Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR

Gavin Lowe*

ABSTRACT In this paper we analyse the well known Needham-Schroeder Public-Key Protocol using FDR, a refinement checker for CSP. We use FDR to discover an attack upon the protocol, which allows an intruder to impersonate another agent. We adapt the protocol, and then use FDR to show that the new protocol is secure, at least for a small system. Finally we prove a result which tells us that if this small system is secure, then so is a system of arbitrary size.

1 Introduction

In a distributed computer system, it is necessary to have some mechanism whereby a pair of agents can be assured of each other's identity—they should become sure that they really are talking to each other, rather than to an intruder impersonating the other agent. This is the role of an *authentication protocol*.

In this paper we use the Failures Divergences Refinement Checker (FDR) [11, 5], a model checker for CSP, to analyse the Needham-Schroeder Public-Key Authentication Protocol [8]. FDR takes as input two CSP processes, a specification and an implementation, and tests whether the implementation refines the specification [6]. It has been used to analyse many sorts of systems, including communications protocols [10], distributed databases [12], and puzzles; we show here how it may be used to analyse security protocols.

We model the agents taking part in the protocol as CSP processes. We also model the most general intruder who can interact with the protocol: the intruder can observe and intercept messages, and so learn information—such as the values of nonces—and then use this information to introduce fake messages into the system. We use FDR to test whether the protocol correctly achieves authentication, and discover an attack upon the protocol,

*Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom. e-mail `gavin.lowe@comlab.ox.ac.uk`.

which allows the intruder to imitate an agent A in a run of the protocol with another agent B. This attack was previously reported in [7].

We then adapt the protocol, and use FDR to show that the new protocol is secure, at least for a small system with a single initiator and a single responder. We then prove that this implies that a system of arbitrary size is secure: we prove that if there were an attack on any system running the protocol, no matter how large, then there would be an attack on this small system. This proof is by hand, rather than being fully automatic; however, we believe that this proof is considerably simpler than a direct proof of the security of an arbitrarily-sized system.

We believe that security protocols provide an excellent subject for analysis using process algebra tools. It is obviously important to get these protocols right, particularly given the increasing commercial and financial use of the internet. However, many protocols have appeared in the literature only to be later broken. Often the attacks are somewhat subtle and hard to spot—the protocol discussed in this paper appeared 17 years before it was eventually broken. Further, existing formalisms for analysing protocols have not proved very effective—an incorrect proof of the protocol of this paper has appeared in [2].

The main contributions of this paper are two-fold: (1) a study of how errors may be found in security protocols using a tool such as FDR; and (2) a study of how a protocol, running on a system of arbitrary size, may be verified by considering just a single, small system.

2 The Needham-Schroeder Public-Key Protocol

The Needham-Schroeder Public-Key Protocol [8] aims to establish mutual authentication between an *initiator* A and a *responder* B. The protocol uses *public key cryptography* [4, 9]. Each agent A possesses a *public key*, denoted K_a, which any other agent can obtain from a key server. It also possesses a *secret key*, K_a^{-1}, which is the inverse of K_a. We will write $\{m\}_k$ for message m encrypted with key k. Any agent can encrypt a message m using A's public key to produce $\{m\}_{K_a}$; only A can decrypt this message, so this ensures secrecy.

The protocol also uses *nonces*: random numbers generated with the purpose of being used in a *single* run of the protocol. We denote nonces by N_a and N_b: the subscripts are intended to denote that the nonces were generated by A and B, respectively.

The complete Needham-Schroeder Public-Key Protocol involves seven steps. However, in this paper we consider a reduced version with only three steps. In the steps we omit, the two agents request and receive each other's public keys from a key server: omitting these steps is equivalent to assuming that each agent initially has the other's public key. There is a well known attack upon the full protocol [3], which allows an intruder to replay old, compromised public keys, because the key delivery messages contain no proof of freshness; however, this attack is easily prevented. The attack we consider in this paper is newer, and more subtle.

The reduced protocol can be described as:

$$\begin{array}{lll} \text{Message 1.} & A \to B & : \; A.B.\{N_a.A\}_{PK(B)} \\ \text{Message 2.} & B \to A & : \; B.A.\{N_a.N_b\}_{PK(A)} \\ \text{Message 3.} & A \to B & : \; A.B.\{N_b\}_{PK(B)} . \end{array}$$

Here A is an initiator who seeks to establish a session with responder B. A selects a nonce N_a, and sends it along with its identity to B (message 1), encrypted using B's public key. When B receives this message, it decrypts the message to obtain the nonce N_a. It then returns the nonce N_a, along with a new nonce N_b, to A, encrypted with A's key (message 2). When A receives this message it would seem that he should be assured that he is talking to B, since only B should be able to decrypt message 1 to obtain N_a. A then returns the nonce N_b to B, encrypted with B's key. It would seem that B should be assured that he is talking to A, since only A should be able to decrypt message 2 to obtain N_b.

3 Using FDR to find an attack on the Needham-Schroeder Public-Key Protocol

In this section we model the protocol using CSP. We assume that the reader is familiar with CSP, as described in [6]. We model the protocol by defining CSP processes corresponding to each of the two agents. We also give a CSP description of the most general intruder who can interact with the protocol. We then use the FDR refinement checker to test whether the intruder can successfully attack the protocol.

We assume the existence of the sets *Initiator* of initiators, *Responder* of responders, *Key* of public keys, and *Nonce* of nonces. We will represent a protocol message of the form:

$$\text{Message 1.} \quad A \to B \; : \; A.B.\{N_a.A\}_{K_b}$$

by the CSP event $comm.Msg1.A.B.Encrypt.K_b.N_a.A$, etc. We are modelling a protocol message with an encrypted component of the form $\{m\}_k$ by a CSP event containing fields of the form $Encrypt.k.m$. We define the sets of communications events corresponding to the three steps in the protocol:

$$\begin{aligned} MSG1 \;\widehat{=}\; \{&Msg1.a.b.Encrypt.k.n_a.a' \mid \\ &a \in Initiator, a' \in Initiator, b \in Responder, \\ &k \in Key, n_a \in Nonce\}, \end{aligned}$$

$$
\begin{aligned}
MSG2 \ &\widehat{=}\ \{Msg2.b.a.Encrypt.k.n_a.n_b \mid \\
&\qquad a \in Initiator, b \in Responder, \\
&\qquad k \in Key, n_a \in Nonce, n_b \in Nonce\}, \\
MSG3 \ &\widehat{=}\ \{Msg3.a.b.Encrypt.k.n_b \mid \\
&\qquad a \in Initiator, b \in Responder, k \in Key, n_b \in Nonce\}, \\
MSG \ &\widehat{=}\ MSG1 \cup MSG2 \cup MSG3.
\end{aligned}
$$

Standard communications in the system will be modelled by the channel $comm$. We also want to model the fact that the intruder can fake or intercept messages, and so we introduce extra channels $fake$ and $intercept$. We declare these channels:

$$\mathsf{channel}\ comm\ ,\ fake\ ,\ intercept : MSG.$$

We will ensure that the receiver of a faked message is not aware that it is a fake, and that the sender of an intercepted message is not aware that it is intercepted.

We introduce two extra channels, defining the external interface of the protocol. We represent a request from a user for initiator a to connect with responder b by the event $user.a.b$; we represent the resulting session by the event $session.a.b$. We also add channels to represent the state of the agents: these will be useful in the subsequent analysis of the system. We represent the initiator a thinking it is taking part in a run of the protocol with b by the event $I\_running.a.b$, and represent the responder b thinking it is taking part in a run of the protocol with a by the event $R\_running.a.b$; we represent the initiator committing to the session by the event $I\_commit.a.b$, and represent the responder committing to the session by $R\_commit.a.b$. We declare these channels by:

$$
\begin{aligned}
\mathsf{channel}\ & user\ ,\ session\ ,\ I\_running\ ,\ R\_running\ ,\ I\_commit\ ,\ R\_commit : \\
& \quad Initiator.Responder.
\end{aligned}
$$

We will represent a responder with identity a, who has a single nonce n_a, by the CSP process $INITIATOR(a, n_a)$. If we want to consider a responder with more than one nonce, then we can compose several such processes, either sequentially or interleaved. Ignoring, for the moment, the possibility of intruder action, the process can be defined by:

$$
\begin{aligned}
& INITIATOR(a, n_a) \ \widehat{=} \\
& \quad user.a?b \to I\_running.a.b \to \\
& \quad comm!Msg1.a.b.Encrypt.key(b).n_a.a \to \\
& \quad comm.Msg2.b.a.Encrypt.key(a)?n_a'.n_b \to \\
& \quad \text{if} \quad\ \ n_a = n_a' \\
& \quad \text{then} \ \ comm!Msg3.a.b.Encrypt.key(b).n_b \to \\
& \qquad\qquad I\_commit.a.b \to session.a.b \to Skip \\
& \quad \text{else} \ \ \ Stop.
\end{aligned}
$$

The initiator receives a request from the user to connect with responder b, and so starts what he believes is a run of the protocol with b. He sends a message 1, containing the nonce n_a, encrypted with b's public key. He receives a message 2 back, and checks the value of the first nonce. He then sends back the corresponding message 3, commits to the session, and carries out the session.

We now introduce the possibility of enemy action by applying a renaming to the above process. Our renaming should ensure that message 1s and message 3s sent by the initiator can be intercepted, and message 2s can be faked. We define an initiator with identity A and nonce N_a by:[1]

$$
\begin{array}{l}
INITIATOR1 \mathrel{\widehat{=}} \\
\quad INITIATOR(A, N_a) \\
\qquad [[comm.Msg1 \leftarrow comm.Msg1\,,\ comm.Msg1 \leftarrow intercept.Msg1, \\
\qquad\ \ comm.Msg2 \leftarrow comm.Msg2\,,\ comm.Msg2 \leftarrow fake.Msg2, \\
\qquad\ \ comm.Msg3 \leftarrow comm.Msg3\,,\ comm.Msg3 \leftarrow intercept.Msg3]].
\end{array}
$$

We can define a CSP process representing the responder, similarly.

3.1 The intruder

We want to model the intruder as a process that can perform any attack that we would expect a real-world intruder to be able to perform. Thus the intruder should be able to:

- Overhear and/or intercept any messages being passed in the system;
- Decrypt messages that are encrypted with his own public key, so as to learn new nonces;
- Introduce new messages into the system, using nonces he knows;
- Replay any message he has seen (possibly changing plain-text parts), even if he does not understand the contents of the encrypted part.

We assume that the intruder is a user of the computer network, and so can take part in normal runs of the protocol, and other agents may initiate runs of the protocol with him. We will define the most general (i.e. the most non-deterministic) intruder who can act as above. We consider an intruder with identity I, with public key K_i, who initially knows a nonce N_i. All of our refinement tests are in the traces model, so—for reasons of efficiency—we define the intruder using external choices, where nondeterministic choices might seem more natural.

---

[1] This is the process that can perform either a $comm.Msg1$ or $intercept.Msg1$ event whenever $INITIATOR(A, N_a)$ can perform a corresponding $comm.Msg1$, etc.

At any instant, the state of the intruder can be parameterized by the knowledge it has acquired. More precisely, our model of the intruder will be parameterized by the sets $m1s$, $m2s$ and $m3s$ of message 1s, message 2s and message 3s that it has been unable to decrypt, and the set ns of nonces that it knows.

The intruder can observe messages being passed in the system, possibly intercepting them. If the messages are encrypted with its own key K_i, then it can learn new nonces; otherwise it remembers the encrypted component. It can introduce fake messages into the system using nonces that it knows, or by replaying encrypted components that it has been unable to decrypt. This is captured by the following (rather long, but reasonably uniform) CSP definition[2].

$$
\begin{array}{l}
I(m1s, m2s, m3s, ns) \mathrel{\widehat{=}} \\
\qquad comm.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad\qquad \text{else } I(m1s \cup \{Encrypt.k.n.a'\}, m2s, m3s, ns) \\
\qquad \Box\, intercept.Msg1?a.b.Encrypt.k.n.a' \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad\qquad \text{else } I(m1s \cup \{Encrypt.k.n.a'\}, m2s, m3s, ns) \\
\qquad \Box\, comm.Msg2?b.a.Encrypt.k.n.n' \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
\qquad\qquad \text{else } I(m1s, m2s \cup \{Encrypt.k.n.n'\}, m3s, ns) \\
\qquad \Box\, intercept.Msg2?b.a.Encrypt.k.n.n' \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n, n'\}) \\
\qquad\qquad \text{else } I(m1s, m2s \cup \{Encrypt.k.n.n'\}, m3s, ns) \\
\qquad \Box\, comm.Msg3?a.b.Encrypt.k.n \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad\qquad \text{else } I(m1s, m2s, m3s \cup \{Encrypt.k.n\}, ns) \\
\qquad \Box\, intercept.Msg3?a.b.Encrypt.k.n \rightarrow \\
\qquad\qquad \text{if } k = K_i \text{ then } I(m1s, m2s, m3s, ns \cup \{n\}) \\
\qquad\qquad \text{else } I(m1s, m2s, m3s \cup \{Encrypt.k.n\}, ns) \\
\qquad \Box\, fake.Msg1?a.b?m : m1s \rightarrow I(m1s, m2s, m3s, ns) \\
\qquad \Box\, fake.Msg2?a.b?m : m2s \rightarrow I(m1s, m2s, m3s, ns) \\
\qquad \Box\, fake.Msg3?a.b?m : m3s \rightarrow I(m1s, m2s, m3s, ns) \\
\qquad \Box\, fake.Msg1?a.b!Encrypt?k?n : ns?a' \rightarrow I(m1s, m2s, m3s, ns) \\
\qquad \Box\, fake.Msg2?b.a!Encrypt?k?n : ns?n' : ns \rightarrow I(m1s, m2s, m3s, ns) \\
\qquad \Box\, fake.Msg3?a.b!Encrypt?k?n : ns \rightarrow I(m1s, m2s, m3s, ns).
\end{array}
$$

We consider an intruder who initially knows the nonce N_i:

$$INTRUDER \mathrel{\widehat{=}} I(\{\}, \{\}, \{\}, \{N_i\}).$$

[2]In practice, it is more efficient to define the intruder slightly differently, as the interleaving of four components.

3.2 Analyzing the system

We may now define a system with an intruder:[3]

$$\begin{array}{l} AGENTS \mathrel{\widehat{=}} \\ \quad INITIATOR1 \,|[\, \{\!|comm, session.A.B|\!\} \,]|\, RESPONDER1, \\ SYSTEM \mathrel{\widehat{=}} AGENTS \,|[\, \{\!|fake, comm, intercept|\!\} \,]|\, INTRUDER. \end{array}$$

We can use FDR to test whether the protocol correctly authenticates the two agents. FDR takes as two inputs, a specification and an implementation, and test whether the implementation refines the specification. In this paper we are working in the traces model of CSP [6], so checking for refinement amounts to testing whether each trace of the implementation is also a trace of the specification.

To test whether the protocol correctly authenticates the responder, we need to find a specification that allows only those traces where the initiator A commits to a session with B only if B has indeed taken part in the protocol run. The initiator committing to a session is represented by an $I\_commit.A.B$ event; the responder taking part in a run of the protocol with A is represented by $R\_running.A.B$. Thus the authenticity of the responder can be tested using the specification AR:

$$\begin{array}{rcl} AR_0 & \widehat{=} & R\_running.A.B \to I\_commit.A.B \to AR_0, \\ A_1 & \widehat{=} & \{\!|R\_running.A.B, I\_commit.A.B|\!\}, \\ AR & \widehat{=} & AR_0 \,|||\, RUN(\Sigma \setminus A_1). \end{array}$$

AR_0 expresses that an $I\_commit.A.B$ event should only occur after an $R\_running.A.B$ event; interleaving this specification with $RUN(\Sigma \setminus A_1)$ (where Σ is the set of all events) allows all other events to occur in an arbitrary order.

FDR can be used to verify that $SYSTEM$ refines AR, and so the protocol does indeed correctly authenticate the responder[4].

We now consider authentication of the initiator. The protocol should ensure that the responder B commits to a session with initiator A only if A took part in the protocol run. Formally, an $R\_commit.A.B$ event should occur only if there has been a corresponding $I\_running.A.B$ event. This requirement is captured by the specification AI:

$$\begin{array}{rcl} AI_0 & \widehat{=} & I\_running.A.B \to R\_commit.A.B \to AI_0, \\ A_2 & \widehat{=} & \{\!|I\_running.A.B, R\_commit.A.B|\!\}, \\ AI & \widehat{=} & AI_0 \,|||\, RUN(\Sigma \setminus A_2). \end{array}$$

---

[3] Our notation differs a little from [6]: we write $P \,|[\, A \,]|\, Q$ for the parallel composition of P and Q, synchronizing on the set of events A; we write $\{\!|c_1, \ldots, c_n|\!\}$ for the set of all communications over channels $c_1, \ldots, c_n$.

[4] The FDR input files used for this case study can be obtained from URL `http://www.comlab.ox.ac.uk/oucl/users/gavin.lowe/Security/NSPKP/index.html`.

FDR can be used to discover that $SYSTEM$ does *not* refine AI. It finds that after the trace:

$$\begin{aligned}&\langle user.A.I\,,\ I\_running.A.I\,,\\ &\ intercept.Msg1.A.I.Encrypt.K_i.N_a.A\,,\\ &\ fake.Msg1.A.B.Encrypt.K_b.N_a.A\,,\\ &\ intercept.Msg2.B.A.Encrypt.K_a.N_a.N_b\,,\\ &\ fake.Msg2.I.A.Encrypt.K_a.N_a.N_b\,,\\ &\ intercept.Msg3.A.I.Encrypt.K_i.N_b\,,\\ &\ fake.Msg3.A.B.Encrypt.K_b.N_b\rangle\end{aligned}$$

the system can perform $R\_commit.A.B$. Thus the responder B commits to a session with A even though A is not trying to establish a session with B (there has been no corresponding $I\_running.A.B$ event).

We can rewrite this attack as follows. The attack consists of the interleaving of two runs, which we write as α and β. (We use the term *run* for a particular instance of the protocol; we use the term *attack*, for any sequence of events leading to a breach of security.) In run α, A tries to establish a session with I, while in run β, the intruder impersonates A to establish a false session with B. We write, for example, $\beta.2$ to represent message 2 of run β; we write $I(A)$ to represent the intruder imitating A.

$$\begin{array}{lrl}
\text{Message } \alpha.1. & A \rightarrow I & : A.I.\{N_a.A\}_{PK(I)} \\
\text{Message } \beta.1. & I(A) \rightarrow B & : A.B.\{N_a.A\}_{PK(B)} \\
\text{Message } \beta.2. & B \rightarrow I(A) & : B.A.\{N_a.N_b\}_{PK(A)} \\
\text{Message } \alpha.2. & I \rightarrow A & : I.A.\{N_a.N_b\}_{PK(A)} \\
\text{Message } \alpha.3. & A \rightarrow I & : A.I.\{N_b\}_{PK(I)} \\
\text{Message } \beta.3. & I(A) \rightarrow B & : A.B.\{N_b\}_{PK(B)}\,.
\end{array}$$

In message $\alpha.1$, A tries to establish a session with I, sending the none N_a encrypted with I's key. In message $\beta.1$, the intruder imitates A to start a run of the protocol with B, sending the same nonce N_a. B responds by choosing a new nonce N_b, and returning it in message $\beta.2$. The intruder cannot decrypt this message to obtain N_b, but instead uses A as an oracle, by replaying this message in message $\alpha.2$; note that this message is of the form expected by A in run α. A decrypts the message to obtain N_b, and returns this to I in message $\alpha.3$. I can then decrypt this message to obtain N_b, which he returns to B in message $\beta.3$, thus completing run β of the protocol. Hence B believes that he has correctly carried out a run of the protocol with A.

4 A corrected protocol

It is easy to adapt the protocol to prevent the attack found above; we simply include the identity of the responder within the encrypted part of message 2:

$$\begin{array}{lrcl} \text{Message 1.} & A \to B & : & A.B.\{N_a.A\}_{PK(B)} \\ \text{Message 2.} & B \to A & : & B.A.\{N_a.N_b.B\}_{PK(A)} \\ \text{Message 3.} & A \to B & : & A.B.\{N_b\}_{PK(B)}\,. \end{array}$$

This prevents the above attack, because message $\beta.2$ would become:

$$\text{Message } \beta.2. \quad B \to I(A) \;:\; B.A.\{N_a.N_b.B\}_{PK(A)}\,,$$

and the intruder can not successfully replay this in message $\alpha.2$, because A is expecting a message containing I's identity.

We may adapt our CSP representation of the protocol and the intruder. FDR then fails to find any attacks on the protocol in the case where the initiator A and responder B each have a *single* nonce, and so can take part in a *single* run of the protocol. We conclude that the protocol is safe, at least for this small system.

The question remains, though: is a more general system safe from attack? If the agents had more nonces, could the intruder obtain enough knowledge from several runs to be able to attack the protocol? How about if there were more than just the two honest agents involved? Or how about if the same agent could act both as initiator and responder?

These kind of questions arise in many model checking problems, and are not unique to the area of security protocols. We may typically use a tool to verify a small system of fixed size; but this does not necessarily tell us that larger systems are also correct. One solution is to prove—by some method—that if a system of arbitrary size were incorrect, then this would imply that the small system were also incorrect. Following this idea, in Section 6 we prove that if there were an attack on a more general system running the Needham-Schroeder protocol, then there would be an attack on the small system we considered above. But first, we define some notation, and prove a useful result concerning the way in which an intruder responds to a nonce challenge. We adopt a very general setting, so that our results may be applicable to a wide class of protocols.

5 A logic for analyzing protocols

5.1 Messages

We begin by defining the data type of messages. A message may be an atom, the concatenation of two simpler messages, or a message encrypted with a key. We may define the set Msg of messages by the following BNF expression:

$$\begin{array}{rcrcl} a & \in & \mathit{Atom} & ::= & C \mid N \mid k \mid \ldots, \\ m & \in & \mathit{Msg} & ::= & a \mid m.m \mid \{m\}_k, \end{array}$$

where C ranges over the set $Agent$ of agent names, k over the set Key of keys, and N over the set $Nonce$ of nonces. We take the concatenation operator "." to be associative. For each key k, we assume the existence of an inverse k^{-1}, such that a message encrypted with k can be decrypted with k^{-1}: in symmetric crypto-systems, each key is its own inverse; in public key systems, the public and secret keys are inverses.

We may also define what it means for a message to *contain* another:

$$\begin{aligned} a \mathsf{\ contains\ } m \ &\mathrel{\widehat{=}}\ a = m, \\ m_1.m_2 \mathsf{\ contains\ } m \ &\mathrel{\widehat{=}}\ m_1.m_2 = m \vee m_1 \mathsf{\ contains\ } m \vee m_2 \mathsf{\ contains\ } m, \\ \{m_1\}_k \mathsf{\ contains\ } m \ &\mathrel{\widehat{=}}\ \{m_1\}_k = m \vee m_1 \mathsf{\ contains\ } m. \end{aligned}$$

We may use this to define the submessages of a particular message:

$$\mathsf{sub\text{-}msgs}(m) \ \mathrel{\widehat{=}}\ \{m' \in Msg \mid m \mathsf{\ contains\ } m'\}.$$

We will want to be able to discuss which messages an intruder can produce given the messages that he has seen so far. We write $B \vdash m$ to represent that the intruder may derive message m from the finite set of messages B. The following definition is adapted from [13].

$$\begin{aligned} m \in B \ &\Rightarrow\ B \vdash m, && (1) \\ B \vdash m \wedge B \vdash m' \ &\Rightarrow\ B \vdash m.m', && (2) \\ B \vdash m.m' \ &\Rightarrow\ B \vdash m \wedge B \vdash m', && (3) \\ B \vdash m \wedge B \vdash k \ &\Rightarrow\ B \vdash \{m\}_k, && (4) \\ B \vdash \{m\}_k \wedge B \vdash k^{-1} \ &\Rightarrow\ B \vdash m. && (5) \end{aligned}$$

If the intruder has already seen message m (i.e. $m \in B$) then he can produce that message (rule 1). If he can produce both halves of a concatenated message, then he can produce the entier message (rule 2), and vice versa (rule 3). If he can produce a message m and a key k, then he can encrypt m with k (rule 4). If we can produce an encrypted message and the corresponding decrypting key, then he can decrypt the message (rule 5). We also write $B \nvdash m$ for $\neg\,(B \vdash m)$.

We state a few lemmas about the $\vdash$ relation that will prove useful. These may be proved by rule induction.

Lemma 1: If m may be derived from some set of information B, then it may be derived from any larger set of information:

$$B \vdash m \wedge B \subseteq B' \ \Rightarrow\ B' \vdash m.$$

Lemma 2: Derived messages may be used in subsequent derivations:

$$B \vdash m' \wedge B \cup \{m'\} \vdash m \ \Rightarrow\ B \vdash m.$$

Lemma 3: If m may be derived from B, but some sub-message X may not:

$$B \vdash m \;\wedge\; m \,\mathsf{contains}\, X \;\wedge\; B \nvdash X,$$

then there is some encrypted sub-message Y of m, which contains X, may be derived from B, is contained in some element of B, but which cannot be decrypted:

$$\begin{aligned}\exists\, Y \in \mathsf{sub\text{-}msgs}(m) \bullet Y \,\mathsf{contains}\, X \wedge B \vdash Y \wedge \exists\, b \in B \bullet b \,\mathsf{contains}\, Y \\ \wedge\, \exists\, Z \in Msg\,;\, k \in Key \bullet Y = \{Z\}_k \wedge B \nvdash k^{-1}.\end{aligned}$$

Lemma 4: Suppose $A \cup B \vdash x$, x $\mathsf{contains}$ a, and $a \in Atom$. Then either some element of B contains a key, some sub-message of x containing a may be derived from A, or a is contained in some element of B:

$$\begin{aligned}&\exists\, k \in Key \bullet \exists\, b \in B \bullet b \,\mathsf{contains}\, k \wedge A \nvdash k \wedge A \cup B \vdash k \\ &\vee\, \exists\, z \in \mathsf{sub\text{-}msgs}(x) \bullet z \,\mathsf{contains}\, a \wedge A \vdash z \\ &\vee\, \exists\, b \in B \bullet b \,\mathsf{contains}\, a \wedge A \nvdash a.\end{aligned}$$

5.2 Traces

We let *RunId* be the space of *run identifiers*, ranged over by α, β, etc. We define a *message number* to be a (run identifier, natural number) pair.

$$MsgNo \;\widehat{=}\; RunId \times \mathbf{N}.$$

We write $\alpha.i$ for message i of run α.

As above, we will write $I(A)$ to represent the intruder imitating agent A. We define the set $Agent^+$ to contain all agent identifiers, and all such $I(A)$:

$$Agent^+ \;\widehat{=}\; Agent \cup \{I(A) \mid A \in Agent \setminus \{I\}\}.$$

We also define the set of agent identifiers representing the intruder:

$$IntId \;\widehat{=}\; \{I\} \cup \{I(A) \mid A \in Agent \setminus \{I\}\}.$$

We define a *communication* to be a quadruple where the fields are: (1) the message number, (2) the sender of the communication, where $I(A)$ represents the intruder imitating A, (3) the receiver, where $I(A)$ represents the intruder intercepting a message meant for A, and (4) the actual message that is sent:

$$Communication \;\widehat{=}\; MsgNo \times Agent^+ \times Agent^+ \times Msg.$$

When convenient, we will represent the communication $(\alpha.i, A, B, m)$ by the more conventional, and visually more pleasing:

$$\text{Message } \alpha.i. \quad A \to B \;:\; m.$$

We define a *run* to be a sequence s of such communications where all the communications have the same run identifier:

$$\exists\, \alpha \bullet \forall (\beta.i, C, D, m) \text{ in } s \bullet \beta = \alpha,$$

and the honest agents follow the protocol.

We define a *trace* to be a sequence of communications, formed from the interleaving of several runs with distinct run identifiers. We let tr, tr' range over traces.

We may abstract the data included in the communications of a particular run:

$$\mathsf{data}(tr) \;\widehat{=}\; \{m \mid \exists\, \alpha, i, A, B \bullet (\alpha.i, A, B, m) \text{ in } tr\}.$$

We make the assumption that when an honest agent introduces a new nonce into a run of the protocol, the nonce really is freshly chosen; this means that the agent will introduce different nonces into different runs, that no other honest agent will introduce the same nonce, and that the intruder does not initially know the value of the nonce. We term this assumption the *nonce assumption*.

5.3 Intruders

We assume that the intruder has some initial knowledge, which may be represented by a set of atoms IK_0. This will normally include the identities of all agents in the system, all the public keys, and I's own secret key. We may define the intruder's knowledge after a particular trace:

$$\mathsf{knowledge\text{-}after}\, tr \;\widehat{=}\; \{m \mid IK_0 \cup \mathsf{data}(tr) \vdash m\}.$$

The intruder knows the message m after trace tr if m may be derived from the data in tr and I's initial knowledge.

We define a trace to be *valid* if the intruder only produces messages that are derivable from the knowledge that it has acquired:

$$\begin{array}{l}\mathsf{valid}(tr) \;\widehat{=} \\ \quad \forall\, tr' \in Trace\,;\, \alpha.i \in Msg\_No\,;\, m \in Msg\,;\, I' \in IntId\,;\, C \in Agent \bullet \\ \qquad tr' \frown \langle(\alpha.i, I', C, m)\rangle \leq tr \Rightarrow m \in \mathsf{knowledge\text{-}after}\, tr'.\end{array}$$

If the intruder sends a message m after observing tr', then m is in the intruder's knowledge at that point.

We write I learns X from $\alpha.i$ if the intruder learns the piece of information X from message number $\alpha.i$:

$$\begin{array}{l}(I \text{ learns } X \text{ from } \alpha.i)(tr) \;\widehat{=} \\ \quad \exists\, m \in Msg\,;\, A, B \in Agent\,;\, tr' \in Trace \bullet \\ \qquad tr' \frown \langle(\alpha.i, A, B, m)\rangle \leq tr \\ \qquad \wedge\, X \notin \mathsf{knowledge\text{-}after}\, tr' \\ \qquad \wedge\, X \in \mathsf{knowledge\text{-}after}(tr' \frown \langle(\alpha.i, A, B, m)\rangle).\end{array}$$

We drop the argument tr when it is obvious from context.

We write I says X in $\alpha.i$ if the intruder sends the communication $\alpha.i$, which contains X as a sub-message:

$$(I \ \mathsf{says}\ X \ \mathsf{in}\ \alpha.i)(tr) \ \widehat{=}\ \exists\, m \in Msg\,;\, I' \in IntId\,;\, A \in Agent \bullet \\ (\alpha.i, I', A, m) \ \mathsf{in}\ tr \wedge m \ \mathsf{contains}\ X.$$

5.4 *Nonce challenges*

We now prove a result concerning the way in which an intruder meets a nonce challenge. We make some additional assumptions about the protocol in question:

- The encrypted parts of differently numbered messages in the protocol are textually distinct: if M_i is a valid message i, and M_j is a valid message j, and M_i $\mathsf{contains}$ $\{M\}_k$ and M_j $\mathsf{contains}$ $\{M\}_k$ then $i = j$. Thus it is always possible to tell which message an encrypted part comes from; this means, for example, that the intruder cannot replay some encrypted text taken from a message 1, and have it interpreted as a message 2.
- All runs of the protocol have essentially the same form.
- The intruder does not learn any additional keys during a trace: if $k \in \mathsf{knowledge\text{-}after}\, tr$ then $k \in IK_0$.

Note that the Needham-Schroeder Public Key Protocol satisfies these assumptions: that the intruder does not learn any additional keys during a trace follows from the fact that that secret keys are never passed during the protocol (this can be proved formally using Lemma 4).

Theorem 5: Consider a valid trace tr that includes a nonce challenge, met by the intruder:

$$\begin{array}{lrcl} \text{Message } \alpha.i. & A \rightarrow I(B) & : & M_1(N) \\ \text{Message } \alpha.j. & I(B) \rightarrow A & : & M_2(N), \end{array}$$

where $M_1(N)$ and $M_2(N)$ contain the nonce N:

$$M_1(N) \ \mathsf{contains}\ N \ \wedge\ M_2(N) \ \mathsf{contains}\ N,$$

and the nonce is first introduced in message $\alpha.i$, and first returned in $\alpha.j$. Suppose further that

$$\forall\, z \in \mathsf{sub\text{-}msgs}(M_2(N)) \bullet \\ z \ \mathsf{contains}\ N \Rightarrow IK_0 \cup \{z\} \vdash N \vee IK_0 \cup \{z\} \vdash M_2(N). \tag{6}$$

Then one of the following holds:

1. The intruder can produce either N or $M_2(N)$ immediately from message $\alpha.i$:

$$I \textsf{ learns } N \textsf{ from } \alpha.i \;\vee\; I \textsf{ learns } M_2(N) \textsf{ from } \alpha.i.$$

2. The intruder replays some encrypted sub-message Y of $\alpha.i$ in message i of some other run β:

$$\begin{aligned}&\exists\, Y \bullet M_1(N) \textsf{ contains } Y \wedge Y \textsf{ contains } N \wedge I \textsf{ says } Y \textsf{ in } \beta.i\\ &\qquad \wedge \exists\, Z \in Msg\,;\, k \in Key \bullet Y = \{Z\}_k \wedge k^{-1} \notin IK_0,\end{aligned}$$

and learns either N or $M_2(N)$ from a later message of β:

$$\exists\, k > i \bullet I \textsf{ learns } N \textsf{ from } \beta.k \vee I \textsf{ learns } M_2(N) \textsf{ from } \beta.k.$$

Proof: Assume the conditions of the theorem. By the nonce assumption, the intruder does not know the value of N or $M_2(N)$ before message $\alpha.i$. Clearly, the intruder must, at some point, learn $M_2(N)$ in order to produce message $\alpha.j$. Suppose the intruder first learns either N or $M_2(N)$ from message $\beta.k$ with contents M_3:

$$\text{Message } \beta.k. \quad C \to D \;:\; M_3.$$

Formally:

$$\begin{aligned}&tr' \frown \langle(\beta.k, C, D, M_3)\rangle \le tr\\ &\wedge N, M_2(N) \notin \textsf{knowledge-after}\, tr'\\ &\wedge (N \in \textsf{knowledge-after}(tr' \frown \langle(\beta.k, C, D, M_3)\rangle)\\ &\quad \vee M_2(N) \in \textsf{knowledge-after}(tr' \frown \langle(\beta.k, C, D, M_3)\rangle)).\end{aligned}$$

If $\alpha.i = \beta.k$ then we have case 1. Otherwise, $\beta.k$ occurs after $\alpha.i$, but before $\alpha.j$.

We now show that M_3 $\textsf{contains}$ N. If the intruder learns N from message $\beta.k$, i.e. $IK_0 \cup \textsf{data}\, tr' \cup \{M_3\} \vdash N$, then the result follows from Lemma 4 and the assumption that the intruder's key knowledge is constant. If the intruder learns $M_2(N)$ from message $\beta.k$, i.e. $IK_0 \cup \textsf{data}\, tr' \cup \{M_3\} \vdash M_2(N)$, then again from Lemma 4 we have:

$$\begin{aligned}&\exists\, z \in \textsf{sub-msgs}\, M_2(N) \bullet z \textsf{ contains } N \wedge IK_0 \cup \textsf{data}\, tr' \vdash z\\ &\vee M_3 \textsf{ contains } N.\end{aligned}$$

But the first disjunct and equation 6 would imply that $IK_0 \cup \textsf{data}\, tr' \vdash N$ or $IK_0 \cup \textsf{data}\, tr' \vdash M_2(N)$, contradicting the above. Hence M_3 $\textsf{contains}$ N.

By the assumption that $\alpha.j$ is the first message after $\alpha.i$ that contains N, we have $\alpha \neq \beta$. From the nonce assumption, we must have that the nonce N

is introduced into run β by the intruder, say in message $\beta.l$ ($l < k$), following trace tr'':

$$\begin{array}{l} \exists I' \in IntId\,;E \in Agent \bullet tr''\frown\langle(\beta.l, I', E, M_4(N))\rangle \leq tr' \\ \wedge\ M_4(N) \textsf{ contains } N. \end{array}$$

That is:

$$\text{Message } \beta.l. \quad I' \to E \ :\ M_4(N).$$

By assumption, the trace is valid, so $IK_0 \cup \textsf{data}\, tr'' \vdash M_4(N)$. Hence, from Lemma 3 and the fact that the intruder does not know N after tr'', $M_4(N)$ contains some encrypted sub-message Y of some previous message $\gamma.m$ with contents $M_5(N)$:

$$\begin{array}{l} \exists Y \bullet M_4(N) \textsf{ contains } Y \wedge Y \textsf{ contains } N \wedge Y \in \textsf{knowledge-after}\, tr'' \\ \quad \wedge\ \exists F, G \in Agent \bullet (\gamma.m, F, G, M_5(N)) \textsf{ in } tr'' \wedge M_5(N) \textsf{ contains } Y \\ \quad \wedge\ \exists Z \in Msg\,; k \in Key \bullet Y = \{Z\}_k \wedge k^{-1} \notin \textsf{knowledge-after}\, tr''. \end{array}$$

That is:

$$\text{Message } \gamma.m. \quad F \to G \ :\ M_5(N).$$

By the assumption that encrypted components of differently numbered messages are textually distinct, we must have that $M_4(N)$ and $M_5(N)$ have the same message number, i.e. $l = m$; hence $\beta \neq \gamma$. Either $\alpha = \gamma$ or $\alpha \neq \gamma$; we show that the latter case leads to a contradiction.

So suppose that $\alpha \neq \gamma$. By the nonce assumption, N must have been introduced into run γ by the intruder, say in message $\gamma.n$ ($n < l$). By the assumption that all runs take the same form, message $\beta.n$ must also contain N. But this contradicts the assumption that N was introduced into β in message $\beta.l$.

Hence we have that $\alpha = \gamma$. Message $\alpha.m$ precedes $\alpha.j$ and contains N, and so must be $\alpha.i$. Collecting all the above information gives us case 2. $\square$

6 Verifying systems of arbitrary size

In this section we show that if there is an attack upon a system of arbitrary size running the corrected protocol given in Section 4, then there is an attack upon the small system described above, with a single initiator A and a single responder B, each of which has a single nonce, and so can carry out a single run of the protocol. The proofs proceed by considering a run leading to a failure of authentication, and considering how many extra runs are needed for the intruder to learn any additional information it uses.

6.1 Attacks upon the initiator

In this section we show that if the intruder may imitate the responder to attack the initiator in a system of arbitrary size, then there is a similar attack upon the small system described above.

Consider a run, α, where the intruder imitates the responder B to attack the initiator A:

$$\begin{array}{lrl} \text{Message } \alpha.1. & A \to I(B) & : A.B.\{N_a.A\}_{PK(B)} \\ \text{Message } \alpha.2. & I(B) \to A \quad & : B.A.\{N_a.N_b.B\}_{PK(A)} \\ \text{Message } \alpha.3. & A \to I(B) & : A.B.\{N_b\}_{PK(B)}. \end{array}$$

Note that the intruder only needs to send message 2 in this run, so the only additional runs necessary are those that are needed in order to produce $B.A.\{N_a.N_b.B\}_{PK(A)}$.

The intruder cannot decrypt message $\alpha.1$, because he does not know B's secret key. Hence he learns neither N_a nor $B.A.\{N_a.N_b.B\}_{PK(A)}$ from $\alpha.1$:

$$\neg\,(I \textsf{ learns } N_a \textsf{ from } \alpha.1) \;\wedge\; \neg\,(I \textsf{ learns } B.A.\{N_a.N_b.B\}_{PK(A)} \textsf{ from } \alpha.1),$$

so from Theorem 5, the intruder must replay the encrypted part of message $\alpha.1$ in message 1 of another run, β say, and learn either N_a or $B.A.\{N_a.N_b.B\}_{PK(A)}$ from message $\beta.2$:

$$\begin{array}{l} I \textsf{ says } \{N_a.A\}_{PK(B)} \textsf{ in } \beta.1 \\ \wedge\,(I \textsf{ learns } N_a \textsf{ from } \beta.2 \vee I \textsf{ learns } B.A.\{N_a.N_b.B\}_{PK(A)} \textsf{ from } \beta.2). \end{array}$$

Note that the responder in run β must be B, because message $\beta.1$ is encrypted with B's public key. The intruder does not need any additional information in order to carry out this second run, so only the two runs are needed.

Thus if the intruder can imitate the responder B to attack A, then such an attack would have been found by considering the small system above.

6.2 Attacks upon the responder

Consider an attack where the intruder succeeds in imitating the initiator A in a run, α say, of the protocol with responder B:

$$\begin{array}{lrl} \text{Message } \alpha.1. & I(A) \to B \quad & : A.B.\{N_a.A\}_{PK(B)} \\ \text{Message } \alpha.2. & B \to I(A) & : B.A.\{N_a.N_b.B\}_{PK(A)} \\ \text{Message } \alpha.3. & I(A) \to B \quad & : A.B.\{N_b\}_{PK(B)}. \end{array}$$

Note that the intruder only needs to produce Messages 1 and 3 in this run, so the only additional runs necessary are those that are needed in order for the intruder to learn something that it sends in one of these messages.

Firstly consider the nonce handshake using N_b. The intruder cannot decrypt message $\alpha.2$, so he learns neither N_b nor $A.B.\{N_b\}_{PK(B)}$ from $\alpha.2$:

$$\neg\ (I \text{ learns } N_b \text{ from } \alpha.2)\ \wedge\ \neg\ (I \text{ learns } A.B.\{N_b\}_{PK(B)} \text{ from } \alpha.2),$$

so from Theorem 5 the intruder must replay the encrypted part of message $\alpha.2$ in message 2 of some other run, β say, and learn N_b or $A.B.\{N_b\}_{PK(B)}$ from message $\beta.3$:

$$\begin{array}{l} I \text{ says } \{N_a.N_b.B\}_{PK(A)} \text{ in } \beta.2 \\ \wedge\ (I \text{ learns } N_b \text{ from } \beta.3 \vee I \text{ learns } A.B.\{N_b\}_{PK(B)} \text{ from } \beta.3). \end{array}$$

Note that the initiator of run β must be A, because message $\beta.2$ is encrypted with A's public key. Further, from the form of message $\beta.2$ we see that A must believe that he is communicating with B. Hence run β is of the form:

$$\begin{array}{lrl} \text{Message } \beta.1. & A \rightarrow I(B) & : A.B.\{N_a.A\}_{PK(B)} \\ \text{Message } \beta.2. & I(B) \rightarrow A & : B.A.\{N_a.N_b.B\}_{PK(A)} \\ \text{Message } \beta.3. & A \rightarrow I(B) & : A.B.\{N_b\}_{PK(B)}. \end{array}$$

Now we see that the intruder learns the component $\{N_a.A\}_{PK(B)}$ from message $\beta.1$, and replays this in $\alpha.1$, and so only these two runs are necessary for the intruder to learn all the knowledge it uses in the attack. Thus if the intruder can imitate the initiator A to attack the responder B, then such an attack would have been found by considering the small system above.

In fact, in this case, we have shown that if there were an attack, it would be of the above form; but the above does not lead to any error of authentication; we may deduce that there is no such attack on the system—even without the aid of FDR.

6.3 Summary

Above we showed that in order to discover an attack upon the protocol, it is enough to consider a system with a single initiator and responder, each with a single nonce.

We now prove a similar result concerning the intruder: it is enough to consider an intruder with a single identity, I say, and a single public key, K_i say. Thus, two intruders working together are no more powerful than a single intruder. Further, it is enough to consider an intruder who initially knows a single nonce, N_i say.

We make the assumption that the honest agents act the same way regardless of the actual values of nonces introduced by other agents; we term this *data independence.*

Suppose, then, that there is a successful attack where the intruder uses more than one identity, more than one public key, and/or more than one nonce. Consider the attack where each intruder's identity is renamed to I, each intruder's key is renamed to K_i, and each intruder's nonce is renamed to N_i. Then, by the data independence assumption, each run proceeds as before. Further, at each stage or the new attack, the intruder's knowledge is related to his knowledge at the corresponding stage of the original attack, in the obvious way, i.e. by the above renaming (this can be proved formally from the definition of the $\vdash$ relation). Thus the intruder is able to produce all of his messages in the new attack. Hence the new attack is indeed successful, and is made by an attacker with a single identity, single public key, and single nonce.

Putting together all these results, we deduce that if there is an attack on a system running the protocol, we would have found it by applying FDR to the small system in Section 4. Hence the protocol is secure.

7 Conclusion

In this paper we have used the Failures Divergences Refinement Checker for CSP to analyse the Needham-Schroeder Public-Key Protocol. We have encoded the protocol and an intruder in CSP, and used FDR to discover a security flaw. We have adapted the protocol to remove this flaw, and used FDR to verify that there are no attacks upon a small system running the protocol. We then proved that this was enough to prove that there are no attacks upon a more general system.

We should be clear as to precisely what we have proved. We have proved that the protocol in Section 4 is secure subject to the assumptions we have made about the method of encryption used, encapsulated in the definition of the $\vdash$ relation. We have assumed that the encryption used is reasonable, in that the intruder is unable to guess the values of keys it does not know. Further, we assume that secret keys are indeed kept secret. We have also assumed that the intruder may not alter an encrypted message before replaying it (unless the message is encrypted using the intruder's key). However, if Cipher Block Chaining is used (see e.g. [14]) then (subject to certain assumptions) it is possible to split an encrypted message into encrypted sub-messages; using the notation of this paper:

$$B \vdash \{m_1.m_2\}_k \;\Rightarrow\; B \vdash \{m_1\}_k \wedge B \vdash \{m_2\}_k.$$

Thus, our proof in not valid in this case. See [1] for examples of attacks upon protocols using CBC.

We believe that this method of analyzing security protocols is very practical. Encoding the protocol and the intruder in CSP is normally straightforward. And the tests using FDR are fast, typically taking less than two minutes. The proof that the security of a small system implies the security of an arbitrarily-sized system is by hand, rather than being fully automatic.

However, we believe that this proof is considerably simpler than a direct proof of the security of an arbitrarily-sized system: we effectively prove the general form an attack must take, and use FDR to do the tedious checking of details.

We intend to analyse more protocols using this approach. In particular, we would like to produce more lemmas and theorems that are useful in proving results concerning the size of system it is necessary to consider in order to be sure that there are no attacks upon a protocol; eventually, we hope to identify properties of protocols (concerning, for example, the number of nonce challenges) that are enough to give us such results directly.

8 References

[1] Colin Boyd. Hidden assumptions in cryptographic protocols. *Proceedings of the IEE*, 137, Part E(6):433–436, November 1990.

[2] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[3] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[4] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.

[5] Formal Systems (Europe) Ltd. *Failures Divergence Refinement—User Manual and Tutorial*, 1993. Version 1.3.

[6] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[7] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

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

[9] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[10] A. W. Roscoe. Developing and verifying protocols in CSP. In *Proceedings of Mierlo workshop on protocols.* TU Eindhoven, 1993.

[11] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[12] A. W. Roscoe and Helen MacCarthy. Verifying a replicated database: A case study in model-checking CSP. Submitted for publication.

[13] Steve Schneider. Security properties and CSP. In preparation, 1995.

[14] Bruce Schneier. *Applied Cryptography*. Wiley, 1994.