertexts and projection only on lists. As secure encryption schemes are necessarily probabilistic, and as the library allows the generation of polynomially many nonces and key pairs, multiple instances of terms of almost every structure can occur, e.g., multiple encryptions of the same message m with the same key pke. There are multiple ways to deal with this in prior Dolev-Yao models, e.g., counting (for nonces) and multisets. The version in [25] corresponds to counting: The terms are globally numbered by a so-called index. Each term is represented by its type (i.e., root node) and its first-level arguments, which can be indices of earlier terms. This enables easy distinction of, e.g., which of many nonces is encrypted in a larger term. These global indices are never visible at the user interface. The indices and the handles for each participant are generated by one counter each.

A novel aspect of this cryptographic library compared with prior Dolev-Yao models is that terms have an abstract length parameter, indicating the length of the corresponding real message. It is derived from a tuple L of length functions that denote how the length of a term depends on the length of its subterms. This is necessary because real encryption cannot entirely hide the length of cleartexts. Moreover, L contains bounds on the accepted message lengths and the number of accepted inputs at each port. All these bounds can be arbitrary, except that they must be polynomially bounded in a security parameter k. Formally, the number n of participants and the tuple L are parameters of the system $Sys^{\mathsf{cry,id}}$, but we omitted them for readability.

Similarly, n and a tuple L' should be parameters of our ideal Needham-Schroeder-Lowe system $Sys^{\mathsf{NS,id}}$, see Section 3.4. As the machines $\mathsf{M}_u^{\mathsf{NS}}$ of this system only make bounded-length inputs to the cryptographic library given n and L', the bounds in L can easily be chosen large enough so that all these inputs are legal. Further, as we only prove an integrity property, it is not a problem in the proof that the number of accepted inputs might be exceeded. This is why we can omit the details of the length functions.

As described above, the terms in the ideal cryptographic library, i.e., in the trusted host $\mathsf{TH}_{\mathcal{H}}$ for every set $\mathcal{H}$ of honest participants, are represented by their top level, and knowledge of them by potential handles for the different participants. The data structure chosen for this in [25] is a database D. Generally, 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 $D :\Leftarrow x$. Further, we write the list operation as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$.

In our case, each entry x in D can have the arguments

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

where $\mathcal{H} = \{u_1, \ldots, u_m\}$ and the arguments have the following types and meaning:

- $x.ind$ is the global index of an entry. Its type $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$ and distinguishes index arguments from others. The index is used as a primary key attribute of the database, i.e., we write $D[i]$ for the selection $D[ind = i]$.

- $x.type \in typeset$ identifies the *type* of x. The types nonce, list, data (for payload data), ske and pke (for secret and public encryption keys), and enc (for encryptions) occur in the following.

- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Arguments of type $\mathcal{INDS}$ are indices of other entries (subterms); we sometimes distinguish them by a superscript "ind".

- $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ are handles, where $x.hnd_u = \downarrow$ means that u does not know this entry and $\mathcal{HNDS}$ is another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.

- $x.len \in \mathbb{N}_0$ denotes the length of the entry.

The machine $\mathsf{TH}_{\mathcal{H}}$ has a counter $size \in \mathcal{INDS}$ for the current size of D and counters $curhnd_u$ (current handle) for the handles, all initialized with 0.

The assumption that keys have already been generated and distributed (Section 3.2) means that for each user $u \in \mathcal{H}$ two entries of the following form are added to D:

$$\begin{aligned}
&(ske_u, type := \mathsf{ske}, arg := (), hnd_u := ske_u^{\mathsf{hnd}}, len := 0);\\
&(pke_u, type := \mathsf{pke}, arg := (), hnd_{u_1} := pke_{u,u_1}^{\mathsf{hnd}}, \ldots,\\
&hnd_{u_m} := pke_{u,u_m}^{\mathsf{hnd}}, hnd_{\mathsf{a}} := pke_{u,\mathsf{a}}^{\mathsf{hnd}}, len := \mathsf{pke_len}^*(k)).
\end{aligned}$$

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 [25] and will not matter here.

6.2 Invariants

This section contains invariants of the system $Sys^{\mathsf{NS,id}}$, which are used in the proof of Theorem 4.1. 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 6.1 *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 6.5.

6.3 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 [25] will be useful in this proof to show that properties proven for one time also hold at another time.

Lemma 6.2 *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 4.1) 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

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

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 $\mathsf{out}_u?$ 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 6.2 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 $x_{1,u}^{2\ \mathsf{hnd}} \in \mathit{Nonce}_{u,w}$. Thus *correct nonce owner* implies

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

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

$$D[x_2^{2^{\mathsf{ind}}}].\mathit{hnd}_v \in \mathit{Nonce}_{v,u} \text{ at time } t_4. \tag{8}$$

■

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 6.2 we therefore obtain

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

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 $\mathsf{out}_v?$ 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 $l^{1^{\mathsf{ind}}} := D[\mathit{hnd}_v = l_v^{1\ \mathsf{hnd}}].\mathit{ind}$, and obtain $D[l^{1^{\mathsf{ind}}}].\mathit{type} = \mathsf{list}$. Let $x_i^{1^{\mathsf{ind}}} := D[l^{1^{\mathsf{ind}}}].\mathit{arg}[i]$ 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 $\mathsf{list\_proj}$ and Lemma 6.2 yield

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

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

$$D[x_1^{1^{\mathsf{ind}}}].\mathit{hnd}_u \in \mathit{Nonce}_{u,v} \text{ at time } t_4. \tag{11}$$

The test in Step 2.6 ensures that $x_2^1 = w' \neq \downarrow$. This implies $D[x_2^{1^{\mathsf{ind}}}].\mathit{type} = \mathsf{data}$ by the definition of $\mathsf{retrieve}$, and therefore with Lemma 6.2,

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

■

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 $\mathsf{list\_proj}$ implies $D[x_1^{1^{\mathsf{ind}}}].\mathit{hnd}_u = n_u^{\mathsf{hnd}} \in \mathit{Nonce}_{u,w''}$. 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 4.1. ∎

6.4 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. By $x := y$++ for integer variables x, y we mean $y := y + 1; x := y$. The length of a message m is denoted as $\mathsf{len}(m)$.

Each input c at a port $\mathsf{in}_u?$ with $u \in \mathcal{H} \cup \{\mathsf{a}\}$ should be a list $(\mathit{cmd}, x_1, \ldots, x_j)$ with cmd from a fixed list of commands and certain parameter domains. We usually write it $y \leftarrow \mathit{cmd}(x_1, \ldots, x_j)$ with a variable y designating the result that $\mathsf{TH}_{\mathcal{H}}$ returns at $\mathsf{out}_u!$. The 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 $i^{\mathsf{hnd}} := D[i].\mathit{hnd}_u \neq \downarrow$, it returns that, else it sets and returns $i^{\mathsf{hnd}} := D[i].\mathit{hnd}_u := \mathit{curhnd}_u$++. 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 $i := D[\mathit{type} = \mathsf{data} \wedge \mathit{arg} = (m)].\mathit{ind} \neq \downarrow$ 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 $m^{\mathsf{hnd}} := \mathit{curhnd}_u$++ and

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

- *Retrieval:* $m \leftarrow \mathsf{retrieve}(m^{\mathsf{hnd}})$.

  $m := D[\mathit{hnd}_u = m^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{data}].\mathit{arg}[1]$.

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

$$\begin{aligned} D \;\; :\Leftarrow \;\; & (\mathit{ind} := \mathit{size}\text{++}, \mathit{type} := \mathsf{list}, \mathit{arg} := (x_1, \ldots, \\ & x_j), \mathit{hnd}_u := l^{\mathsf{hnd}}, \mathit{len} := \mathit{length}). \end{aligned}$$

- *i-th projection:* $x^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$, for $1 \le i \le \mathsf{max\_len}(k)$.

  If $D[\mathit{hnd}_u = l^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{list}].\mathit{arg} = (x_1, \ldots, x_j)$ with $j \ge i$, then $x^{\mathsf{hnd}} := \mathsf{ind2hnd}_u(x_i)$, otherwise $x^{\mathsf{hnd}} := \downarrow$.

The abstract command to create a fresh nonce simply creates a new entry in D.

- *Generate a nonce:* $n^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.

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

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

Further, we used commands to encrypt and decrypt a list.

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

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

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

- *Decryption:* $l^{\mathsf{hnd}} \leftarrow \mathsf{decrypt}(\mathit{sk}^{\mathsf{hnd}}, c^{\mathsf{hnd}})$.

  Let $\mathit{sk} := D[\mathit{hnd}_u = \mathit{sk}^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{ske}].\mathit{ind}$ and $c := D[\mathit{hnd}_u = c^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{enc}].\mathit{ind}$. Return $\downarrow$ if $c = \downarrow$ or $\mathit{sk} = \downarrow$ or $\mathit{pk} := D[c].\mathit{arg}[1] \ne \mathit{sk} + 1$ or $l := D[c].\mathit{arg}[2] = \downarrow$. 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 $\mathsf{in_a}?$, we only describe the command $\mathsf{adv\_parse}$. 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:* $(\mathit{type}, \mathit{arg}) \leftarrow \mathsf{adv\_parse}(m^{\mathsf{hnd}})$.

  Let $m := D[\mathit{hnd}_{\mathsf{a}} = m^{\mathsf{hnd}}].\mathit{ind}$ and $\mathit{type} := D[m].\mathit{type}$. In most cases, set $\mathit{arg} := \mathsf{ind2hnd}^*_{\mathsf{a}}(D[m].\mathit{arg})$. (Recall that this only transforms arguments in $\mathcal{INDS}$.) The only exception is for $\mathit{type} = \mathsf{enc}$ and $D[m].\mathit{arg}$ of the form (pk, l) (a valid ciphertext) and $D[\mathit{pk} - 1].\mathit{hnd}_{\mathsf{a}} = \downarrow$ (the adversary does not know the secret key); then $\mathit{arg} := (\mathsf{ind2hnd_a}(\mathit{pk}), D[l].\mathit{len})$.

We finally describe the commands for sending messages on insecure channels. In the second one, the adversary sends list l to v, pretending to be u.

- $\mathsf{send\_i}(v, l^{\mathsf{hnd}})$, for $v \in \{1, \ldots, n\}$ at port $\mathsf{in}_u?$ for $u \in \mathcal{H}$.

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

- $\mathsf{adv\_send\_i}(u, v, l^{\mathsf{hnd}})$, for $u \in \{1, \ldots, n\}$ and $v \in \mathcal{H}$ at port $\mathsf{in_a}?$.

  Let $l^{\mathsf{ind}} := D[\mathit{hnd}_{\mathsf{a}} = l^{\mathsf{hnd}} \wedge \mathit{type} = \mathsf{list}].\mathit{ind}$. If $l^{\mathsf{ind}} \ne \downarrow$, output $(u, v, \mathsf{i}, \mathsf{ind2hnd}_v(l^{\mathsf{ind}}))$ at $\mathsf{out}_v!$.

6.5 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 $\mathsf{M}_u^{\mathsf{NS}}$ 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 $\mathsf{M}_u^{\mathsf{NS}}$ 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 6.2 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 6.2.

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 [25]. 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 $\mathsf{gen\_nonce}$ was input at $\mathsf{in}_u?$ at time t. The definition of $\mathsf{gen\_nonce}$ 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 $\mathsf{list\_proj}$ or $\mathsf{adv\_parse}$ input at $\mathsf{in}_w?$, where $\mathsf{adv\_parse}$ 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 6.2. 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 $\mathsf{list\_proj}$, $\mathsf{decrypt}$, $\mathsf{adv\_parse}$, $\mathsf{send\_i}$, and $\mathsf{adv\_send\_i}$ 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 $\mathsf{list\_proj}$, $\mathsf{send\_i}$, and $\mathsf{adv\_send\_i}$ 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 [25]. Hence the command $\mathsf{decrypt}$ does not violate the invariant. Finally, the command $\mathsf{adv\_parse}$ 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 6.2 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 $\mathsf{encrypt}$ (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. ■

7 Conclusion

We have shown that the Needham-Schroeder-Lowe public-key protocol is secure in the real cryptographic setting. This was done via a proof over a Dolev-Yao-style deterministic idealization of cryptography which has a provably secure real cryptographic implementation. Composition and integrity preservation theorems from the underlying model imply that the protocol proof with the idealized cryptography carries over to the real protocol implementation. This was the first example of such a proof. In spite of certain differences to usual Dolev-Yao variants, in particular a representation of terms or real cryptographic objects to the protocol layer by handles (local names) and length functions in the idealization, the proof seems to be of a type readily accessible to automatic proof tools. We therefore hope that our hand-made proof paves the way towards automated, cryptographically sound proofs of the Needham-Schroeder-Lowe protocol and many other security protocols. In fact, we have executed our proof technical to obtain computational sound guarantees for further protocols [21, 7, 23, 15] and for more comprehensive security guarantees [6].

Acknowledgements

We thank Michael Waidner and the anonymous reviewers for many helpful 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. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[5] M. Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. *Journal of Logic and Algebraic Programming (JLAP)*, 2:157–188, 2005.

[6] M. Backes. Real-or-random key secrecy of the Otway-Rees protocol via a symbolic security proof. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 155:111–145, 2006.

[7] M. Backes and 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)*, pages 78–93, 2005.

[8] M. Backes, M. Duermuth, D. Hofheinz, and R. Kuesters. Conditional reactive simulatability. In *Proceedings of 11th European Symposium on Research in Computer Security (ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2006. Preprint on IACR ePrint 2006/132.

[9] M. Backes and T. Gross. Tailoring the dolev-yao abstraction to web services realities - a comprehensive wish list. In *Proceedings of 2005 ACM Secure Web Services Workshop (SWS)*, pages 65–74, 2005.

[10] M. Backes and D. Hofheinz. How to break and repair a universally composable signature functionality. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004. Preprint on IACR ePrint 2003/240.

[11] M. Backes, D. Hofheinz, J. Müller-Quade, and D. Unruh. On fairness in simulatability-based cryptographic systems. In *3rd ACM Workshop on Formal Methods in Security Engineering: From Specifications to Code*, pages 13–22, 2005. Preprint on IACR ePrint 2005/294.

[12] 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.

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

[14] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS)*, pages 370–379, 2006.

[15] M. Backes, S. Moedersheim, B. Pfitzmann, and L. Vigano. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging Scenario. In *Proceedings of Foundations of Software Science and Computational Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2006.

[16] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proceedings of 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. Preprint on IACR ePrint 2003/121.

[17] M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proceedings of 24th IEEE Symposium on Security and Privacy*, pages 140–152, 2003.

[18] M. Backes and B. Pfitzmann. Computational probabilistic non-interference. *International Journal of Information Security (IJIS)*, 3(1):42–60, 2004.

[19] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas of Computing (JSAC)*, 22(10):2075–2086, 2004.

[20] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Proceedings of 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004. Preprint on IACR ePrint 2004/059.

[21] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.

[22] M. Backes and B. Pfitzmann. Relating cryptographic und symbolic secrecy. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(2):109–123, 2005.

[23] M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *Proceedings of 21st IFIP International Information Security Conference (SEC)*, pages 233–245, 2006.

[24] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial liveness. *Journal of Computer Security*, 12(3-4):589–617, 2004.

[25] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003.

[26] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, 2003:15, 2003.

[27] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive system. In *Proceedings of 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

[28] M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2004.

[29] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. *IACR Cryptology ePrint Archive*, 2004:82, 2004.

[30] M. Backes, B. Pfitzmann, and M. Waidner. Reactively secure signature schemes. *International Journal of Information Security (IJIS)*, 4(4):242–252, 2005.

[31] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security (IJIS)*, 4(3):135–154, 2005.

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

[33] 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.

[34] 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.

[35] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.

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

[37] 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. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[38] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.

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

[40] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition (extended abstract). In *Proc. 1st ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 11–23, 2003.

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

[42] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[43] 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.

[44] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.

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

[46] J. Herzog. *Computational Soundness of Formal Adversaries*. PhD thesis, MIT, 2002.

[47] J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

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

[49] D. M. Johnson and F. Javier Thayer. Security and the composition of machines. In *Proc. 1st IEEE Computer Security Foundations Workshop (CSFW)*, pages 72–89, 1988.

[50] R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.

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

[52] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[53] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–135, 1995.

[54] 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.

[55] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 18–30, 1997.

[56] H. Mantel. On the composition of secure systems. In *Proc. 23rd IEEE Symposium on Security & Privacy*, pages 88–101, 2002.

[57] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. 8th IEEE Symposium on Security & Privacy*, pages 161–166, 1987.

[58] D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

[59] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. 15th IEEE Symposium on Security & Privacy*, pages 79–93, 1994.

[60] J. McLean. A general theory of composition for a class of "possibilistic" security properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

[61] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[62] 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.

[63] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[64] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.

[65] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[66] J. K. Millen. The Interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.

[67] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

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

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

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

[71] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

[72] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, `http://eprint.iacr.org/`.

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

[74] S. Schneider. Verifying authentication protocols with CSP. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 3–17, 1997.

[75] P. Syverson. A new look at an old protocol. *Operation Systems Review*, 30(3):1–4, 1996.

[76] 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.

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

[78] A. C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.