EINDHOVEN UNIVERSITY OF TECHNOLOGY

Generalizing Needham-Schroeder-Lowe for multi-party authentication

Citation for published version (APA):
Cremers, C. J. F., & Mauw, S. (2006). *Generalizing Needham-Schroeder-Lowe for multi-party authentication*. (Computer science reports; Vol. 0604). Technische Universiteit Eindhoven.

Document status and date:
Published: 01/01/2006

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 16. Nov. 2023

Generalizing Needham-Schroeder-Lowe for Multi-Party Authentication

C.J.F. Cremers S.Mauw

Eindhoven University of Technology,
Department of Mathematics and Computer Science,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
{ccremers,sjouke}@win.tue.nl

Abstract

We propose a protocol for multi-party authentication for any number of parties, which generalizes the well-known Needham-Schroeder-Lowe protocol. We show that the protocol satisfies authentication of the communicating parties (by proving injective synchronisation) and secrecy of the generated challenges. For p *parties, the protocol consists of* $2p - 1$ *messages, which we show to be the minimal number of messages required to achieve the desired security properties in the presence of a Dolev-Yao style intruder. The underlying communication structure of the generalized protocol can serve as the backbone of a range of authentication protocols.*

1 Introduction

In the context of Dolev-Yao style modeling of security protocols, several protocols have been proposed in order to satisfy forms of *mutual authentication* (for an overview of authentication protocols see [9, 25]). Of these, the best known is Needham-Schroeder-Lowe (NSL) from [21,23]. The NSL protocol satisfies even the strongest forms of authentication [15], and has been studied extensively.

The operation of the three-message base protocol is as follows (see Figure 1). In the first step, the initiator of the protocol, executing the a role, creates a random value (often called a nonce or a challenge) na. He encrypts this value along with his name with the public key pkb of the intended responder. When the responder, executing the b role, receives such a message, he generates his own random value nb. He responds to the challenge by encrypting both nonces as well has his own name, with the public key of the initiator. In the third step, the initiator sends back the random value nb to the responder, encrypted by the public key of the responder.

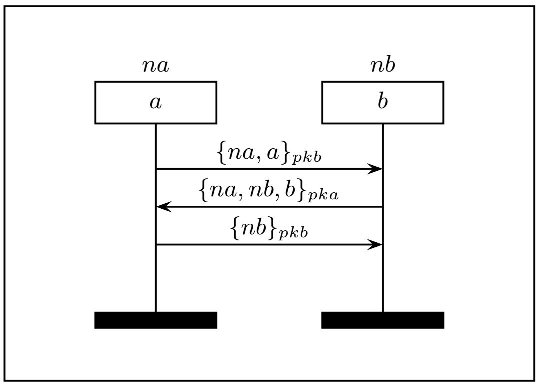
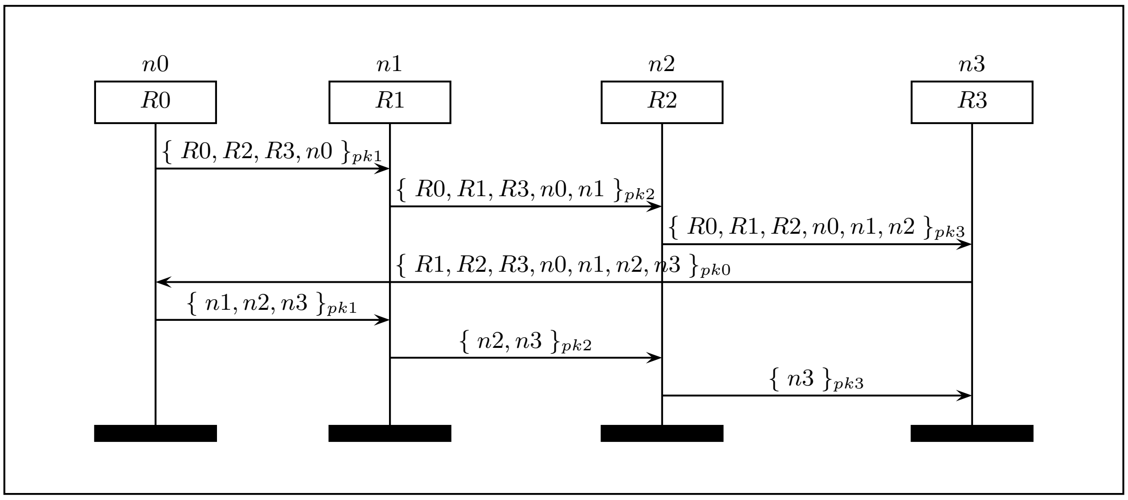


Figure 1. The Needham-Schroeder-Lowe protocol with public keys.

This protocol was designed for two parties who want to authenticate each other, which is often referred to as bilateral authentication. In many settings such as modern e-commerce protocols there are three or more parties that need to authenticate each other. In such a setting we could naively instantiate multiple NSL protocols to mutually authenticate all partners. For p parties, such mutual authentication would require $(p\times(p-1))/2$ instantiations of the protocol, and three times as many messages. In practice, when multi-party authentication protocols are needed, protocol designers instead opt to design new protocols (often 3 or 4-



$n0$
$n1$
$n2$
$n3$
$R0$
$R1$
$R2$
$R3$
$\{\ R0, R2, R3, n0\ \}_{pk1}$
$\{\ R0, R1, R3, n0, n1\ \}_{pk2}$
$\{\ R0, R1, R2, n0, n1, n2\ \}_{pk3}$
$\{\ R1, R2, R3, n0, n1, n2, n3\ \}_{pk0}$
$\{\ n1, n2, n3\ \}_{pk1}$
$\{\ n2, n3\ \}_{pk2}$
$\{\ n3\ \}_{pk3}$

Figure 2. Four-party generalized NSL.

way handshakes, see e.g. [18]), which possibly introduces new faults.

The goal of the current paper is to improve upon this approach and to generalize the NSL protocol as to obtain a multi-party authentication protocol with optimal message complexity (which turns out to be $2p - 1$ for p parties).

Since the development of correct security protocols has proven to be a notoriously difficult task we use the framework introduced in [14–16] to prove the proposed protocol correct.

Although many methodologies and tools for verifying security protocols in a Dolev-Yao setting have been developed, these have been focussed on protocols with a fixed number of parties. The challenge is in proving the proposed parameterized protocol correct.

We proceed as follows. In Section 2 we generalize the Needham-Schroeder-Lowe protocol for any number of parties. In Section 3 we show the security properties that the protocol satisfies and sketch proofs of correctness and preconditions. We discuss some variations of the pattern of the generalized protocol in Section 4. Related work is discussed in Section 5. We draw conclusions and discuss further work in Section 6.

2 A multi-party authentication protocol

The basic idea behind the NSL protocol is that each agent has a challenge-response cycle to validate the other agent's identity. These two challenge-response cycles are linked together by identifying the response of the second agent with its challenge.

Its generalization follows the same line of thinking. Every agent conducts a challenge-response cycle with its neighbouring agent, while combining its own challenge with a response to another agent's challenge whenever possible.

We will first explain the four-party version of the protocol in some detail. Afterwards we give a generalized specification for p parties.

The four-party protocol goes as follows (see Figure 2). First, the initiating agent chooses which parties he wants to communicate with. He creates a new random value, $n0$, and combines this with his name and the names of agents $R2$ and $R3$. The resulting message is encrypted with the public key of $R1$, and sent to $R1$. Upon receipt and decryption of this message, the second agent adds his own name and a fresh nonce, and removes the name of the next agent in the line from the message. This modified message is then encrypted with the public key of the next agent and sent along. This continues until each agent has added his nonce, upon which the message is sent back to the initiating agent. This agent checks whether the message contains the nonce he created earlier, and whether all agent names match. Then he can conclude that the other agents are authenticated. Next, in order to prove his own identity, he sends a message containing the other agents' nonces to $R1$. The subsequent agents again check whether their own nonces are in the message, remove this nonce, and

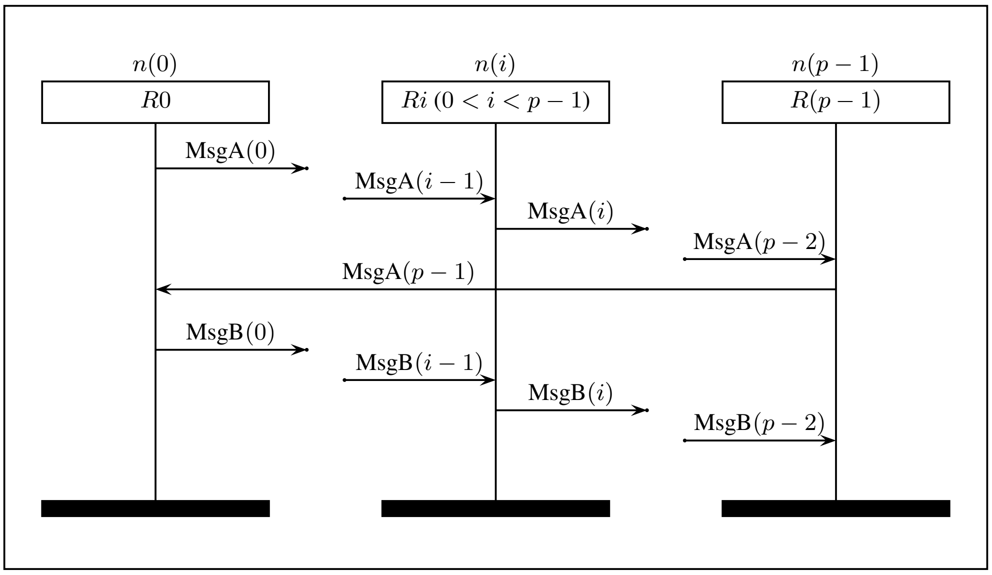


Figure 3. Generalized NSL pattern

pass the resulting message on.

This four-party protocol can be generalized to any number of agents p. In Figure 3 we schematically describe the communication structure of the protocol. The abstract messages are defined below. The function $next$ determines the next role in the list of participants in a cyclic way. The ordered list AL(x) contains all roles, except for role x. The protocol makes use of two types of messages. The first p messages are of type MsgA and the final $p-1$ messages are of type MsgB.

$$\begin{aligned} next(i) &= R((i+1) \mod p) \\ \text{AL}(x) &= [R0, R1, \ldots, R(p-1)] \setminus x \\ \text{MsgA}(i) &= \{\ [n(0)\ldots n(i)], \text{AL}(next(i))\ \}_{pk(next(i))} \\ \text{MsgB}(i) &= \{\ [n(i+1)\ldots n(p-1)]\ \}_{pk(next(i))} \end{aligned}$$

For i such that $0 \leq i < 2p-1$, protocol message labeled with (p,i) is now defined by

$$\text{Msg}(i) = \begin{cases} \text{MsgA}(i) & \text{if } 0 \leq i < p, \\ \text{MsgB}(i-p) & \text{if } p \leq i < 2p-1. \end{cases}$$

The purpose of the protocol is to achieve authentication of all parties and secrecy of all nonces, in the presence of a Dolev-Yao intruder, that has full control over the network. We will make this precise in the next section, but first we make two observations. First, agent Rx reads messages with labels $(p, x-1)$ and $(p, x+p-1)$, and sends out messages with labels (p,x) and $(p, x+p)$. Second, a nonce created by agent Rx occurs in $p-1$ messages: to be exact, it occurs in the messages labeled with (p,i), where $x \leq i < x+p$.

The protocol can be deployed in two main ways. First, it can be instantiated for a specific number of parties, to yield e.g. a four-party authentication protocol. In this way it can be used instead of custom n-way handshake protocols.

Second, it can be used in its most generic form, and have the initiating role $R0$ choose the number of participants p. Agents receiving messages can deduce the chosen p at runtime from the number of agents in the first message, and their supposed role from the number of nonces in the message. For the analysis in the next section we use the generic form, where the number of parties p is not fixed: the properties that we prove will then automatically hold for specific instantiations as well, where only a certain number of parties is allowed.

3 Analysis

In this section we will prove that the protocol satisfies *injective synchronisation*. Informally, for a two-party protocol, it states the following:

> Initiator I considers a protocol synchronising, whenever I as initiator completes a run of the protocol with responder R, then R as responder has been running the protocol with I. Moreover, all messages are received exactly as they were sent, in the order as described by the pro-

tocol. Initiator I considers a protocol injectively synchronising if the protocol synchronizes and each run of I corresponds to a unique run of R.

This definition extends in a natural way to multi-party protocols. We refer the reader to [15, 16] for a detailed definition of injective synchronisation and a proof that the more common notion of *agreement* is implied by injective synchronisation.

Important to the approach here is that we consider *local synchronisation claims*. That means that an agent may decide that the complete protocol has been executed exactly as expected, based on his local observations only. These observations only take into account the contents of the communications that the agent was involved in. Whenever such an agent successfully completes a run of a synchronising protocol, all other parties involved in the protocol have executed their part exactly as expected.

We notice that the proposed multi-party authentication protocol performs an *all or none* authentication. This means that whenever an agent finishes his part of the protocol successfully, he will be sure that all other parties are authenticated to him. On the other hand, if any of the communication partners is not able to authenticate himself, the protocol does not terminate successfully. So, this protocol does not establish authentication of a subset of the selected agents.

A second observation concerning this protocol is the fact that authentication is only guaranteed if all agents indicated in the first message of the initiator are *trusted*. This means that if the decryption key of any of the agents is compromised, the other agents in the list can be falsely authenticated. The reason is that e.g. agent $R0$ only verifies the authenticity of agent $R1$. Verification of the identity of agent $R2$ is delegated to agent $R1$, and so on. This chain of trust is essential to the design of an efficient multi-party authentication protocol.

Finally we want to mention that the proof does not only imply correctness for any specific choice of p: rather, we prove that the protocol is correct when p is chosen at run-time by the initiator. Put differently, we prove correctness of the protocol in an environment with all p-party variants running in parallel.

3.1 Properties of generalized NSL

The multi-party authentication protocol described above has a number of interesting properties. It satisfies secrecy of each of the nonces, and injective synchronisation for all parties (and therefore also agreement). Furthermore, it uses a minimal number of messages to achieve these properties.

Correctness holds in the presence of an active Dolev-Yao intruder, thus assuming perfect cryptographic primitives, but with compromised/dishonest agents in the system. It is possible that agents start a protocol session with dishonest agents, in which case no security properties are guaranteed. We prove that even though honest agents sometimes communicate with dishonest agents, the communications among groups of honest agents remain secure.

3.2 Proof of correctness

We will show the proofs of the security claims of the generalized version of Needham-Schroeder-Lowe in some detail. For these proof descriptions we rely on the concepts introduced in [14–16]. Although it is our intention that this paper is as self-contained as possible, many subtleties are explained more clearly in the papers mentioned.

We briefly introduce some of the concepts involved. Black box modeling of security protocols starts from a term algebra with at least two constructors: tupling of terms t_1 and t_2, denoted as (t_1, t_2), and encryption of a term t with a term k, denoted as $\{ \ t \ \}_k$.

We assume a typed model, in other words we assume the messages are constructed in such a way that the recipient can distinguish e.g. a nonce from an agent name. This assumption is addressed in more detail in Section 3.3

The subterm relation $\sqsubseteq$ concerns all terms involved in the construction of a term. It therefore also contains the keys used in the construction.

Definition 1 *The subterm relation $\sqsubseteq$ is the least reflexive, transitive relation satisfying $t_1, t_2 \sqsubseteq (t_1, t_2)$ and $t_1, t_2 \sqsubseteq \{ \ t_1 \ \}_{t_2}$.*

We also define a subterm with an encryption depth parameter, that signals the number of encryptions applied to a term, using a subscript notation. Thus we still have $t \sqsubseteq_0 t$. We have $t_1 \sqsubseteq_0 (t_1, t_2)$ and $t_1 \sqsubseteq_1 \{ \ t_1 \ \}_{t_2}$. Note that we have $t_2 \sqsubseteq_0 \{ \ t1 \ \}_{t2}$ because the key is not encrypted itself. If a term occurs at more levels of encryption, we refer to the lowest level.

We consider an unbounded number of agents, each executing one or more instances of one or more roles of the protocol in parallel. An instantiated role is called a run. Agents communicate via a network that is under complete control of the (Dolev-Yao) intruder. The operational semantics describes all possible behaviours of such a system. This gives rise to a set of traces: each trace is a sequence of (instantiated) send and read events. A send (or read) event executed in run rx is denoted by $\mathit{send}\sharp rx$ (or $\mathit{read}\sharp rx$). So, an execution trace consists of a possible interleaving of the send and read events of the runs. In this model, the network is equated with the intruder. Therefore a send event implies that the intruder learns the contents of the sent message, and possibly decomposes it using available decryption keys. A read event implies that the intruder was able to construct the contained message. The intruder knowledge models the collection of messages that the intruder is able to construct.

Definition 2 *The initial knowledge of the intruder is denoted as M_0. Given a trace α and an index i, the knowledge of the intruder before an event α_i is denoted as:*

$$M(\alpha, i) = M_0 \cup \{m \mid \exists j < i : \alpha_j = \mathit{send}(m)\}$$

The intruder knowledge is closed with respect to the available operators, such as tupling/projection and encryption/decryption. We assume perfect encryption: an encrypted term can only be decrypted if the intruder is in possession of the key. Thus we have $\{\ x\ \}_k \in M \wedge k^{-1} \in M \Rightarrow x \in M$.

Fresh values (nonces) created in runs are not known to the intruder initially. Even if the intruder learns them at some point, there is a point before which the nonce was not known.

Lemma 1 *Given a trace α, and a nonce n that was created by a run rx in role y, we have that:*

$$\exists j : n \in M(\alpha, j) \;\Rightarrow\; \exists i : n \notin M(\alpha, i) \;\wedge\; n \in M(\alpha, i+1)$$

If a fresh value is not known to the intruder, he cannot construct terms that have this fresh value as a direct subterm himself. Thus, these messages must have been learned somewhere before.

Lemma 2 *Assume a protocol model with explicit type checking for terms. Given a trace α, terms t, m, k and an index i:*

$$(t \sqsubseteq_0 m \wedge t \notin M(\alpha, i) \wedge \alpha_i = \mathit{read}(\{\ m\ \}_k) \Rightarrow \exists (j < i : m \sqsubseteq m' \wedge \alpha_j = \mathit{send}(m'))$$

The proof of this Lemma is not detailed here.

The following lemmas only hold for the specific protocol under investigation.

Based on Lemma 2 we can establish the following property:

Lemma 3 *Consider the generalized version of NSL Let α be a trace, and rx be a run executing role x, in which a nonce n was created. Let ry be a run and m a message such that $n \sqsubseteq_1 m$. If we have*

$$n \notin M(\alpha, i) \wedge \alpha_i = \mathit{send}(m)\sharp ry$$

then we have

$$\begin{aligned} & ry = rx \vee \\ & \quad (\exists m', j, j', rz : j < j' < i \;\wedge\; n \sqsubseteq_1 m' \wedge \\ & \quad\; \alpha_j = \mathit{send}(m')\sharp rz \;\wedge\; \alpha_{j'} = \mathit{read}(m')\sharp ry) \end{aligned}$$

Intuitively, the lemma states that if an agent sends out a nonce that the intruder does not know, it is either its own nonce or it is one that it learned before from the send of some other agent.

The previous lemma gives us a preceding run for a message that is sent. If we repeatedly apply this lemma on a similar situation where a nonce is received, we can establish a sequence of events that must have occurred before a nonce is received. We introduce the notation run(e) to denote the run that executes the trace event e.

Lemma 4 *Consider the generalized version of NSL. Let α be a trace, and rx be a run executing role x in which a nonce n was created. Let ry be a run and m be a message such that $n \sqsubseteq_1 m$. If we have*

$$n \notin M(\alpha, i) \wedge \alpha_i = \mathit{read}(m)\sharp ry$$

then there exists a non-empty finite sequence of events β such that the events in β are a subset of the events in α, and

$$\begin{aligned} & \mathit{run}(\beta_0) = rx \;\wedge\; \beta_{|\beta|-1} = \mathit{send}(m) \wedge \\ & \quad (\forall n : 0 \leq n < |\beta| : \exists m', j, r : n \sqsubseteq_1 m' \wedge \\ & \qquad\quad \beta_n = \alpha_j = \mathit{send}(m')\sharp r \wedge \\ & \quad (\exists rr, j' : j < j' < i : \alpha_{j'} = \mathit{read}(m')\sharp rr \wedge \\ & \qquad\quad (n < |\beta| - 1 \Rightarrow rr = \mathit{run}(\beta_{n+1})))) \end{aligned}$$

The resulting sequence of events is a chain of send events that include the nonce: each sent message is read by the run of the next send. This lemma is used to trace the path of the nonce from the creator of a nonce to a recipient of a message with the nonce, as long as the nonce is not known to the intruder. In other words, β represents a subset of send events of α through which the nonce n has passed before α_i.

Given a run identifier r, we denote the local mapping of roles to agents (which is used to denote the intended communication partners of a run) with $\rho(r)$. We use A_T to denote the set of trusted agents: these are the agents that are "honest", and thus not compromised by, or conspiring with, the intruder. Intuitively, we prove the following: even though not all agents in the system can be trusted, we still want the communications between trusted agents to be secure. For a run r that wants to communicate with trusted agents we have that $\rho(x) \subseteq A_T$, whereas for a run $r2$ that sets up a communication with an untrusted agent we have that $\rho(r2) \not\subseteq A_T$. We use this terminology in the next lemma.

Lemma 5 *For the generalized version of NSL, given a trace α, and a nonce n that was created by a run rx, and a trace index k:*

$$\begin{aligned}&\rho(rx) \subseteq A_T \ \wedge\ n \in M(\alpha,k) \Rightarrow\\ &\quad (\exists ry, j, m : \rho(ry) \not\subseteq A_T \ \wedge\ j < k \ \wedge\\ &\quad n \notin M(\alpha,j) \ \wedge\ n \in M(\alpha,j+1) \ \wedge\\ &\qquad\qquad n \sqsubseteq m \ \wedge\ \alpha_j = \mathit{send}(m)\sharp ry)\end{aligned}$$

Intuitively, this lemma expresses that the protocol only reveals any terms to the intruder in runs that communicate with an untrusted agent. The lemma follows from the notion of untrusted agents, Lemma 1 and Lemma 2.

We now return to the sequence of events β as established by Lemma 4. Given a send event e, the type of message is either A or B, and is denoted as mtype(e).

Lemma 6 *Given a sequence of run events β established by application of Lemma 4, we have that there exists a k, where $1 \leq k \leq |\beta|$ such that*

$$\begin{aligned}&(k < |\beta| \Rightarrow \mathit{role}(\beta_k) = R0) \ \wedge\\ &\qquad \forall n : 0 \leq n < |\beta| :\\ &\qquad\quad \mathit{mtype}(\beta_n) = \begin{cases} A & \mathit{if}\, n < k \\ B & \mathit{if}\, n \geq k \end{cases}\end{aligned}$$

This lemma follows from the protocol rules: When a run creates a nonce, it is sent out first as part of a message of type A. Messages of type A contain agent names, whereas messages of type B do not. The run that receives such a message, sends it out again within a type A message (containing agent names), unless it is executing role $R0$, in which case it sends out the nonce within a type B message, without agent names. After receiving a message without agent names (type B), runs only send out type B messages.

This allows us to draw some further conclusions: because the messages in the sequence β are received as they were sent, and because the messages before k include a list of agents, we deduce:

- The runs executing the events $\beta_0 \ldots \beta_k$ have the same parameter p (the number of agents in the messages plus one) and each run has the same agent lists ρ.
- Given the parameter p and the number of nonces in a message, we can uniquely determine the role of a message.

This leads to the following result:

Lemma 7 *Given a sequence β resulting from applying Lemma 4 for a nonce created in a run rx executing role x, and an index k resulting from Lemma 6 we have*

$$\begin{aligned}&(k < |\beta| \Rightarrow \mathit{role}(\beta_k) = R0) \ \wedge\\ &\quad \forall n : 0 \leq n < k : \rho(\mathit{run}(\beta_n)) = \rho(rx) \ \wedge\\ &\qquad\qquad\qquad \mathit{role}(\beta_n) = R(n+x)\end{aligned}$$

3.2.1 Secrecy of nonces created in role $R0$

Lemma 8 *Given a trace α, a nonce n created by a run rx in role $R0$, we have that*

$$\rho(rx) \subseteq A_T \Rightarrow \forall i : n \notin M(\alpha, i)$$

Proof by contradiction. We assume the generated nonce is leaked to the intruder, and establish a contradiction, from which we conclude the nonce cannot be leaked.

Assume that there exists a trace α in which a nonce n was generated in a run rx executing role $R0$, and that this run tries to communicate with trusted agents only. In other words, $\rho(rx) \subseteq A_T$. Assume the nonce is learned by the intruder at some point. We apply Lemma 5 to find an event α_j of a run ry where the nonce is first leaked. Thus we have $n \notin M(\alpha, j)$ and $n \in M(\alpha, j+1)$). Note that $rx \neq ry$: the nonce could not have been created in run ry because that would imply ry only communicates with trusted agents, contradicting the lemma. In fact, in all sub case we will arrive at a contradiction of the type $\rho(rx) = \rho(ry)$ combined with the communication with (un)trusted agents in the runs. We apply Lemmas 4 and 6 to yield a sequence of events β and an index k.

We split cases based on the type of message of the send event at α_j. We distinguish two cases, and show that both lead to a contradiction.

- *The message sent at α_j is of type A.* Thus we conclude $k = |\beta|$, and from Lemma 7 we have that $\rho(ry) = \rho(rx)$. Because rx communicates with trusted agents only, and ry does not, we arrive at a contradiction.

- *The message sent of α_j is of type B*, so it does not contain agent names. Because the nonce n was not created by the run ry, it must have been received before. If we look at the protocol definitions, we see that each send of message B is preceded by a read of a message containing one extra nonce: the one created by the run. Thus, there must be a read event α_{j2}, $j2 < j$, with message $\{ \ [ny, \ldots, n, \ldots] \ \}_{pk(\ldots)}$, where ny is the nonce created by run ry. Because this message again contains n, the intruder could not have created this message himself, and an agent must have sent it. If we look at the messages in the sequence β and the protocol rules, we find that there must exist an index $k2$, such that run$(\beta_{k2}) = ry$ (where ny was sent out first), and that $k2 < k$ on the basis of Lemma 6. If we combine this with Lemma 7, we arrive at $\rho(ry) = \rho(rx)$, which yields a contradiction.

3.2.2 Non-injective synchronisation of role $R0$

Given that the secrecy of nonces generated by role $R0$ holds, the following is straightforward:

Lemma 9 *Non-injective synchronisation holds for role $R0$.*

As stated before, a full definition of synchronisation is given in [15]. Briefly, it states that when an agent reaches the end of its role, there are other runs, that fulfil the other roles of the protocol, with which messages are exchanged exactly as prescribed by the protocol. We have to prove that all communications that precede the end of the role have occurred correctly: in this case, for role $R0$ we only have to prove that the communications of type A have occurred as expected, because from the viewpoint of role $R0$, we cannot be sure that any messages of type B ever arrive.

Here we sketch the proof, which is not difficult given the secrecy of the nonce: Given a trace α with a run rx executing role $R0$, we have that the nonce n generated by this role is secret on the basis of Lemma 8. Thus, if the agent completes his run, there must have been two indices j and i, $j < i$ such that $\alpha_j = send_{(p,0)}(m)$ and $\alpha_i = read_{(p,p-1)}(m')$. If we use Lemma 4 and Lemma 6 we find that the events in the sequence β are exactly the events that are required to exist for the synchronisation of the role $R0$: they are received exactly as they were sent, which is required for synchronisation. This leaves us with only one proof obligation: we have to show that the sequence β contains all the messages of type A, in the right order, after the start of run rx, and before the end of run rx. This follows directly from the role assignment in Lemma 7.

3.2.3 Secrecy of nonces created in role Rx for $x > 0$

Lemma 10 *Given a trace α, a nonce n created by a run rx executing role Rx with $x > 0$, we have that*

$$\rho(rx) \subseteq A_T \Rightarrow \forall i : n \notin M(\alpha, i)$$

Proof by contradiction, similar to that of Lemma 8. Assume the nonce n was created by a run rx executing role x, and is leaked by a run ry to the intruder in some trace α. We have $\rho(rx) \subseteq A_T$ and $\rho(ry) \not\subseteq A_T$. We use Lemma 4 and Lemma 6 to yield a sequence β and index k. We distinguish two cases:

- $k = |\beta|$: We derive $\rho(rx) = \rho(ry)$, leading to a contradiction.

- $k < |\beta|$: Because role$(\beta_k) = R0$, we use Lemma 9 to extend the sequence β backwards, by merging the sequence from the leaking of the nonce with the sequence from the synchronisation. From the messages in the initial segment, which now includes all roles, we find that $\rho(rx) = \rho(ry)$, leading to a contradiction.

3.2.4 Non-injective synchronisation of role Rx for $x > 0$

For non-injective synchronisation of role Rx for $x > 0$, we not only have to prove that all messages of type A have occurred as expected, but also all messages MsgB(x), and that there is so-called run-consistency: for each role Ry we want there to be a single run that sends and receives the actual messages.

Lemma 11 *Non-injective synchronisation holds for role Rx, where $x > 0$.*

Proof sketch: based on the secrecy of the nonce generated in such a role, we determine an index k, and sequence β that precedes the last read event of the role, with role$(\beta_0) = Rx$. Because the sequence must include an event of role $R0$, for which non-injective synchronisation holds, we merge the sequences for both (as in the previous lemma). This gives us a complete sequence of send and events which exactly meet the requirements for non-injective synchronisation: they are received exactly as they were sent. The requirement of the existence of all messages of type A follows from Lemma 7: thus there is a run for each role in the protocol. Furthermore, the nonce of each these runs is present in the message sent at α_k. If we examine the protocol rules, we see that the message of type B is only accepted by runs whose own nonce is contained in the message: therefore we have that the run executing role $R1$ must be equal to run(α_{k+1}), and that the read event must be labeled as the MsgB(0). Similarly, we establish that the correct messages have been consistently read and sent by the runs that created the nonces. Thus all conditions for non-injective synchronisation are met.

3.2.5 Injective synchronisation of all roles

The additional requirement of *injectivity* ensures that a security protocol is not vulnerable to a certain class of replay attacks. In [16] we formalised the notion of injectivity and proved that for synchronising protocols inspection of the protocol at a syntactic level suffices to conclude injectivity. This syntactic criterion, the *loop-property*, roughly states that there must be a sequence of messages from the claiming role to each of the other roles, and back. For the proposed protocol this boils down to verifying that each role has a challenge-response loop to each of the other roles, which is obviously the case. Therefore, the synchronisation proof presented above implies injective synchronisation as well.

3.3 Observations

The Needham-Schroeder protocol If we instantiate the generalized protocol for $p = 2$, we get exactly the three message version of the NSL protocol. The NSL protocol was designed to fix a flaw in the Needham-Schroeder protocol, shown in Figure 4. If we compare our generalized version with the original Needham-Schroeder protocol, we see that the second message of the Needham-Schroeder protocol does not contain the agent list (AL(a)). Therefore we cannot conclude all the contradictions based on the matching agent lists, as we did in the proof of our protocol.

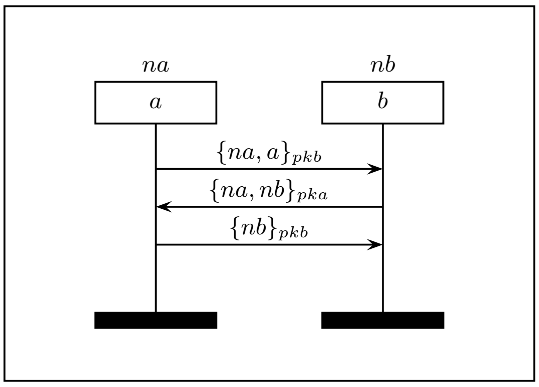


Figure 4. The Needham-Schroeder protocol with public keys.

Type-flaw attacks We have assumed that type-flaw attacks are not possible, i.e. agents can verify whether an incoming message is correctly typed. There are several reasons for doing this.

Without this assumption, there are type-flaw attacks on the generalized version of the protocol: not only simple ones for specific instances of p, but also multi-protocol type-flaw attacks involving instances for several choices of p in one attack, as in [13]. Thus, we find that typing is crucial. Solutions for preventing type flaw attacks using type information is examined in detail in e.g. [19]. Such type information can be easily added to each message, but a simple labeling will also suffice. If we add a tuple (p, l) before each message inside the encryption, where p is the number of participants for this instance, and l is the label of the message, the protocol becomes robust against type-flaw attacks and multi-protocol attacks with other instances of itself.

Using an automatic protocol verification tool (Scyther, [12]) we have established that the type-flaw attacks are not due to the specific ordering of the nonces and agent names within the messages. In particular, we examined different options for the message contents (without adding labels): reversing the order of either the agent or the nonce list, interleaving the lists, etc. We established the existence of type-flaw attacks for some choices of p for all variants we constructed.

3.4 Message minimality

As discussed in Section 3.2.5, the loop-property is instrumental to achieve injectivity. Moreover, in the context of a unicast communication model with a Dolev-Yao intruder, this loop-property turns out to be a necessity. Phrased in terms of challenge-response behaviour: in order to achieve injective synchronisation, each role must send a challenge that is replied to by all other roles.

From this requirement we can easily derive the minimal number of messages to achieve injective synchronisation. Consider the first message sent by some role Rx, and call this message m. In order to achieve a loop to all other roles after this first message, every role will have to send at least one message after m. Including message m this will yield at least p messages. Next we observe that every role must take part in the protocol and we consider the first message sent by each of the roles. If we take Rx to be the last of the p roles that becomes active in the protocol, it must be the case that before Rx sends his first message, at least $p-1$ messages have been sent. Adding this to the p messages that must have been sent after that message, yields a lower bound of $2p-1$ messages.

4 Variations on the pattern

The communication structure from Figure 3 can be instantiated in several different ways as to obtain authentication protocols satisfying different requirements. In this section we list some of the more interesting possibilities. Due to space limitations, we present the protocols without analysing their properties in detail.

Generalized Bilateral Key Exchange First, we observe that the nonces generated in the protocol are random and unknown to the adversary, which makes them suitable keys for symmetric encryption. Furthermore, if we examine the proofs, the authentication of the messages is only derived from the encryption of the messages of type A, not of type B. Similar to the Bilateral Key Exchange protocol (BKE) as described in [10] we can opt to replace the asymmetric encryption for the messages of type B by symmetric encryption with the nonce of the recipient. We can then omit this nonce from the list. We use ϵ to denote a constant representing the empty list. This yields the following message definitions.

$$\text{nlist}(i) = \begin{cases} [n(i+2)\dots n(p-1)] & \text{if } i < p-1 \\ \epsilon & \text{if } i = p-1 \end{cases}$$

$$\text{MsgA}(i) = \{\ [n0\dots ni], \text{AL}(next(i))\ \}_{pk(next(i))}$$

$$\text{MsgB}(i) = \{\ \text{nlist}(i)\ \}_{n(i+1)}$$

Using private keys If secrecy of the nonces is not required, we can use the private key of the sender of a message for encryption, instead of the public key of the receiver. This gives the following protocol.

$$\begin{aligned} \text{MsgA}(i) &= \{\ \text{AL}(Ri), [n0,\dots,ni]\ \}_{sk(Ri)} \\ \text{MsgB}(i) &= \{\ \text{AL}(Ri), [n(i+1),\dots,n(p-1)]\ \}_{sk(Ri)} \end{aligned}$$

Although this protocol is minimal in the number of messages, it is not minimal in the complexity of the messages. In the first message of role $R0$, e.g., we can take the role names outside the encryption operator. Although there are some other local optimizations, we prefer to present this more regular protocol. Finding such a protocol with minimal complexity of the messages is still an open question.

Rearranging message contents In the proofs of correctness, we have used some (but not all) information that distinguishes the messages in the protocol. In particular, we used:

- *The ordered agent list AL().* We used this to derive the parameter p from an incoming message, and the order in the list is required to be able to derive that the agent list of the sender is identical to the agent list of the recipient.
- *The list of nonces.* We used the number of nonces to derive the role an agent is supposed to assume (given p).

A direct consequence of this is that the exact order of the agent list and nonce list is not relevant, as long as it is consistent. We could e.g. redefine messages of type A as to start with a reversed list of roles, followed by the list of nonces.

Furthermore we did not require in the proof of the protocol, that there was nothing else inside the encrypted terms besides names and nonces. Thus, we can add any payload inside the encryption, as long as we ensure that it cannot be confused with an agent term or a nonce.

This opens up several possibilities for establishing e.g. keys between pairs of agents inside of the generalized NSL protocol. We discuss one such option in the next paragraphs.

Key agreement protocols In the field of cryptographic protocols many, so-called, group key agreement protocols have been developed. Although these have different goals from the protocol mentioned here, we see some possibilities to use the underlying structure of the protocol for these purposes.

The generalized NSL presented here can be turned into a naive group key agreement protocol by deriving a session key using a hash function over all the nonces, e.g. $h(n(0) \dots n(p-1))$. This would constitute a fresh authenticated session key, which is shared by all the participants. However, the resulting protocol would not satisfy e.g. *forward secrecy* of the session key: if one of the private keys of one of the participants is leaked after a session, the nonces that were used can be determined. From this the original session key can be retrieved, which allows an intruder to decrypt a session afterwards.

To establish forward secrecy of a session key, derivatives of the Diffie-Hellman key agreement protocol are used, as e.g. in [27]. We envisage that such an approach would be possible here as well: either by adding Diffie-Hellman derivatives as payload, or more efficiently, by replacing the nonces that are sent by the public halves of the Diffie-Hellman constructs.

5 Related Work

In Dolev-Yao style analysis of security protocols, where black-box abstractions of cryptographic operators are considered, protocols usually consist of two or three roles only. There are many recent successful methodologies [17,28] and analysis tools [3, 4, 11, 12, 24] that can be used to analyse protocols in the Dolev-Yao model. All the tools mentioned assume the protocols have a fixed number of participants. Therefore, they cannot be used to analyze multi-party protocols in general, but they can be used to analyze specific instances of such protocols. For example, Proverif [4] has been used to analyze instances of the GDH protocols from [2], and here we have used Scyther [12] to analyze instances of our protocol.

In spite of the success of these methods, few multi-party protocols have been constructed in the Dolev-Yao setting. As a notable exception we would like to mention [7], where the authors construct a challenge-response protocol for any number of parties. However, the protocol described there does not satisfy synchronisation or agreement.

On the other hand, many multi-party protocols have been constructed and analyzed in a cryptographic setting, e.g. [1, 2, 5]. These protocols are typically assumed to employ a multicast primitive, and based on this primitive their complexity can be analyzed, as in e.g. [20, 22]. Unfortunately the protocols in this category are designed to meet different goals than the protocol presented here, and therefore cannot be used to compare with our approach.

Recently, a corpus of multi-party protocols have been established as part of the Coral project [26], aiming to establish a reference set for multi-party protocol analysis.

Regarding proving authentication protocols correct, there have been some recent attempts to simplify such proofs. For example, one successful approach is to use static analysis of the protocol to prove authentication properties, as described in e.g. [6]. However, the notions of authentication used there are weaker than synchronisation or agreement, and the methods used there do not seem suitable for proving synchronisation.

Another approach to proving authenticity is taken in [8], where a simple logic is developed to prove authentication, assuming that some secrecy properties hold. The idea is then to delegate the proof of the secrecy properties to a dedicated secrecy logic. While a promising idea, in this case we have seen that the lemmas used to prove secrecy (i.e. yielding a sequence of send events) are also used to prove authenticity: thus, in this particular case, the proof structure seems to suggest that splitting the proof strictly into two (one for secrecy, one for authentication) leads to duplication of a large part of the proof. It would be interesting to see how a dedicated proof in this authentication logic would differ from our proof.

6 Conclusions and Future Work

We proposed a security protocol for multi-party authentication and proved it correct, i.e. the protocol satisfies injective synchronisation and all nonces are secret. The proof is formulated in terms of the operational semantics framework introduced in [14–16] and takes the number of parties p as a parameter. This is in line with more recent attempts (e.g. [26]) to develop methodologies for such (parameterised) multi-party protocols, for which this protocol could be used as a case study.

Correctness of the protocol is subject to the assumption that the messages include enough information as to allow a receiving agent to check if a message is correctly typed.

As has been shown by history, constructing correct security protocols is not trivial. Even knowing this, we were surprised to find that all variants of the proposed protocol (irrespective of the ordering of nonces and role names in the messages) suffer from type-flaw attacks. We found this out by using the Scyther tool [12]. In fact, we extensively used this tool to investigate instances of the protocol for a specific number of participants to guide us in our research and to study the variations presented in Section 4. A simple (and standard) extension of the messages will make the protocol resilient against such type-flaw attacks.

The communication structure underlying the protocol can serve as a generic pattern for multi-party challenge-response mechanisms, in which we can capture generalized NSL, BKE, and several other variants.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM Conference on Computer and Communications Security*, pages 17–26, 1998.

[2] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, 2000.

[3] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4:181–208, 2005.

[4] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules, 2001.

[5] E. Bresson, O. Chevassut, D. Pointcheval, and J.J. Quisquater. Provably authenticated group diffie-hellman key exchange. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264, New York, NY, USA, 2001. ACM Press.

[6] M. Bugliesi, R. Focardi, and M. Maffei. Authenticity by tagging and typing. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[7] L. Buttyn, A. Nagy, and I. Vajda. Efficient multi-party challenge-response protocols for entity authentication. 45(1):43–64, April 2001.

[8] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *CSFW*, pages 48–61, 2005.

[9] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature. Technical report.

[10] J.A. Clark and J.L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997.

[11] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In M. V. Hermenegildo and G. Puebla, editors, *9th Int. Static Analysis Symp. (SAS)*, volume LNCS 2477, pages 326–341, Madrid, Spain, Sep 2002. Springer-Verlag, Berlin.

[12] C.J.F. Cremers. Scyther: Automatic verification of security protocols.

[13] C.J.F. Cremers. Verification of multi-protocol attacks. Computer science report csr-05-10, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Mar 2005.

[14] C.J.F. Cremers and S. Mauw. Operational semantics of security protocols. In S. Leue and T. Systä, editors, *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl Castle, Germany, September 7-12, 2003, Revised Selected Papers*, volume 3466 of *LNCS*. Springer, 2005.

[15] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Defining authentication in a trace model. In Theo Dimitrakos and Fabio Martinelli,

editors, *FAST 2003*, pages 131–145, Pisa, September 2003. IITT-CNR technical report.

[16] C.J.F. Cremers, S. Mauw, and E.P. de Vink. A syntactic criterion for injectivity of authentication protocols. In P. Degano and L. Vigano, editors, *Arspa 2005*, volume 135(1) of *ENTCS*, pages 23–38, July 2005.

[17] A. Datta, A. Derek, J.C. Mitchell, and D. Pavlovic. Secure protocol composition. In *FMSE '03: Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, pages 11–23, New York, NY, USA, 2003. ACM Press.

[18] C. He and J.C. Mitchell. Analysis of the 802.11i 4-way handshake. In *WiSe '04: Proceedings of the 2004 ACM workshop on Wireless security*, pages 43–50, New York, NY, USA, 2004. ACM Press.

[19] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[20] J. Katz and M. Yung. Scalable protocols for authenticated group key exchange, 2003.

[21] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.

[22] D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. In Jan Camenisch and Christian Cachin, editors, *Advances in cryptology - EUROCRYPT 2004, proceedings of the internarional conference on the theory and application of cryptographic techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 153–170, Interlaken, Switzerland, May 2004. Springer-Verlag.

[23] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(2):120–126, February 1978.

[24] D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[25] Security protocols open repository (spore).

[26] G. Steel. Coral project: Group protocol corpus, 2004. `http://homepages.inf.ed.ac.uk/gsteel/group-protocol-corpus`.

[27] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-hellman key distribution extended to group communication. In *ACM Conference on Computer and Communications Security*, pages 31–37, 1996.

[28] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct? In *Proc. 1998 IEEE Symposium on Security and Privacy*, pages 66–77, Oakland, California, 1998.