The Needham-Schroeder-Lowe Protocol with Event-B

J.R. Abrial[1]

1 Introduction

This text contains the formal development (with proofs) of the Needham-Schroeder-Lowe Protocol [1] [2] using Event-B [3] and the Rodin Platform [4]. The Needham-Schroeder-Lowe Protocol is an authentication protocol. In other words, at the end of its execution the two agents involved in it (called respectively the initiator and the recipient) must be sure that they speak to each other.

As it is well known, there is a standard attack to this protocol as defined initially by Needham and Schroeder in [1]. By this attack, the authentication property is not guaranteed any more for the recipient of the protocol. This attack was discovered by Lowe in [2]. It involves, as usual, the presence of a, so-called, *attacker* having a bad behavior.

The purpose of the presentation of this example is to show how this attack of the protocol can be *simulated* without incorporating an attacker within the protocol, but rather by having one normal participant *doing a mistake*. More precisely, in this example, the initiator of an execution of the protocol is not following exactly what is defined in the protocol. The net result of this approach is to simplify the formal development.

Of course, what we describe here is just an example of such a simulation. At the moment, it is not clear whether any protocol attack involving an attacker can be simulated by means of participants doing mistakes.

2 Requirement Document

Before engaging in the formal development of this protocol in subsequent sections, it is advisable to define it informally by means of a number of environmental assumptions (labeled ENV) as well as functional requirements (labeled FUN).

2.1 General Requirements and Assumptions

This protocol involves some agents that are spread at different sites of a network

This system involves some agents situated on the sites of a network	ENV-1

An initiator and a recipient are concerned with a specific execution of the protocol.

An execution of the protocol involves two agents: the initiator and the recipient	ENV-2

[1] jrabrial@neuf.fr

The initiative of an execution of the protocol is in the hands of an initiator.

An initiator of the protocol wants to speak to a recipient	ENV-3

At the end of the execution of the protocol, when a recipient accepts to speak to an initiator, we want to be sure that indeed the initiator will speak to the agent it intends to speak to, and conversely we want to be sure that the recipient is indeed going to speak to the initiator that intended to speak to it initially. This behavior is called "mutual authentication".

The protocol must ensure *mutual authentication* between initiators and recipients	FUN-1

Since many protocol executions can happen simultaneously on the network, each agent uses, so-called, *nonces* to identify executions of the protocol. Each nonce, generated by an agent, is guaranteed to be unique over the network at all time.

Agents use *unique nonces* to identify specific executions of the protocol	ENV-4

As a consequence, an execution of the protocol is identified by a unique pair of nonces: one of these nonces is generated by the initiator, while the other is generated by the recipient.

A protocol execution is identified by a pair of nonces	ENV-5

Agents communicate with each other by means of encrypted messages.

Encrypted messages are used for the communication between agents	ENV-6

Encryption is ensured by means of public keys that are owned by each agent. More precisely, each agent A owns a pair of keys K_R and K_R^{-1}. The key K_R is *public* (known to every agents): it is used to encrypt messages sent to A. But the key K_R^{-1} is *secret* (known to A only): this key is used by A to decrypt messages sent to him encrypted with key K_R.

Message encryption is ensured by means of *public keys*	ENV-7

2.2 The Needham-Schroeder Protocol

So far, we described general assumptions (ENV_1 to ENV_7) together with the basic property (FUN_1) of an authentication protocol. Now, we are going to describe more specifically the Needham-Schroeder protocol.

The protocol is defined as follows: an initiator I sends an encrypted message (with the public key K_R owned by R) to a recipient R. The message contains I's name together with a new nonce N_I generated by I. The recipient R replies to this message by sending an encrypted message to I (encrypted with the public key K_I owned by I). This message contains the received nonce N_I sent by I and the new nonce

N_R generated by R. The initiator I replies to this message by sending an acknowledgment message to R (encrypted with the public key K_R). This message contains R's nonce N_R.

1. $I \rightarrow R : \{I, N_I\}_{K_R}$ 2. $R \rightarrow I : \{N_I, N_R\}_{K_I}$ 3. $I \rightarrow R : \{N_R\}_{K_R}$	FUN-2

This protocol seems to guarantee mutual authentication. More precisely, at step 2 the initiator I receives a message, $\{N_I, N_R\}_{K_I}$, containing *the nonce N_I that was sent previously by initiator I to recipient R only*. As a consequence, the message $\{N_I, N_R\}_{K_I}$ was certainly sent by the recipient R since nonces are unique according to assumptions ENV-4 (no other agent could have generated the nonce N_I). In fact, I is sure to have received this message $\{N_I, N_R\}_{K_I}$ from R because I sent to R the nonce N_I in a message encrypted with public key K_R ($\{I, N_I\}_{K_R}$) that *only R can decrypt*. A similar argument applies for the recipient R at step 3.

2.3 An Attack to the Needham-Schroeder Protocol

However, there is a well-known attack to this protocol. It was discovered by Lowe [2]. Here is the story: an initiator I sends a message (encrypted with the public key K_A owned by A) to a *recipient A that happens to be an attacker*. The attacker A decrypts this message and forward it to another recipient R using the public key K_R of R. The recipient R is misled, it believes to have received a message form I. R sends back a message to I as in the normal protocol. The initiator I believes to have received a reply from A. Therefore I sends to A the acknowledgment message. And now A decrypts this message and forward it (encrypted with key K_R) to R. Here is the more formal description of this attack:

1. $I \rightarrow A : \{I, N_I\}_{K_A}$
2. $A \rightarrow R : \{I, N_I\}_{K_R}$
3. $R \rightarrow I : \{N_I, N_R\}_{K_I}$
4. $I \rightarrow A : \{N_R\}_{K_A}$
5. $A \rightarrow R : \{N_R\}_{K_R}$

At the end of these message exchanges, the attacker A knows the nonce N_I (after step 1) and nonce N_R (after step 4) as well. The recipient R also knows these two nonces. Further messages can then be sent to R by A using the pair N_I-N_R as a justification. When R receives such a message *it believes that it comes from I* although it comes from A: the authentication is destroyed for R. For instance, if R is a bank, A could send the following message to R:

$$\{N\_I, N\_R, \text{"transfer some money into A's account"}\}_{K_R}$$

Now R, the banks, believes that this message comes from A and does the money transfer. Lowe provides a correction to the protocol in order to prevent the attack: the recipient must send its name in the normal step 2 of the protocol.

1. $I \rightarrow R : \{I, N_I\}_{K_R}$
2. $R \rightarrow I : \{N_I, N_R, R\}_{K_I}$
3. $I \rightarrow R : \{N_R\}_{K_R}$

When the initiator receives the second message it can check that the recipient sending this message is indeed the recipient with which he wanted to speak to initially.

In the present approach (as we explained in the Introduction), this attack is *simulated* by supposing that the Initiator I *makes a mistake*: rather than sending its initial message to the recipient R it intends to speak to, it sends this message to possibly another agent S (note that S and R can be identical but not necessarily). More precisely, in its first message, the initiator I uses public key K_S rather than K_R it should have used. In order to cope with this mistake of I, S should send its name to I. Then I will possibly discover its mistake by comparing S and R.

1. $I \rightarrow R \ : \ \{I, N_I\}_{K_S}$ 2. $S \rightarrow I \ : \ \{N_I, N_S, S\}_{K_I}$ if $S = R$ then 3. $I \rightarrow R \ : \ \{N_R\}_{K_R}$	FUN-3

As can be seen, step 3 will never happen in case S and R are not indentical: the protocol execution never terminates in this case.

3 Refinement Strategy

In what follows, we propose two distinct developments where we assume that there is no attacker within the network. First a development without mistake on the part of the Initiator (section 4), and then a development with a mistake on the part of the initiator (section 5).

In both cases, the developments contain an initial model where the agents do not send messages to each other. In fact, each agent is supposed to locally store some data corresponding to what happens at each phase of a protocol execution. We suppose then that each agent has the possibility to *directly access* what other agents have recorded. Of course, it is very dangerous. But, if we suppose that *the agents are obedient*, then nothing bad could happen and the authentication is ensured (and proved). In the subsequent refinements, we see how we can *enforce the obedience* of the agents by means of encrypted messages.

4 The Protocol Without Mistake

4.1 Initial Model

Sets, Contants, and Axioms

Here we define a set AGT of agents and a set NNC of nonces. In order to simplify the development, we suppose that the agents are splitt into two distinct sets: the Initiators and the Recipients

sets: AGT NNC	**constants:** $Initiator$ $Recipient$	**axm_1:** partition$(AGT, Initiator, Recipient)$

These elements take account of assumptions ENV_1 and ENV_2.

This system involves some agents situated on the sites of a network	ENV-1

An execution of the protocol involves two agents: the initiator and the recipient	ENV-2

Variables and Invariants

We define the variable set nnc corresponding to nonces that have been generated so far. As for the agents, we simplify matters by splitting this set into two incompatible sets: nni, the set of nonces generated by Initiators, and, nnr, the set of nonces generated by the Recipients.

variables: nnc nni nnr	**inv0_1:** $nnc \subseteq NNC$ **inv0_2:** $\mathsf{partition}(nnc, nni, nnr)$

These elements take partially account of assumption ENV_4.

Agents use *unique nonces* to identify specific executions of the protocol	ENV-4

Now, we define three variables $i1$, $i2$, and $i3$ that correspond to what Initiators are recording. The variable $i1$ is a total function from the set nni to the set $Initiator$. When a pair $ni \mapsto i$ belongs to $i1$, it means that the initiator i has started a protocol execution with nonce ni.

variables: $i1$	**inv0_3:** $i1 \in nni \rightarrow Initiator$

The variable $i2$ is a total function from the set nni to the set $Recipient$. When a pair $ni \mapsto r$ belongs to $i2$, it means that the initiator $i1(ni)$ has started a protocol execution in order to speak to recipient r.

variables: $i2$	**inv0_4:** $i2 \in nni \rightarrow Recipient$

The variable $i3$ is supposed to be a partial injection from the set nni to the set nnr: a pair $ni \mapsto nr$ belonging to $i3$ records the fact that (from the point of view of the initiator $i1(ni)$) there is a protocol execution represented by this pair. A given nonce can only be involved in one protocol execution, hence the injectivity of $i3$, which is only *partial* as an initiator cannot know initially the nonce generated by the recipient.

variables: $i3$	**inv0_5:** $i3 \in nni \rightarrowtail nnr$

Then, for the Recipients we have three variables $r1$, $r2$ and $r3$ that are similar to those of the Initiators seen previously. Here $r3$ is a *total* injection because the recipient knows the nonce of the initiator when it generates its own nonce.

variables: $r1$ $r2$ $r3$	**inv0_6:** $r1 \in nnr \rightarrow Recipient$ **inv0_7:** $r2 \in nnr \rightarrow Initiator$ **inv0_8:** $r3 \in nnr \rightarrowtail nni$

These elements take account of assumptions ENV_4 and ENV_5.

Agents use *unique nonces* to identify specific executions of the protocol	ENV-4

A protocol execution is identified by a pair of nonces	ENV-5

Events

Next are three events P1, P2, and P3, corresponding to the two first phases of the protocol as defined in requirement FUN_2. However, in this initial model, we do not send messages.

1. $I \rightarrow R$: $\{I, N_I\}_{K_R}$ 2. $R \rightarrow I$: $\{N_I, N_R\}_{K_I}$ 3. $I \rightarrow R$: $\{N_R\}_{K_R}$	FUN-2

Event P1 corresponds to step 1 of the protocol where an initiator elaborates and send a message to a recipient. Event P2 corresponds to a recipient elaborating and sending the message described in step 2. Finally, event P3 correspond to an initiator receiving the message sent in step 2. The last step of the protocol is not represented in this model. In the event P1, the Initiator i records its intention to speak to the recipient r by using the new nonce ni. The event $P1$ is supposed to be "executed" by a recipient i.

P1
any ni, i, r **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1 := i1 \cup \{ni \mapsto i\}$
$i2 := i2 \cup \{ni \mapsto r\}$
end

In the event P2, recipient r (that is $i2(ni)$) records that initiator i (that is $i1(ni)$) wants to speak to r by using nonce ni. This recipient r generates the new nonce nr (in $r1$) and connect nr to ni in $r3$. The event $P2$ is supposed to be "executed" by the recipient $i2(ni)$.

P2
any ni, i, r, nr **where**
$ni \in nni \setminus \mathrm{ran}(r3)$
$i = i1(ni)$
$r = i2(ni)$
$nr \notin ncc$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto i\}$
$r3 := r3 \cup \{nr \mapsto ni\}$
end

Finally, in the event P3, the initiator i (that is, $i1(ni)$) having generated the nonce ni records in $i3$ the nonce nr connected to ni in recipient r. Notice that $i1(ni)$ must be equal to $r2(nr)$. The event $P3$ is

supposed to be "executed" by the initiator $r2(nr)$

P3
any ni, nr **where**
$\underline{nr \mapsto ni \in r3}$
$ni \notin dom(i3)$
then
$i3 := i3 \cup \{ni \mapsto nr\}$
end

As can be seen, in this first model each participant of the protocol has free access to what the other has recorded. In the event P2, the recipient r (that is, $i2(ni)$) has access to the initiator $\underline{i1(ni)}$ stored in the site of the initiator i. Likewise, in the event P3, the initiator i (that is $i1(ni)$) has access to the nonce nr connected to ni in $r3$: $\underline{nr \mapsto ni \in r3}$.

Formalizing Mutual Authentication

Finally, we have now to see which additional invariants are necessary in order to ensure the authentication properties as defined in the requirement FUN_1:

The protocol must ensure *mutual authentication* between initiators and recipients	FUN-1

First, $i3$ and $r3$ are converse of each other (**inv0_9**) meaning that both participants use the same pair of nonces. We have no equality here because $i3$ is completed (in the event P3) after $r3$ (in the event P2). What this invariant states is that the initiator connected to nonce ri and the recipient connected to nr share the same nonces.

inv0_9: $i3^{-1} \subseteq r3$

Then the mutual authentication is expressed as follows:

1. If the pair $nr \mapsto ni$ belongs to $r3$ and the pair $nr \mapsto r$ belongs to $r1$ (that is, if $ni \mapsto r \in r3^{-1} ; r1$), then it means that the recipient r believes that he will speak to the initiator associated with nonce ni. So, we must be sure that the pair $ni \mapsto r$ belongs to $i2$. In other words, the recipient r must indeed be sure to speak to the initiator that wants to speak to him. This leads to the following invariant:

   inv0_10: $r3^{-1} ; r1 \subseteq i2$

   This invariant is easily maintained by our three events.

2. Conversely, if the pair $ni \mapsto nr$ belongs to $i3$ and the pair $ni \mapsto i$ belongs to $i1$ (that is, if $nr \mapsto i \in i3^{-1} ; i1$), then it means that the initiator i wants to speak to the recipient associated with nonce nr. So, we must be sure that the pair $nr \mapsto i$ belongs to $r2$. In other words, the initiator i must indeed be sure to speak to the recipient r that speak to him. This leads to the following:

   thm0_1: $i3^{-1} ; i1 \subseteq r2$

The statement is **thm0_1** in fact a *theorem* that is easily proved thanks to the following invariant (maintained by all events) and previous invariant **inv0_9** ($i3^{-1} \subseteq r3$):

inv0_11:	$r3 ; i1 = r2$

Proofs

The Proof Obligation Generator of the Rodin Platform produces 31 proof obligations (invariant and well-definedness), all proved automatically by the prover of the Rodin Platform.

4.2 Refinement

In this refinement, we introduce the encrypted messages. As a consequence, each participant does not use any more what is recorded in the other, it only uses the contents of the message it can decrypt.

Sets

We introduce here the set of possible messages.

sets:	MSG

Variables and Invariants

The variable msg is the set of messages circulating so far in the network. This set is partitioned into three sets $msg1$, $msg2$, and $msg3$. As all messages are encrypted, the variable $crypto$ records the agent owning the public key encrypting each message.

variables:	msg
	$msg1$
	$msg2$
	$msg3$
	$crypto$

inv1_1:	$msg \subseteq MSG$
inv1_2:	$\mathsf{partition}(msg, msg1, msg2, msg3)$
inv1_3:	$crypto \in msg \rightarrow AGT$

Variables $m1\_ini$ and $m1\_nni$ record the contents of messages belonging to $msg1$ as indicated in the description of the protocol:

1. $I \rightarrow R \ : \ \{I, N_I\}_{K_R}$ 2. $R \rightarrow I \ : \ \{N_I, N_R\}_{K_I}$ 3. $I \rightarrow R \ : \ \{N_R\}_{K_R}$	FUN-2

These message fields are the initiator (**inv1_4**) and its nonce (**inv1_5**). Notice that $m1\_nni$ is a bijection towards the set nni. The cryptographic key associated with messages of the set $msg1$ corresponds to the recipient recorded by the initiator (**inv1_6**). Invariant **inv1_6** will not be valid any more below (section

5) when the initiator will make a mistake. The initiator contained in a message belonging to $msg1$ is the initiator associated with the nonce contained in that message (**inv1_7**).

variables:	$m1\_ini$ $m1\_nni$

inv1_4:	$m1\_ini \in msg1 \rightarrow Initiator$
inv1_5:	$m1\_nni \in msg1 \rightarrowtail\!\!\!\!\!\rightarrow nni$
inv1_6:	$m1\_nni^{-1} ; crypto = i2$
inv1_7:	$m1\_nni^{-1} ; m1\_ini = i1$

Variables $m2\_nnr$ and $m2\_nni$ record the contents of messages belonging to $msg2$: the recipient nonce which is unique (**inv1_8**) and the initiator nonce (**inv1_9**).. The recipient nonce stored in the message is exactly the nonce associated with the initiator nonce in $r3$ (**inv1_10**).

variables:	$m2\_nni$ $m2\_nnr$

inv1_8:	$m2\_nnr \in msg2 \rightarrowtail\!\!\!\!\!\rightarrow nnr$
inv1_9:	$m2\_nni \in msg2 \rightarrow nni$
inv1_10:	$\forall m \cdot m \in msg2 \Rightarrow m2\_nnr(m) \mapsto m2\_nni(m) \in r3$

Finally, we have the contents of messages belonging to $msg3$. Such messages are not used here.

variables:	$m3\_nnr$

inv1_11:	$m3\_nnr \in msg3 \rightarrow nnr$

We take accounts of the encrypted messages, as described in ENV_6, and of the contents of these messages as described in FUN_2.

Events

In this section, we follow exactly the description of the protocol as decribed in FUN_2:

1. $I \rightarrow R$: $\{I, N_I\}_{K_R}$ 2. $R \rightarrow I$: $\{N_I, N_R\}_{K_I}$ 3. $I \rightarrow R$: $\{N_R\}_{K_R}$	FUN-2

Here are the concrete version of the events. In these events we take account of the encryption by public keys as described in assumptions ENV_6 and ENV_7:

Encrypted messages are used for the communication between agents	ENV-6

Message encryption is ensured by means of *public keys*	ENV-7

The event P1 is an extension of its abstraction (more guards and more actions). Besides recording some data as in the abstraction, it generates a new message belonging to $msg1$.

P1
any $ni, i, r, m1$ **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
$m1 \notin msg$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1 := i1 \cup \{ni \mapsto i\}$
$i2 := i2 \cup \{ni \mapsto r\}$
$msg := msg \cup \{m1\}$
$msg1 := msg1 \cup \{m1\}$
$crypto(m1) := crypto \cup \{m1 \mapsto r\}$
$m1\_nni := m1\_nni \cup \{m1 \mapsto ni\}$
$m1\_ini := m1\_ini \cup \{m1 \mapsto i\}$
end

(abstract-)P1
any ni, i, r **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1 := i1 \cup \{ni \mapsto i\}$
$i2 := i2 \cup \{ni \mapsto r\}$
end

The event P2 now uses the encryped message it receives in order to record data equivalent to what it was doing in the abstraction. It also generates a new message belonging to $msg2$.

P2
any $m1, r, nr, m2$ **where**
$m1 \in msg1$
$m1\_nni(m1) \notin \mathrm{ran}(r3)$
$nr \notin ncc$
$r = crypto(m1)$
$m2 \notin msg$
with
$ni = m1\_nni(m1)$
$i = m\_ini(m1)$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto m1\_ini(m1)\}$
$r3 := r3 \cup \{nr \mapsto m1\_nni(m1)\}$
$msg := msg \cup \{m2\}$
$msg2 := msg2 \cup \{m2\}$
$m2\_nni(m2) := m1\_nni(m1)$
$m2\_nnr(m2) := nr$
$crypto(m2) := m1\_ini(m1)$
end

(abstract-)P2
any ni, i, r, nr **where**
$ni \in nni \setminus \mathrm{ran}(r3)$
$i = i1(ni)$
$r = \overline{i2(ni)}$
$nr \notin ncc$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto i\}$
$r3 := r3 \cup \{nr \mapsto ni\}$
end

As can be seen, event r (that is $crypto(m1)$) who is able to decrypt the message $m1$ does not look at the intitiator state. All needed information are those given in the message $m1$.

The event P3 now uses the encrypted message. More precisely, initiator i (that is crypto(m1)) uses the nonce it recives in the message to update its state.

P3
any $m2$ **where**
$m2 \in msg2$
$m2\_nni(m2) \notin \mathrm{dom}(i3)$
$i1(m2\_nni(m2)) = crypto(m2)$
with
$ni = m2\_nni(m2)$
$nr = m2\_nnr(m2)$
then
$i3 := i3 \cup \{m2\_nni(m2) \mapsto m2\_nnr(m2)\}$
end

(abstract-)P3
any ni, nr **where**
$\underline{nr \mapsto ni \in r3}$
$ni \notin dom(i3)$
then
$i3 := i3 \cup \{ni \mapsto nr\}$
end

Proofs

The Proof Obligation Generator of the Rodin Platform produces 49 proof obligations (invariant, refinement, and well-definedness), all proved automatically by the prover of the Rodin Platform.

5 The Protocol With a Mistake Made by the Initiator and its Discovery

We now present another development where the initiator i records, as in the previous section, the recipient r it wants to speak to. But the initiator i, rather than sending its message $m1$ with the public key of r, sends its message with the public key of any recipient s. Notice again that s might be identical to r but not necessarily. This mistake of the initiator will break invariant **inv1_6**, that is $m1\_nni^{-1} ; crypto = i2$, saying that the public key used in the message $m1$ is that of the recipient recorded by the initiator (in variable $i2$).

In order for the initiator to detect the mistake, the recipient will send its name in its reply message $m2$ so that the initiator can discover that this is not the recipient it wants to speaks to. If this is the case, then the initiator stops the execution of the protocol.

5.1 Initial Model

Sets, Contants, and Axioms

The sets, constants, and axioms are as in section 4.1

sets: AGT, NNC

constants: $Initiator$, $Recipient$

axm_1: partition$(AGT, Initiator, Recipient)$

Variables and Invariants

In this section, the local invariants are prefixed by **2**. Next to the local invariants, we put the corresponding invariants of section 4.1 that are prefixed with **0**. In this way, we can see the differences between the two developments (with or without mistake of the initiator).

The three variables nnc, nni, and nnr are as in section 4.1.

variables: nnc, nni, nnr

inv2_1: $nnc \subseteq NNC$

inv2_2: partition(nnc, nni, nnr)

inv0_1: $nnc \subseteq NNC$

inv0_2: partition(nnc, nni, nnr)

Likewise, the three variables $i1$, $i2$, and $i3$ are as in section 4.1.

variables: $i1$, $i2$, $i3$

inv2_3: $i1 \in nni \rightarrow Initiator$

inv2_4: $i2 \in nni \rightarrow Recipient$

inv2_5: $i3 \in nni \rightarrowtail\!\!\!\!\!\!\rightarrow nnr$

inv0_3: $i1 \in nni \rightarrow Initiator$

inv0_4: $i2 \in nni \rightarrow Recipient$

inv0_5: $i3 \in nni \rightarrowtail\!\!\!\!\!\!\rightarrow nnr$

Things are modified a bit now. Due to the mistake made by the initiator, the variable $r3$ is now a *partial* injection only. We introduce a new variable $r4$ recording what the recipient "believes" to be the connection between the two nonces. Of course, it can be erroneous because the recipient might not be the one that the initiator wanted to speak to (due to its mistake). The variable $r3$ is the corrected connection: as we shall see below, $r3$ can only be extended by event P4.

variables: $r1$, $r2$, $r3$, $r4$

inv2_6: $r1 \in nnr \rightarrow Recipient$

inv2_7: $r2 \in nnr \rightarrow Initiator$

inv2_8: $r3 \in nnr \rightarrowtail\!\!\!\!\!\!\rightarrow nni$

inv2_9: $r4 \in nnr \rightarrowtail nni$

inv0_6: $r1 \in nnr \rightarrow Recipient$

inv0_7: $r2 \in nnr \rightarrow Initiator$

inv0_8: $r3 \in nnr \rightarrowtail nni$

Next are the authentication invariants. We can see that **inv2_11** is the same as **inv0_10**, and that **inv2_12** is the same as **thm0_1**. As **inv0_10** and **thm0_1** were the authentication properties in section 4, we can then deduce that the authentication properties are the same as in section 4.

inv2_10: $r3^{-1} \subseteq i3$

inv2_11: $r3^{-1} ; r1 \subseteq i2$

inv2_12: $i3^{-1} ; i1 \subseteq r2$

inv2_13: $r4 ; i1 = r2$

inv2_14: $i3^{-1} \subseteq r4$

inv0_9: $i3^{-1} \subseteq r3$

inv0_10: $r3^{-1} ; r1 \subseteq i2$

thm0_1: $i3^{-1} ; i1 \subseteq r2$

inv0_11: $r3 ; i1 = r2$

Events

In this section, the events are named Q1, Q2, and Q3. Next to each of them, we put the corresponding events P1, P2, and P3 of section 4 so that we can see their differences.

As can be seen, Q1 and P1 are the same.

Q1
any ni, i, r **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1(ni) := i$
$i2(ni) := r$
end

P1
any ni, i, r **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1(ni) := i$
$i2(ni) := r$
end

The difference between Q2 and P2 corresponds to the mistake made by the initiator. This can be seen in the third guard of Q2 where the recipient r is any recipient, whereas in P2 it was indeed the recipient recorded by the initiator (that is $i2(ni)$). In the action part, one can see that the event Q2 does not update $r3$ as in P2. As a matter of fact, Q2 updates $r4$ only since it is not sure that it is correct.

Q2
any ni, i, r, nr **where**
$ni \in nni \setminus \mathrm{ran}(r4)$
$i = i1(ni)$
$r \in Recipient$
$nr \notin ncc$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto i\}$
$r4 := r4 \cup \{nr \mapsto ni\}$
end

P2
any ni, i, r, nr **where**
$ni \in nni \setminus \mathrm{ran}(r3)$
$i = i1(ni)$
$r = i2(ni)$
$nr \notin ncc$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto i\}$
$r3 := r3 \cup \{nr \mapsto ni\}$
end

Here the difference between the two events Q3 and P3.

Q3
any ni, nr **where**
$nr \mapsto ni \in r4$
$ni \notin \mathrm{dom}(i3)$
$i2(ni) = r1(nr)$
then
$i3 := i3 \cup \{ni \mapsto nr\}$
end

P3
any ni, nr **where**
$nr \mapsto ni \in r3$
$ni \notin dom(i3)$
then
$i3 := i3 \cup \{ni \mapsto nr\}$
end

There is now an additional events, Q4 in order to update $r3$

Q4
any ni, nr **where**
$ni \mapsto nr \in i3$
$nr \notin \mathrm{dom}(r3)$
$i2(ni) = r1(nr)$
then
$r3 := r3 \cup \{nr \mapsto ni\}$
end

Proofs The Proof Obligation Generator of the Rodin Platform produces 49 proof obligations (invariant, well-definedness), all proved automatically by the prover of the Rodin Platform.

5.2 Refinement

Contants

As in section 4.2

sets: MSG

Variables and Invariants

As in section 4.2

variables: msg, $msg1$, $msg2$, $msg3$, $crypto$

inv3_1: $msg \subseteq MSG$

inv3_2: $\mathsf{partition}(msg, msg1, msg2, msg3)$

inv3_3: $crypto \in msg \rightarrow AGT$

inv1_1: $msg \subseteq MSG$

inv1_2: $\mathsf{partition}(msg, msg1, msg2, msg3)$

inv1_3: $crypto \in msg \rightarrow AGT$

Here we see that **inv1_6** has disappeared.

variables: $m1\_ini$, $m1\_nni$

inv3_4: $m1\_ini \in msg1 \rightarrow Initiator$

inv3_5: $m1\_nni \in msg1 \rightarrowtail\!\!\!\!\!\rightarrow nni$

inv3_7: $m1\_nni^{-1} ; m1\_ini = i1$

inv1_4: $m1\_ini \in msg1 \rightarrow Initiator$

inv1_5: $m1\_nni \in msg1 \rightarrowtail\!\!\!\!\!\rightarrow nni$

inv1_6: $m1\_nni^{-1} ; crypto = i2$

inv1_7: $m1\_nni^{-1} ; m1\_ini = i1$

Here we see that messages belonging to $msg2$ have an additional "field", $m2\_rcv$, correponding to the name of the recipient that has received the message $m1$.

variables: $m2\_nnr$, $m2\_nni$, $m2\_rcv$

In section 4.2 the invariant were mentioning $r3$, here it is $r4$ instead.

inv3_8: $m2\_nni \in msg2 \rightarrow \mathrm{ran}(r4)$

inv3_9: $m2\_nnr \in msg2 \rightarrowtail\!\!\!\!\!\!\rightarrow nnr$

inv3_10: $m2\_nnr = m2\_nni\,;\, r4^{-1}$

inv3_11: $m2\_rcv \in msg2 \rightarrow Recipient$

inv1_8: $m2\_nni \in msg2 \rightarrow \mathrm{ran}(r3)$

inv1_9: $m2\_nnr \in msg2 \rightarrowtail\!\!\!\!\!\!\rightarrow nnr$

inv1_10: $m2\_nnr = m2\_nni\,;\, r3^{-1}$

variables: $m3\_nnr$

inv3_12: $m3\_nnr \in msg3 \rightarrow nnr$

There are other invariants that are rather technical and omitted in this text. They were discovered gradually while doing the interactive proofs.

Events

The only difference between the refinement event Q1 and the refinement event P1 corresponds to the introduction of the additional recipient s that is used (instead of r in the event $P1$) in the variable $crypto$.

Q1
any $ni, i, r, m1, s$ **where**
$ni \notin ncc$
$p \in Initiator$
$q \in Recipient$
$m1 \notin msg$
$s \in Recipient$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1 := i1 \cup \{ni \mapsto i\}$
$i2 := i2 \cup \{ni \mapsto r\}$
$msg := msg \cup \{m1\}$
$msg1 := msg1 \cup \{m1\}$
$crypto := crypto \cup \underline{\{m1 \mapsto s\}}$
$m1\_nni := m1\_nni \underline{\cup \{m1 \mapsto} ni\}$
$m1\_ini := m1\_ini \cup \{m1 \mapsto i\}$
end

P1
any $ni, i, r, m1$ **where**
$ni \notin ncc$
$i \in Initiator$
$r \in Recipient$
$m1 \notin msg$
then
$ncc := ncc \cup \{ni\}$
$nni := nni \cup \{ni\}$
$i1 := i1 \cup \{ni \mapsto i\}$
$i2 := i2 \cup \{ni \mapsto r\}$
$msg := msg \cup \{m1\}$
$msg1 := msg1 \cup \{m1\}$
$crypto(m1) := crypto \cup \underline{\{m1 \mapsto r\}}$
$m1\_nni := m1\_nni \cup \underline{\{m1 \mapsto ni\}}$
$m1\_ini := m1\_ini \cup \{m1 \mapsto i\}$
end

The only difference between the refinement event Q2 and the refinement event P2 is the updating of the additional field $m2\_rcv$ of the message $m2$

Q2
any $m1, r, nr, m2$ **where**
$m1 \in msg1$
$m1\_nni(m1) \notin \mathrm{ran}(r4)$
$nr \notin ncc$
$r = crypto(m1)$
$m2 \notin msg$
with
$ni = m1\_nni(m1)$
$i = m\_ini(m1)$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto q\}$
$r2 := r2 \cup \{nr \mapsto m1\_ini(m1)\}$
$r4 := r4 \cup \{nr \mapsto m1\_nni(m1)\}$
$msg := msg \cup \{m2\}$
$msg2 := msg2 \cup \{m2\}$
$m2\_nni(m2) := m1\_nni(m1)$
$m2\_nnr(m2) := nr$
$crypto(m2) := m1\_ini(m1)$
$m2\_rcv(m2) := r$
end

P2
any $m1, r, nr, m2$ **where**
$m1 \in msg1$
$m1\_nni(m1) \notin \mathrm{ran}(r3)$
$nr \notin ncc$
$r = crypto(m1)$
$m2 \notin msg$
with
$ni = m1\_nni(m1)$
$i = m\_ini(m1)$
then
$ncc := ncc \cup \{nr\}$
$nnr := nnr \cup \{nr\}$
$r1 := r1 \cup \{nr \mapsto r\}$
$r2 := r2 \cup \{nr \mapsto m1\_ini(m1)\}$
$r3 := r3 \cup \{nr \mapsto m1\_nni(m1)\}$
$msg := msg \cup \{m2\}$
$msg2 := msg2 \cup \{m2\}$
$m2\_nni(m2) := m1\_nni(m1)$
$m2\_nnr(m2) := nr$
$crypto(m2) := m1\_ini(m1)$
end

Here the difference between Q3 and P3 is more important than in previous cases. The very fundamental guard in the event Q3 is the fourth one, $m2\_rcv(m2) = i2(m2\_nni(m2))$, where it is checked that the message $m2$ is indeed received from the expected recipient. Besides the updating of $i3$ (as in the event P3), the actions contains the sending of message $m3$. Notice that when the guard $m2\_rcv(m2) = i2(m2\_nni(m2))$ does not hold then the protocol does not terminate since the message $m3$ is never sent to the recipient.

Q3
any $m2, m3$ **where**
$m2 \in msg2$
$m2\_nni(m2) \notin \mathrm{dom}(i3)$
$i1(m2\_nni(m2)) = crypto(m2)$
$m2\_rcv(m2) = i2(m2\_nni(m2))$
$m3 \notin msg$
with
$ni = m2\_nni(m2)$
$nr = m2\_nnr(m2)$
then
$i3 := i3 \cup \{m2\_nni(m2) \mapsto m2\_nnr(m2)\}$
$msg := msg \cup \{m3\}$
$msg3 := msg3 \cup \{m3\}$
$m3\_nnr := m3\_nnr \cup \{m3 \mapsto m2\_nnr(m2)\}$
$crypto := crypto \cup \{m3 \mapsto m2\_rcv(m2)\}$
end

P3
any $m2$ **where**
$m2 \in msg2$
$m2\_nni(m2) \notin \mathrm{dom}(i3)$
$i1(m2\_nni(m2)) = crypto(m2)$
with
$ni = m2\_nni(m2)$
$nr = m2\_nnr(m2)$
then
$i3 := i3 \cup \{m2\_nni(m2) \mapsto m2\_nnr(m2)\}$
end

This last event updates the variable $r3$ when the recipient receives the message $m3$

Q4
any $m3$ **where**
$m3 \in msg3$
$m3\_nnr(m3) \notin \mathrm{dom}(r3)$
$r1(m3\_nnr(m3)) = crypto(m3)$
with
$ni = r4(m3\_nni(m3))$
$nr = m3\_nnr(m3)$
then
$r3 := r3 \cup \{m3\_nnr(m3) \mapsto r4(m3\_nnr(m3))\}$
end

Proofs

The Proof Obligation Generator of the Rodin Platform produces 97 proof obligations (invariant, refinement, well-definedness), with 11 of them proved interactively.

6 Conclusion

We presented a formal development with proofs of the Needham-Schroeder-Lowe protocol. The purpose of this exercise was to show that the classical attack of this protocol can be simulated by means of one of the participants making a mistake, Further work is needed to see whether this approach to attacks can be generalized to other protocols

References

1. R.M. Needham and M.D. Schroeder *Using encryption for authentication in large networks of computers*. CACM 21 (1978)
2. G. Lowe *A Breaking and fixing the Needham-Schroeder public-key protocol using FDR*. TACAS 1996 LNCS vol.1055 (1996)
3. J.R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press 2010
4. http: //www.event-b.org *Rodin Platform*