Using CSP for protocol analysis: the Needham-Schroeder Public-Key Protocol

Steve Schneider
Royal Holloway, University of London

Technical Report

CSD-TR-96-14

November 21, 1996

Department of Computer Science
Egham, Surrey TW20 0EX, England

Abstract

This paper presents a general approach for analysis and verification of authentication properties in CSP. It is illustrated by an examination of the Needham-Schroeder Public-Key protocol. The paper aims to develop a specific theory appropriate to the analysis of authentication protocols, built on top of the general CSP semantic framework. This approach aims to combine the ability to express such protocols in a natural and precise way with the ability to reason formally about the properties they exhibit.

Contents

1 Introduction

Authentication comes in a number of flavours. For example, Gollmann [Gol96] has identified four different varieties of authentication, which raises the question for any particular authentication protocol as to which kind of authentication the protocol was designed for, and which kinds it actually provides.

The aim of the CSP approach is to reduce questions about security protocols and properties to questions concerning whether CSP processes satisfy particular CSP specifications. This approach forces the separation of properties and protocols, and allows discussion of what is meant by particular kinds of security property independently of the protocols that are intended to achieve them. The formal analysis will then be entirely within the CSP framework which allows the possibility of verification of protocols with respect to the CSP properties.

The general CSP framework proposed by this author has been described in [Sch96]. That paper is concerned with the theoretical foundations. It offers definitions of security properties, including anonymity. This means that since a CSP description of a protocol has a precisely defined semantics it is a precise mathematical question as to whether the protocol meets the property or not. However, the practicalities of how such a verification might be carried out was not addressed; this is the purpose of this paper.

The approach taken here is firstly to express the protocol in CSP. The authentication property we consider states that if some events R in the system are restricted, then other events T should not occur. We establish this by defining a suitable rank function on messages which shows that only messages above a particular rank can circulate in the restricted system, and hence that messages from T are not possible.

CSP is particularly suitable for describing protocols at a level close to the level we think of them. In other contexts the rank argument essentially amounts to an unreachability analysis or a proof that a particular word is not in a language. The strength of this approach is that there is a formal link between the rank arguments and a natural description of the protocol. Formalisation of the protocol into CSP also exposes issues and forces design decisions that may not have been explicit in the original abstract protocol description. The formalisation of the required authentication property likewise forces consideration of what, precisely, is meant by authentication. It is useful to know which precise (CSP) properties the protocol does indeed guarantee, and which it does not.

One of the strengths of CSP is the ease with which specialised theories can be constructed on top of the semantic models. This allows particular specification statements to be defined in terms of the standard semantics, and new proof rules appropriate to these specifications to be provided. This approach is taken here, where we specify and reason about authentication properties, and also about agents' inability to generate particular messages. Although standard proof rules would support the verification (since they are sound and complete), it is preferable to develop a specialised theory since it provides an appropriate level of abstraction for supporting the kind of reasoning we require.

2 CSP notation

CSP is an abstract language designed specifically for the description of communication patterns of concurrent system components that interact through message passing. It is underpinned by a theory which supports analysis of systems described in

CSP. It is therefore well suited to the description and analysis of network protocols: protocols can be described within CSP, as can the relevant aspects of the network. Their interactions can be investigated, and certain aspects of their behaviour can be verified through use of the theory. This section introduces the notation and ideas used in this paper. In particular, only the traces model for CSP is used here. For a fuller introduction to the language the reader is referred to [Hoa85].

Events

Systems are modelled in terms of the events that they can perform. The set of all possible events (fixed at the beginning of the analysis) is denoted Σ. Events may be atomic in structure or may consist of a number of distinct components. For example, an event $put.5$ consists of two parts: a channel name put, and a data value 5. An example of events used in this paper are those of the form $c.i.j.m$ consisting of a channel c, a source i, a destination j and a message m. If M and N are sets of messages, then $M.N$ will be the set of messages $\{m.n \mid m \in M \wedge n \in N\}$. If m is a single message then we elide the set brackets and define $m.N$ to be $\{m\}.N$. Thus for example the set of events $i.N.m = \{i.n.m \mid n \in N\}$. A channel c is said to be of type M if any message $c.m \in \Sigma$ has that $m \in M$.

Processes

Processes are the components of systems. They are the entities that are described using CSP, and they are described in terms of the possible events that they may engage in. The process *Stop* is the process that can engage in no events at all; it is equivalent to deadlock. The output $c!v \rightarrow P$ is able initially to perform only $c.v$, the output of v on channel c, after which it behaves as P. The input $c?x : T \rightarrow P(x)$ can accept any input x of type T along channel c, following which it behaves as $P(x)$. Its first event will be any event of the form $c.t$ where $t \in T$. The process $P \Box Q$ (pronounced 'P choice Q') can behave either as P or as Q: its possible communications are those of P and those of Q. An indexed form of choice $\Box_{i \in I} P_i$ is able to behave as any of its arguments P_i. Another form is nondeterministic choice: the process $P \sqcap Q$ is able to behave nondetermistically either as P or as Q. Its indexed form is $\bigsqcap_{i \in I} P_i$.

Processes may also be composed in parallel. If D is a set of events then the process $P \,|[\, D \,]|\, Q$ behaves as P and Q acting concurrently, with the requirement that they have to synchronise on any event in the synchronisation set D; events not in D may be performed by either process independently of the other. A special form of parallel operator in which the two components do not interact on any events is $P \,|||\, Q$ which is equivalent to $P \,|[\, \{\} \,]|\, Q$.

Processes may be recursively defined by means of equational definitions. Process names must appear on the left hand side of such definitions, and CSP expressions which may include those names appear on the right hand side. For example, the definition

$$LIGHT = on \rightarrow off \rightarrow LIGHT$$

defines a process *LIGHT* whose only possible behaviour is to perform *on* and *off* alternately.

Mutually recursive processes may also be defined, where a (possibly infinite) collection of process names X_k appear on the left hand side of definitions, and CSP expressions $F_k(\vec{X})$ possibly involving any of those names appear on the right. For example, the set of definitions

$$\begin{array}{rcl} COUNT_0 & \widehat{=} & up \rightarrow COUNT_1 \\ COUNT_{n+1} & \widehat{=} & (up \rightarrow COUNT_{n+2}) \,\Box\, down \rightarrow COUNT_n \end{array}$$

define a collection of processes; $COUNT_0$ can do any number of *up* and *down* events, but can never do more *downs* than *ups*.

Process definitions may also contain conditions to separate difference cases. The collection of $COUNT$ definitions could also be given as

$$\begin{array}{rcll} COUNT_m & \widehat{=} & up \rightarrow COUNT_{m+1} & \text{if } m = 0 \\ & & (up \rightarrow COUNT_{m+2}) \,\Box\, down \rightarrow COUNT_m & \text{otherwise} \end{array}$$

or could be given by $COUNT_m \mathrel{\widehat{=}} F_m(\vec{COUNT})$, where

$$\begin{array}{rcll} F_m(\vec{COUNT}) & \widehat{=} & up \rightarrow COUNT_{m+1} & \text{if } m = 0 \\ & & (up \rightarrow COUNT_{m+2}) \,\Box\, down \rightarrow COUNT_m & \text{otherwise} \end{array}$$

For readability, subscripted information may also appear bracketed on the same line as its corresponding process: for example, $COUNT_m$ may also be written as $COUNT(m)$.

For a full discussion of single and mutually recursive process definitions, see [DaS93].

Traces

The semantics of a process P is defined to be the set of sequences of events ($traces(P)$) that it may possibly perform. Examples of traces include $\langle\rangle$ (the empty trace, which is possible for any process) and $\langle on, off, on\rangle$ which is a possible trace of $LIGHT$.

A useful operator on traces is projection: If D is a set of events then the trace $tr \upharpoonright D$ is defined to be the maximal subsequence of tr all of whose events are drawn from D. If D is a singleton set $\{d\}$ then we overload notation and write $tr \upharpoonright d$ for $tr \upharpoonright \{d\}$. Message extraction $tr \downarrow C$ for a set of channel names C provides the maximal sequence of messages passed on channels C. Finally, $tr \Downarrow C$ provides the set of messages in tr passed along some channel in C. These may all be described by sequence or set comprehension:

$$\begin{array}{rcl} tr \upharpoonright D & = & \langle d \leftarrow tr \mid d \in D\rangle \\ tr \downarrow C & = & \langle m \bullet c.m \leftarrow tr \mid c \in C\rangle \\ tr \Downarrow C & = & \{m \mid (tr \downarrow C) \upharpoonright m \neq \langle\rangle\} \end{array}$$

If tr is a sequence, then $\sigma(tr)$ is the set of events appearing in the sequence. The operator σ extends to processes: $\sigma(P)$ is the set of events that appear in some trace of P.

In the traces model, if $traces(Q) \subseteq traces(P)$ then we say that Q is a refinement of P, written $P \sqsubseteq Q$.

Analysing processes

Specifications are given as predicates on traces, and a process P satisfies a specification S if all of its traces satisfy S:

$$P \ \mathbf{sat} \ S \Leftrightarrow \forall \, tr \in traces(P).S$$

Then

$$P \sqsubseteq Q \wedge P \ \mathbf{sat} \ S \Rightarrow Q \ \mathbf{sat} \ S$$

Proof rules

The traces model for CSP is associated with a proof system for describing specifications on processes in terms of specifications on their components. There is a proof rule for each CSP operator. For example, the rule for the prefix operator is

$$\frac{P \ \mathbf{sat} \ S(tr)}{a \rightarrow P \ \mathbf{sat} \ (tr = \langle\rangle \vee (tr = \langle a \rangle \frown tr' \wedge S(tr')))}$$

Particular application domains are often concerned only with specifications of one particular form. In such cases, it is often possible to give more specialised proof rules for such specifications. This amounts to developing a specialised theory and proof system for this application area. The benefits are that the full generality of the proof rules are not required, and there will often be lemmas and theorems, built on top of the traces model, which allow higher-level reasoning. This is the approach taken in this paper, where a key property `maintains` ρ `on` i used for verification is defined in terms of traces, and various proof rules are provided for deducing when particular process descriptions meet this property. We will see the specialised rules in Figures 2 and 5.

Equations

The traces model for CSP supports a number of algebraic equivalences on processes. These are often useful in manipulating process descriptions into a form which is easier to reason about. There are many laws expressing useful identities. For the purposes of this paper, we will be interested in the effect of restricting particular events of a parallel combination. The equations are given in Figure 1.

Rule `restrict.1` states that restricting a process on a set of events A that it cannot perform has no effect. Rule `restrict.2` states that restricting a process on a set of events distributes over interleaving.

Rule `restrict.3` states that two ways of constructing a system are the same:

- ◇ taking a parallel combination of two processes and restricting them on a subset A of their synchronisation set B
- ◇ restricting one of the processes on A and then placing the result in parallel with the other.

Rules `restrict.4` and `restrict.5` are concerned with the effect of a restriction on inputs and outputs.

These equations are used throughout the paper whenever a process of the form $USER_a \, |[\, S \,]| \, Stop$ is expanded. They will not be referred to explicitly when used, in order to avoid cluttering proofs.

Rule `restrict.1` If $\sigma(P) \cap A = \varnothing$ then $P \,|[\, A \,]|\, Stop = P$

Rule `restrict.2`

$$(P \,|||\, Q) \,|[\, A \,]|\, Stop \quad = \quad (P \,|[\, A \,]|\, Stop) \,|||\, (Q \,|[\, A \,]|\, Stop)$$

Rule `restrict.3` If $A \subseteq B$ then

$$(P \,|[\, A \,]|\, Q) \,|[\, B \,]|\, Stop \quad = \quad (P \,|[\, B \,]|\, Stop) \,|[\, A \,]|\, Q$$

Rule `restrict.4`

$$(c?x : T \rightarrow P(x)) \,|[\, B \,]|\, Stop \quad = \quad c?x : U \rightarrow (P(x) \,|[\, B \,]|\, Stop)$$

where $U = T \setminus \{t \mid c.t \in B\}$.

Rule `restrict.5`

$$\begin{array}{lcll} (c!v \rightarrow P) \,|[\, B \,]|\, Stop & = & c!v \rightarrow (P \,|[\, B \,]|\, Stop) & \text{if } c.v \notin B \\ & & Stop & \text{if } c.v \in B \end{array}$$

Figure 1 Equations for restricted parallel combinations

3 The general CSP model

3.1 Authentication

An message-oriented approach to authentication is discussed in [Sch96]. Authentication considers a set of messages T to authenticate another set of messages R if occurrence of some element of T must have been preceded in P by occurrence of some element of R. The sets T and R are taken to be disjoint. The trace specification may be captured as follows

Definition 3.1

$$T \ \texttt{authenticates} \ R \quad = \quad tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle$$

$\square$

The following Lemmas are immediate consequences of the definition.

Lemma 3.2

$$P \ \textbf{sat} \ T \ \texttt{authenticates} \ R \quad \Leftrightarrow \quad P \,|[\, R \,]|\, Stop \ \textbf{sat} \ tr \upharpoonright T = \langle\rangle$$

$\square$

Lemma 3.3 if P **sat** T `authenticates` R and $R \subseteq R'$
then P **sat** T `authenticates` R' $\square$

Transitivity of authentication also follows from the definition:

Lemma 3.4 T `authenticates` R and R `authenticates` R' implies T `authenticates` R' $\square$

The CSP trace semantics also support a number of proof rules for establishing when particular processes provide authentication. Those used in this paper are given in Figure 2.

Rule `auth.stop`

$$\frac{}{\mathit{Stop}\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}$$

Rule `auth.prefix.1`

$$\frac{}{a \rightarrow P\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}\quad [\, a \in R \,]$$

Rule `auth.prefix.2`

$$\frac{P\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}{a \rightarrow P\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}\quad [\, a \notin T \,]$$

Rule `auth.choice`

$$\frac{\forall\, j. V_j\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}{\square_j\ V_j\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}$$

Rule `auth.parallel`

$$\frac{P\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}{P \,|[\, A \,]|\, Q\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}\quad [\, (R \cup T) \subseteq A \,]$$

Rule `auth.interleaves`

$$\frac{\begin{array}{l}P\ \mathbf{sat}\ T\ \texttt{authenticates}\ R\\ Q\ \mathbf{sat}\ T\ \texttt{authenticates}\ R\end{array}}{P \,|||\, Q\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}$$

Rule `auth.recursion`

$$\frac{\begin{array}{l}(\forall\, k. X_k\ \mathbf{sat}\ T\ \texttt{authenticates}\ R) \Rightarrow\\ \qquad (\forall\, k. F_k(\vec{X})\ \mathbf{sat}\ T\ \texttt{authenticates}\ R))\end{array}}{\forall\, k. X(k)\ \mathbf{sat}\ T\ \texttt{authenticates}\ R}\quad [\, \forall\, k. X_k \mathrel{\widehat{=}} F_k(\vec{X}) \,]$$

Figure 2 Proof rules for authentication

The soundness of the rules follows from the trace semantics of the operators, and the formal definition of T **`authenticates`** R. We may give informal justification of their soundness by considering that occurrence of an event from T is intended to provide evidence that some event from R previously occurred. Hence a process *fails* to satisfy T **`authenticates`** R only when some event from T occurs before some event from R.

Rule `auth.stop` is therefore sound because *Stop* cannot perform any events at all, and so cannot perform some R before some T.

Rule `auth.prefix.1` is sound because if the very first event a performed by $a \rightarrow P$ is an event from R, then it is not possible for an event from T to occur before an event from R.h is is independent of the nature of the subsequent process P, which therefore has no restrictions placed on it by the rule—the rule is applicable for any process P.

Rule `auth.prefix.2` is most useful when the event a is not in R, since otherwise `auth.prefix.1` is applicable. In this case it states that if the first event is not in T, then occurrence of a is irrelevant to authentication of R by T, and such authentication is guaranteed for $a \rightarrow P$ whenever it is guaranteed for P.

Rule `auth.choice` states that if each branch of a choice guarantees the authentication property T **`authenticates`** R, then so does the entire choice—since whenever some event from T occurs, it must have been performed by one of the arms of the choice, and that choice must previously have performed some event from R.

Rule `auth.parallel` states that if a single component P of a parallel combination is able to guarantee that T **`authenticates`** R, and it is involved in all occurrences of events from T and R, then that is enough to ensure that the entire parallel combination $P \,|[\, A \,]|\, Q$ guarantees it: since P will not allow any event from T to occur before an event from R occurs. There are no restrictions on the rest of the system Q, so the rule holds for any process description Q.

Rule `auth.interleaves` states that if both components of an interleaved combination can guarantee T **`authenticates`** R, then the combination itself can. This follows from the fact that if some event from T occurs, then it must have been performed by one of the component processes, which must have previously performed an event from R.

Finally, the rule `auth.recursion` for mutually recursive processes states that if the property T **`authenticates`** R is preserved by recursive calls—if each variable X_k **sat** T **`authenticates`** R then so does each function $F_k(\vec{X})$ applied to the variables—then the processes defined by the mutual recursion satisfy the property T **`authenticates`** R. This rule is a special case of the general rule for mutual recursion; for further details, see the discussion in [DaS93].

3.2 The CSP network description

The approach taken is to provide a CSP description of the Dolev-Yao model [DoY83]. Here it is assumed that the communications medium is entirely under the control of the enemy, which can block, re-address, duplicate, and fake messages. We will define a 'generates' relation $\vdash$ which describes when new messages may be derived from existing ones: $S \vdash m$ means that knowledge of all the messages in S is sufficient to produce m. This will be used to capture the enemy's ability to fake messages. In [Sch96] the roles of the passive medium and of the active enemy were described using distinct CSP processes which enabled the capabilities of the enemy to be separately described. For the purposes of this paper it is preferable to describe the

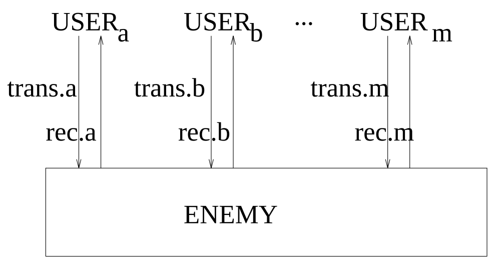

Figure 3 CSP model of the network

combination of the enemy and the medium as a single CSP process *ENEMY*, since this makes for an easier analysis.

There is a set *USER* consisting of the names of all the agents which use the network. For each $i \in USER$ we associate a process $USER_i$ which describes how user i behaves. Each process $USER_i$ communicates with *ENEMY* by means of a channel $trans.i$ on which it transmits messages, and a channel $rec.i$ on which it receives messages. Thus we have $\sigma(USER_i) \subseteq trans.i \cup rec.i$.

The resulting network is then described as follows:

$$NET \quad = \quad (|||_{j \in USER}\ USER_j) \,|[\, trans, rec \,]|\, ENEMY$$

This network is pictured in Figure 3.

In the analysis of a protocol we might consider a and b as the two parties involved in the protocol, and $USER_a$ and $USER_b$ will describe the respective roles that they play.

If there are actually no other users connected to the system, then we can have $USER_i = Stop$ for each $i \neq A, B$ and we obtain

$$NET \quad = \quad (USER_a \;|||\; USER_b) \,|[\, trans, rec \,]|\, ENEMY$$

We retain the names of other users in the set *USER* to retain the potential of the *ENEMY* to masquerade as a different user.

The channels *trans* and *rec* are of type *USER.USER.MESSAGE*. A message $trans.i.j.m$ should be thought of as node i sending a message m with destination j. Thus i is the source, j the destination, and m the message. The message space *MESSAGE* will generally be defined as an abstract data type, as in Section 4.1.2

Refusals have been abstracted away, in the sense that input can never be refused, which amounts to making the assumption that nothing can be deduced from how or when the messages are accepted. This is a reasonable assumption, since there are protocols currently in use to perform tasks such as masking network traffic. Hence at this level of abstraction we can assume that messages are always accepted by the network. (If this is later felt to be unrealistic, the definition can be altered accordingly, so that messages may not be input after the number of messages in the network reaches some capacity threshold). Attacks on protocols (apart from denial of service attacks) tend to exploit unexpected interactions between messages, and the traces model is adequate for capturing these.

3.2.1 Description of *ENEMY*

A natural CSP description of the Dolev-Yao model separates out an essentially passive medium *MEDIUM* from an intruder *INTRUDER* which is able to read messages from the medium (via channel *leak*) and to manipulate it by removing and adding messages (via channels *kill* and *add*). This approach was taken in [Sch96]. The passive medium containing a set T of messages to be delivered might be defined by

$$\begin{array}{rcl} \mathit{MEDIUM}(T) & \widehat{=} & \Box_{t \in T}\, rec!t \rightarrow \mathit{MEDIUM}(T \setminus \{t\}) \\ & & \Box \\ & & trans?i?j?m \rightarrow \mathit{MEDIUM}(T \cup \{j.i.m\}) \\ & & \Box \\ & & \Box_{t \in T}\, leak!t \rightarrow \mathit{MEDIUM}(T) \\ & & \Box \\ & & add?i?j?m \rightarrow \mathit{MEDIUM}(T \cup \{j.i.m\}) \\ & & \Box \\ & & kill?t \rightarrow \mathit{MEDIUM}(T \setminus \{t\}) \end{array}$$

The intruder in possession of information described by set U, who interacts with the medium through the channels *leak*, *add*, and *kill*, may be described as follows:

$$\begin{array}{rcl} \mathit{INTRUDER}(U) & \widehat{=} & leak?u \rightarrow \mathit{INTRUDER}(U \cup \{u\}) \\ & & \Box \\ & & \Box_{(U \vdash m), i, j}\, add!i.j.m \rightarrow \mathit{INTRUDER}(U) \\ & & \Box \\ & & \Box_{(U \vdash m), i, j}\, kill!i.j.m \rightarrow \mathit{INTRUDER}(U) \end{array}$$

The combination of medium and intruder, which provides the environment in which an authentication protocol is intended to operate, is described as

$$(\mathit{MEDIUM}(T) \,|[\, leak, add, kill \,]|\, \mathit{INTRUDER}(U)) \setminus \{leak, add, kill\}$$

Observe that the intruder can remove any message from the medium (by reading it along *leak* and then replaying it along *kill*), can replay any message, can redirect messages (by removing the original and then placing a copy with altered source and destination fields). These follow from the elementary properties of the $\vdash$ relation given in Section 4.1.2, and from the fact that source and destinations of a message can be removed: $\{i.j.m\} \vdash m$.

Initially, the medium would be empty, and the intruder would be in possession of some initial information *INIT*.

Although this description is the most natural, it may be considerably condensed to an equivalent form involving only one mutually recursive process ENEMY_0 with just two channels: *trans* and *rec*.

$$\begin{array}{rcll} \mathit{ENEMY}_0(S) & = & trans?i?j?m \rightarrow \mathit{ENEMY}_0(S \cup \{m\}) & (1) \\ & & \Box & \\ & & \Box_{i,j \in \mathit{USER}, S \vdash m}\, rec.i!j!m \rightarrow \mathit{ENEMY}_0(S) & (2) \\ & & \Box & \\ & & \sqcap_{m \in S}\, \mathit{ENEMY}_0(S \setminus \{m\}) & (3) \end{array}$$

It is possible to prove in the traces model that

$$\begin{aligned}(\mathit{MEDIUM}(T)\,|[\,\mathit{leak}, \mathit{add}, \mathit{kill}\,]|\,\mathit{INTRUDER}(U)) \setminus \{\mathit{leak}, \mathit{add}, \mathit{kill}\} \\ = \mathit{ENEMY}_0(T \cup U)\end{aligned}$$

The first two arms of the choice correspond to communications with the users: the enemy can accept messages from any user on *trans*, and can provide any user with any message that it can generate, together with any source, on *rec*. The third choice corresponds to a message being intercepted and not passed on.

In fact, one further simplification can be made within the traces model: refusals are not modelled, so the delay of messages is always possible. Delaying a message until after an attack has the same effect as killing it. In other words, if an attack relies on killing a message, then that message can in any case be delayed until after the attack. In this context the ability to delay messages arbitrarily is therefore as powerful as the ability to kill messages, so we can drop the nondeterministic choice of line 3. [1]

The following description has the same trace semantics as ENEMY_0 above. It is also the easiest to manipulate in proofs:

$$\begin{aligned}\mathit{ENEMY}(S) &= \mathit{trans}?i?j?m \rightarrow \mathit{ENEMY}(S \cup \{m\}) &(4)\\ &\quad \Box \\ &\quad \Box_{i,j \in \mathit{USER}, S \vdash m}\ \mathit{rec}.i!j!m \rightarrow \mathit{ENEMY}(S) &(5)\end{aligned}$$

We will use this last description through this paper: $\mathit{ENEMY} = \mathit{ENEMY}(\mathit{INIT})$.

If we wished to model the enemy as simply an eavesdropper with no power to add messages to the medium, then line 5 will be a choice of messages in S rather than those generated from S: $\Box_{i,j \in \mathit{USER}, m \vdash S}\ \mathit{rec}.i!j!m$.

The agents implementing the protocols place restrictions on the messages that may be passed on *trans*, which in turn restricts the possibilities for messages being passed on *rec*.

We are already in a position to prove that all messages passed on *rec* must be generable from the initial set *INIT* together with the messages input on *trans*:

Theorem 3.5 ENEMY **sat** $(\mathit{INIT} \cup (\mathit{tr} \Downarrow \mathit{trans})) \vdash \mathit{tr} \Downarrow \mathit{rec}$ $\Box$

The proof is given in Appendix A.1

3.2.2 Protocol participants

The agents a and b that are the participants in the protocol are modelled as USER_a and USER_b, consisting of CSP implementations of the two halves of the protocol. More generally, if there are other participants (such as trusted third parties) then their activity will also be described as CSP processes. Obviously their description will depend entirely on the protocol being modelled.

[1] Technically this is justified by the result that if $S \subseteq S'$ then $\mathit{traces}(\mathit{ENEMY}(S)) \subseteq \mathit{traces}(\mathit{ENEMY}(S'))$, and so $\mathit{traces}(\mathit{ENEMY}(S) \sqcap \mathit{ENEMY}(S')) = \mathit{traces}(\mathit{ENEMY}(S'))$.

3.2.3 Other users

One of the benefits of the CSP approach is that processes representing other users may be included in the analysis without the need to provide a precise description of what they do. It is possible to define a most general description of other users, and assume that their actual behaviour is a refinement of that description. It is of course necessary to have some confidence that this assumption is appropriate!

In general, other users may be considered to be refinements of $Un_i(S_i)$, where

$$\begin{array}{rcl} Un_i(S) & = & \Box_{j \in USER, S \vdash m}\, trans.i!j!m \rightarrow Un_i(S) \\ & & \Box \\ & & rec.i?j?m \rightarrow Un_i(S \cup \{m\}) \end{array}$$

The set S_i will include all the information $USER_i$ has initially available, such as the secret key s_i, various nonces, and so on. This approach is entirely similar to the way we model the enemy.

3.3 General proof strategy

3.3.1 Rank functions

In order for the proof to work, a rank function $\rho : MESSAGE \rightarrow \mathbb{Z}$ requires a number of key properties. These will be formalised in Theorem 3.9, but we list them here since they are the properties to keep in mind when attempting to construct a rank function.

R1: every message in $INIT$ has rank 1 or greater;

R2: if every message in a set S has rank $\geqslant 1$ and $S \vdash m$ then $\rho(m) \geqslant 1$;

R3: every message in T has rank 0 or less.

R4: Every user, when prevented from performing messages in R, is unable to introduce messages of rank 0 or less;

The rank function provides evidence that only messages of rank $\geqslant 1$ can circulate in $NET \,|[\, R \,]|\, Stop$, and hence that nothing in T can be generated.

Observe that the rank function is dependent on R and T, as well as the descriptions of the users. In other words, it is dependent on the protocol and on the particular property to be established.

ENEMY and rank

Theorem 3.5 together with a rank function yields the following corollary:

Corollary 3.6 If ρ meets conditions $R1$ and $R2$ then

$$ENEMY(INIT)\ \mathbf{sat}\ \rho(tr \downarrow trans) \geqslant 1 \Rightarrow \rho(tr \downarrow rec) \geqslant 1$$

$\Box$

Our security properties reduce to properties of messages passing on *trans* and *rec*.

If $\rho : MESSAGE \rightarrow \mathbb{Z}$ is a function defined on messages, then we lift it to sets S and sequences tr as follows:

$$\begin{aligned} \rho(S) &= \min\{\rho(s) \mid s \in S\} \\ \rho(tr) &= \min\{\rho(m) \mid c.m \in tr\} \end{aligned}$$

Users maintaining positive rank

We already have a result concerning $ENEMY$: that it is not able to generate messages of a rank lower than those already in its possession. The proof will proceed by introducing each $USER$ in turn and analysing its description. The key specification for each $USER_i$ is that it cannot introduce messages of rank 0: that if the messages it has received are all of strictly positive rank, then the messages it transmits must also be of positive rank. This is formally defined on traces as follows:

Definition 3.7

$$\texttt{maintains } \rho \texttt{ on } i \quad \widehat{=} \quad \rho(tr \upharpoonright trans.i) \geqslant \rho((tr \upharpoonright rec.i) \cup INIT)$$

□

Observe that this is parameterised by both the rank function ρ and the user name i.

This specification states that the rank of the messages transmitted on $trans.i$ is not less than both the rank of the messages received on $rec.i$ and the rank of the messages in $INIT$. If $USER_i$ meets this specification then it cannot introduce a message of rank 0 into the system (along channel $trans.i$) if there was not one already present (which was received along channel $rec.i$).

Other users and rank

The following inference rule formalises our assumption about the capabilities of the enemy:

Rule 3.8

$$\frac{\rho(S_i) \geqslant 1}{Un_i(S_i) \textbf{ sat } \texttt{maintains } \rho \texttt{ on } i}$$

□

Hence if $Un_i \sqsubseteq USER_i$ then $USER_i$ **sat** $\texttt{maintains } \rho \texttt{ on } i$.

Theorem 3.9 If

R1: $\rho(INIT) \geqslant 1$

R2: $\rho(S) \geqslant 1 \wedge S \vdash m \Rightarrow \rho(m) \geqslant 1$

R3: $\rho(T) \leqslant 0$

R4: $\forall\, i.(USER_i \,|[\, R \,]|\, Stop$ **sat** `maintains` ρ `on` $i)$

then $NET \,|[\, R \,]|\, Stop$ **sat** $tr \upharpoonright T = \langle\rangle$ $\square$

Figure 4 A theorem for *NET*

3.3.2 Proof obligations

We obtain an extremely specialised theorem that applies to authentication properties on this specific description *NET* of the network. This theorem is at the heart of the proof strategy presented in this paper. It provides a sufficient list of conditions whose achievement guarantees that *NET* **sat** *T* `authenticates` *R*. It is given in Figurenetwork2

Theorem 3.9 relies on the assumption that $\forall\, i.\sigma(USER_i) \cap (trans \cup rec) \subseteq trans.i \cup rec.i$: i.e. that no user except $USER_i$ has any interaction with *ENEMY* on *trans.i* or *rec.i*. This assumption is built into the way we are modelling the network.

As discussed earlier, authentication properties expressed in CSP are in the form '*T* authenticates *R*', meaning that if some event from *T* occurs then some event from *R* must previously have occurred. Generally *T* will be a set of possible inputs at a particular node, which provides evidence of the occurrence of another set of messages *R* at a different node.

The proof strategy we adopt is to show that if all occurrences of events *R* are prevented in *NET*, then the events in *T* is not possible. In other words, we aim to show that the messages in *T* cannot be generated by the resulting system description $NET \,|[\, R \,]|\, Stop$.

Having expressed this requirement, we aim to provide a *rank function* ρ which associates a value with each message in the message space. This function will depend on the generation relation $\vdash$, the description $USER_a$ and $USER_b$ of the protocol, and the sets *INIT*, *R* and *T*. It should have the property that the rank of every message in *T* is 0 or less. We then prove that every message possible in the restricted system $NET \,|[\, R \,]|\, Stop$ has rank 1 or greater, allowing the deduction that no message in *T* is possible for the system.

To apply Theorem 3.9 we must meet the list of requirements $R1$ to $R4$. Some of these correspond to assumptions which we must be confident can be made, and others are proof obligations.

Item $R1$ is an assumption on the enemy, and $R4$ is an assumption for users other than a and b. Confidence in these assumption will depend on the nature of ρ, and the messages that are of rank 0 or below.

Item $R2$ must be checked, and may be established by an induction over the definition of the generates relation $\vdash$. It therefore relies on the fact that the clauses defining $\vdash$ completely determine which messages can be generated from already known ones. Item $R3$ must be shown for the particular set T. Item $R4$ must be proven for the cases $(R4a) : USER_a \,|[\, R \,]|\, Stop$ and $(R4b) : USER_b \,|[\, R \,]|\, Stop$. It is assumed for all other users.

3.3.3 Specialised CSP proof rules

There are also a number of rules which can be given concerning the relationship between various CSP operators and the `maintains` ρ `on` i specification. These are given in Figure 5. They are all sound with respect to the CSP traces model. Together with the equations of Figure 1 they will be used in establishing $R4a$ and $R4b$ in various cases.

Informally, their soundness can be justified as follows. Rule `stop` is sound because *Stop* is unable to violate `maintains` ρ `on` i since to do so requires an output of a message of non-positive rank, and *Stop* can perform no such output. Rule `output` states that if the first output provided by a process has positive rank, then the process satisfies `maintains` ρ `on` i provided the behaviour after this first output does not violate it.

Rule `input` is concerned with the behaviour of a process subsequent to an input. The requirement to maintain positive rank is concerned that if messages coming in have positive rank, then the messages going out should also have positive rank. For a particular incoming message, there are therefore two cases to consider: if the input message $f(x)$ has rank 0 or less, then the subsequent behaviour is irrelevant since responsibility for maintaining positive rank is no longer required; if the message $f(x)$ input has positive rank, then the subsequent process $P(j, x)$ should maintain positive rank. Hence the rule states that the input process $rec.i?j?f(x) \rightarrow P(j, x)$ satisfies `maintains` ρ `on` i whenever $P(j, x)$ does so after an input of positive rank. The form of the input $f(x)$ describes the pattern matching implicit in the input process: f describes the input patterns allowed.

Finally, rule `choice` states that if each branch of a choice maintains positive rank, then so does the entire choice.

4 Modelling the Needham-Schroeder protocol

4.1 CSP model

The description of the Needham-Schroeder public key protocol [NeS78] is often slimmed down to the following:

$$\begin{aligned} A \rightarrow B &: p_b(n_a.a) \\ B \rightarrow A &: p_a(n_a.n_b) \\ A \rightarrow B &: p_b(n_b) \end{aligned}$$

This is illustrated in Figure 6. For each message there is a point si where it is sent and a point ri where it is received.

This version assumes that the public keys of A and B are already known to each other. The full version also involves communication between the parties and a trusted server to obtain the public keys.

Rule `stop`

$$\frac{}{\mathit{Stop}\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}$$

Rule `output`

$$\frac{P\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}{\mathit{trans}.i.j.m \to P\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}\quad [\,\rho(m) \geqslant 1\,]$$

Rule `input`

$$\frac{\forall j, x.(\rho(f(x)) \geqslant 1 \Rightarrow (P(j,x)\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i))}{\mathit{rec}.i?j?f(x) \to P(j,x)\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}$$

Rule `choice`

$$\frac{\forall j. V_j\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}{\Box_j\ V_j\ \textbf{sat}\ \texttt{maintains}\ \rho\ \texttt{on}\ i}$$

Figure 5 Proof rules for " `maintains` ρ `on` i"

4.1.1 Protocol participants

In CSP, the agents a and b which implement a run of the protocol may be modelled as in Figure 7. In this section, for the purposes of illustrating the approach, we simplify the analysis by considering only the case where a is the initiator and b is the receiver. This model does not allow for attacks where both parties act as initiators, or as senders. This assumption will be relaxed in Section 6, where each party can independently play either role.

4.1.2 Message space

The message space we use for anaylsis of this protocol is as follows:

$$\begin{aligned} \mathit{RAW} &:= \mathit{USER} \mid \mathit{TEXT} \mid \mathit{NONCE} \mid \mathit{KEY} \\ \mathit{MESSAGE} &:= \mathit{RAW} \mid \mathit{KEY}(\mathit{MESSAGE}) \mid \mathit{MESSAGE}.\mathit{MESSAGE} \end{aligned}$$

In fact for this example using public key cryptography, the space KEY will split into public keys PUBLIC and secret keys SECRET, one of each for each user in USER:

$$\mathit{KEY} ::= \mathit{PUBLIC} \mid \mathit{SECRET}$$

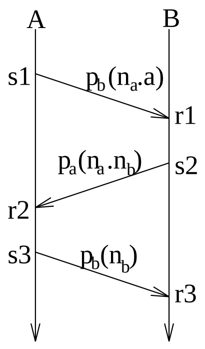

Figure 6 The Needham-Schroeder public key protocol

$$\begin{aligned} USER_a \quad = \quad & \Box_{i \in USER}\ trans.a!i!p_i(n_a.a) \rightarrow \\ & \qquad rec.a.i?p_a(n_a.x) \rightarrow \\ & \qquad\quad trans.a!i!p_i(x) \rightarrow Stop \end{aligned} \tag{6}$$

$$\begin{aligned} USER_b \quad = \quad & rec.b?j?p_b(y.j) \rightarrow \\ & \quad trans.b!j!p_j(y.n_b) \rightarrow \\ & \qquad rec.b.j.p_b(n_b) \rightarrow Stop \end{aligned} \tag{7}$$

Figure 7 A CSP description of the Needham-Schroeder protocol

We have rules concerning the way messages may be generated from existing ones:

A1 If $m \in S$ then $S \vdash m$

A2 If $S \vdash m$ and $S \subseteq S'$ then $S' \vdash m$

A3 If $S \vdash m_i$ for each $m_i \in S'$ and $S' \vdash m$ then $S \vdash m$

M1 $S \vdash m \wedge S \vdash k \Rightarrow S \vdash k(m)$

M2 $S \vdash m_1 \wedge S \vdash m_2 \Leftrightarrow S \vdash m_1.m_2$

K1 $\{p_i(s_i(m))\} \vdash m$

K2 $\{s_i(p_i(m))\} \vdash m$

Equations

We might also have equations on the message space. Some natural ones would be those describing the relationship between encryption and decryption:

$$E1 \qquad p_a(s_a(m)) = s_a(p_a(m)) = m$$

This would remove the need for $K1$ and $K2$, since they reduce under the equality to $\{m\} \vdash m$ which is already covered by $A1$

There would also be properties such as associativity of catenation:

$$E2 \qquad m_1.(m_2.m_3) = (m_1.m_2).m_3$$

Equations could also capture possible properties of encryption mechanisms. For example, encryption distributing over catenation would be quite a significant weakness. Commutativity of encryption is sometimes a necessary property.

$$\begin{array}{ll} E3 & k(m_1.m_2) = k(m_1).k(m_2) \\ E4 & k_1(k_2(m)) = k_2(k_1(m)) \end{array}$$

We will assume for the rest of this paper that $(E1)$ and $(E2)$ are the only equations on the message space unless explicitly stated otherwise, and that $A1$—$A3$ and $M1$—$M2$ define the relation $\vdash$.

It is important to ensure that any equations on the message space are respected by the definition of ρ, since otherwise it will not be well-defined. We expect that

$$\text{R0: } m_1 = m_2 \Rightarrow \rho(m_1) = \rho(m_2)$$

for any messages m_1 and m_2. If ρ is defined by induction on the structure of messages then $R0$ will be a proof obligation.

4.1.3 Issues

There are a number of issues concerning the way the protocol has been modelled. The use of pattern matching on message input corresponds to the assumption that any message that fails to match the pattern would be ignored, though we are modelling this as blocking receipt rather than accepting and throwing it away: the $USER_i$ processes simply do not engage in any message which does not match the pattern. In practice, this is unlikely to be achievable, especially in cases where an agent must decrypt a message before finding out if it is of a particular form. However, it does not affect the ability of the enemy to attack protocols described in this way.

Typing of messages is also implicit in pattern matching. The type of the channel determines the range of possible messages that might match the pattern. The permissible input for the code fragment $rec.a.i.p_a(n_a.x) \rightarrow \ldots$ described in line 6 of the protocol in Figure 7 depends on the type of x. A correct run of the protocol would have x as a nonce, but without the ability to type messages the input could accept an arbitrary message for x. As we shall see at the end of Section 5.1.2 the type of such an input might affect correctness of the protocol. For the purposes of this paper, we will assume that inputs defined by pattern matching must also conform to the expected type. This amounts to assuming that it is not possible for a message of one type to be mistaken for a message of another type.

In line 6 the pattern matching amounts to any message of the form $p_a(n_a.x)$ being accepted for any nonce x, whatever the claimed origin. This relies on the assumption that $USER_a$ can remember the nonce he sent out, and associate that with b, since the j received is just ignored and the response sent to b. This is a reasonable assumption, since a would expect to remember the nonce anyway in order to check that the response is correct.

The apparent source j of b's first message (line 7) must match the source given in the message itself, since that information is the only information b has concerning the originator of the protocol.

5 Analysis of the Needham-Schroeder protocol

5.1 Verification of origin authentication

Authentication of the origin is authentication for b: that a message received by b authenticates that an earlier message in a occurred.

The property of user b authenticating user a requires that if b engages in the protocol as if a is the initiator, then a is indeed the initiator.

Hence we are interested only in those executions of b which begin with some message of the form $rec.b.a.m$, where the source of message m appears to be a. The description required for the analysis is the description $USER_b$ under the restriction that the first message has a as source. This results in the following:

$$\begin{aligned} USER_b \quad = \quad & rec.b.a?p_b(y.a) \rightarrow \\ & \quad trans.b!a!p_a(y.n_b) \rightarrow \\ & \qquad rec.b.a.p_b(n_b) \rightarrow Stop \end{aligned}$$

5.1.1 Flavours of origin authentication property

Different messages of a to be authenticated correspond to different flavours of authentication property. We will prove properties (4) and (7). We will also discuss why the proof does not extend to property (5), which is shown not to hold in [Low95].

If the authenticating message is b's last one $rec.b.a.p_b(n_b)$, then there are a number of possibilities as to what this message might authenticate:

1. "$rec.b.a.p_b(n_b)$ **authenticates** $trans.a.b.p_b(n_a.a)$" corresponds to the requirement that occurrence of b's final message guarantees that a initiated the protocol run with b, with the nonce n_a.

2. "$rec.b.a.p_b(n_b)$ **authenticates** $trans.a.b.p_b(NONCE.a)$" corresponds to the requirement that on occurrence of b's last message it is guaranteed that a initiated the protocol run with b, but possibly with a different nonce. Lemma 3.2 yields that this is implied by property (1).

3. "$rec.b.a.p_b(n_b)$ **authenticates** $\{trans.a.j.p_j(NONCE.a) \mid j \in USER\}$" corresponds to the requirement that on occurrence of b's last message it is guaranteed that a initiated a protocol run, but possibly with a different user and nonce. Lemma 3.2 yields that this is implied by property (2), and hence by property (1).

4. "$rec.b.a.p_b(n_b)$ **authenticates** $rec.a.USER.p_a(NONCE.n_b)$" confirms only that a received b's nonce challenge. It does not confirm that a initiated the run with b.

5. "$rec.b.a.p_b(n_b)$ **authenticates** $trans.a.b.p_b(n_b)$" confirms that a responded to b's nonce challenge.

6. "$rec.b.a.p_b(n_b)$ **authenticates** $trans.a.b.p_b(NONCE)$" authenticates that a responded to some challenge believed to come from b, but not necessarily with the nonce that b issued. It follows from Lemma 3.2 that this is implied by property (5).

7. "$rec.b.a.p_b(n_b)$ **authenticates** $\{trans.a.j.p_j(n_b) \mid j \in USER\}$" authenticates that a responded to b's nonce challenge, but does not necessarily associate it with b. It follows from Lemma 3.2 that this is also implied by property (5).

8. "$rec.b.a.p_b(n_b)$ **authenticates** $\{trans.a.j.p_j(NONCE) \mid j \in USER\}$" authenticates only that a responded to some nonce challenge—in other words, it verifies that a is live. It follows from Lemma 3.2 that this is implied by property (7) and hence by property (5).

5.1.2 Proof of authentication property (4)

To introduce the approach, we will first address one of the easier properties, property (4): that $rec.a.USER.p_a(NONCE.n_b)$ is authenticated by $rec.b.a.p_b(n_b)$:

$$NET \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop \quad \textbf{sat} \quad tr \upharpoonright rec.b.a.p_b(n_b) = \langle\rangle$$

In order to use Theorem 3.9 we must provide a rank function ρ such that $R0$—$R3$ hold, and also $USER_a \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop$ **sat** **maintains** ρ **on** a and $USER_b \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop$ **sat** **maintains** ρ **on** b. (Observe that $USER_b \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop = USER_b$.)

We define a rank function $\rho : MESSAGE \rightarrow \mathbb{Z}$ on messages. This is given in Figure 8.

This rank function was in fact dreamed up by considering the various constraints that it is required to satisfy. The author began by assigning rank 1 to all messages required to have positive rank, and rank 0 to all messages required to have non-positive rank.

1. All of the users identities, and their public keys are assumed to be known, and hence assigned rank 1.

2. We also assume that all secret keys are known apart from those of a and b: this amounts to the assumption that all other agents are really controlled by the enemy. Hence rank 1 is assigned to all secret keys apart from s_a and s_b which are assigned rank 0.

3. We also require that $\rho(p_b(n_b)) \leqslant 0$, in order to meet $R3$. This means that we must have $\rho(n_b) \leqslant 0$, since the enemy can generate $p_b(n_b)$ from n_b since the public key p_b is publically available.

4. On the other hand, $p_a(n.n_b)$ must have positive rank, since it is actually broadcast during the protocol run. This suggests incrementing the rank when encrypting with p_a. Since $s_a(p_a(m)) = m$ we require that encrypting with s_a should decrement the rank.

These considerations proved sufficient to produce the satisfactory rank function given in Figure 8. The function indicates that a and b's secret keys are not known to the enemy, and that neither is the challenge n_b which b sends to a. For the purposes of this authentication property, the nonce n_a does not need to be secret: this is because the property is concerned only with b's nonce challenge.

R0

It is immediate from the definition of ρ that equations $E1$ and $E2$ both hold: $\rho(p_i(s_i(m))) = \rho(s_i(p_i(m))) = m$ and $\rho(m_1.(m_2.m_3)) = \rho((m_1.m_2).m_3)$.

$$
\begin{aligned}
\rho_0(u) &= 1 \\
\rho_0(t) &= 1 \\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_b \\ 1 & \text{otherwise} \end{cases} \\
\rho_0(p_i) &= 1 \\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = a \text{ or } i = b \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\rho(r) &= \rho_0(r) \\
\rho(p_j(m)) &= \begin{cases} \rho(m) + 1 & \text{if } j = a \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(s_j(m)) &= \begin{cases} \rho(m) - 1 & \text{if } j = a \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}
$$

Figure 8 Rank function for verification of Property (4)

R1

We assume that $\rho(INIT) = 1$. This is a reasonable assumption, amounting to the claim that the enemy does not initially have access to s_a, s_b, or n_b.

R2

We obtain the following result from a consideration of each clause of the definition of $\vdash$. The result follows from the fact that $\vdash$ is defined to be the smallest relation closed under all of the clauses.

$$\rho(S) > 0 \wedge S \vdash m \Rightarrow \rho(m) \geqslant \rho(S)$$

R3

Observe that $\rho(p_b(n_b)) = 0$.

R4a

Rules `restrict.4` and `restrict.5` yield that

$$
\begin{aligned}
&USER_a \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop = \\
&\quad \square_{i \in USER} \; trans.a!i!p_i(n_a.a) \rightarrow \\
&\qquad\qquad rec.a.i?p_a(n_a.x) : p_a(n_a.(NONCE \setminus \{n_b\})) \rightarrow \\
&\qquad\qquad\quad trans.a!i!p_i(x) \rightarrow Stop
\end{aligned}
$$

We will analyse this description by building it up in stages.

Firstly, if $x \in NONCE \setminus \{n_b\}$ then $\rho(p_i(x)) \geqslant 1$ for any i and so Rule **output** and Rule **stop** yield that

$$trans.a!i!p_i(x) \rightarrow Stop \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a$$

Now $\rho(p_a(n_a.x)) \geqslant 1 \wedge x \in NONCE \setminus \{n_b\} \Rightarrow \rho(p_i(x)) \geqslant 1$, so Rule **input** gives

$$\begin{array}{l} rec.a.i?p_a(n_a.x) : p_a(n_a.(NONCE \setminus \{n_b\})) \rightarrow \\ \quad trans.a!i!p_i(x) \rightarrow Stop \qquad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a \end{array} \tag{8}$$

for any i

Now another application of Rule **output** (since $\rho(p_i(n_a.a)) \geqslant 1$ for any i) yields for any i that

$$\begin{array}{l} trans.a!i!p_i(n_a.a) \rightarrow \\ \quad rec.a.i?p_a(n_a.x) : p_a(n_a.(NONCE \setminus \{n_b\})) \rightarrow \\ \quad\quad trans.a!i!p_i(x) \rightarrow Stop \qquad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a \end{array} \tag{9}$$

and so finally from Rule **choice** we deduce

$$USER_a \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]| \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a$$

as required.

R4b

Rule **stop** and Rule **input** (with vacuous antecedent) yield that

$$rec.b.a.p_b(n_b) \rightarrow Stop \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } b \tag{10}$$

Now since $\rho(p_b(y.a)) \geqslant 1 \Rightarrow \rho(y) \geqslant 1 \Rightarrow \rho(p_a(y.n_b)) \geqslant 1$ we obtain from Rule **output** and line 10 that

$$\begin{array}{l} \rho(p_b(y.a)) \geqslant 1 \Rightarrow \\ \quad trans.b!a!p_a(y.n_b) \rightarrow rec.b.a.p_b(n_b) \rightarrow Stop \; \textbf{sat} \; \texttt{maintains } \rho \texttt{ on } b \end{array} \tag{11}$$

This provides the antecedent for Rule **input** to yield

$$\begin{array}{l} rec.b.a?p_b(y.a) \rightarrow trans.b!a!p_a(y.n_b) \rightarrow rec.b.a.p_b(n_b) \rightarrow Stop \\ \qquad\qquad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } b \end{array}$$

or in other words $USER_b$ **sat** `maintains` ρ `on` b, as required.

This completes the proof that $rec.b.a.p_b(n_b)$ authenticates $rec.a.USER.p_a(NONCE.n_b)$

The properties of ρ that were used in the proof, as side-conditions to the proof rules, were

1. $\rho(p_a(n_a.x)) \geqslant 1 \wedge x \neq n_b \wedge x \in NONCE \Rightarrow \rho(p_i(x)) \geqslant 1$ for any i, (used to obtain line 8)
2. $\rho(p_i(n_a.a)) \geqslant 1$ for all i (used to obtain line 9)

3. $\rho(y) \geqslant 1 \Rightarrow \rho(p_a(y.n_b)) \geqslant 1$ (used to obtain line 11)
4. $\rho(INIT) \geqslant 1$ (required for $R1$)
5. $\rho(p_b(n_b)) = 0$ (required for $R3$)

This has proven that b's receipt of n_b encrypted under his own public key provides evidence that a received the same nonce encrypted under her public key in some message of the form $rec.a.j.p_a(n.n_b)$ for some j and n.

- ◇ The proof works even if $\rho(s_b) = 1$. This means that a can be authenticated to b even if b's own secret key is compromised. This makes sense, since the fact that s_b is known only to b is really required by a rather than by b.
- ◇ If equation $E3$ holds on the message space, the function ρ remains well-defined; i.e. $\rho(k(m_1, m_2) = \rho(k(m_1)).\rho(k(m_2))$. In other words, even though the enemy might be able to alter the message $p_a(n_a.n_b)$ sent by a to a different message such as $p_a(n_a.n_b')$ this capability does not affect the authentication property.
- ◇ The proof works even in the presence of commutativity of encryption: the equation $E4$ is also respected by the rank function.

5.1.3 An aside on typing

The proof above relies on the fact that messages are *typed*: the pattern matching built into the analysis of $USER_a \,|[\, rec.a.USER.p_a(NONCE.n_b) \,]|\, Stop$ assumes that all inputs on $rec.a$ will be of the form $i.p_a(NONCE.NONCE)$. In other words, the description of $USER_a$ assumes that the message $i.p_a(n_a.x)$ will be rejected unless x is a nonce.

In fact, the minimal assumption required is simply that user a is able to recognise a message of the form $p_a(n_a.x)$ for arbitrary messages x, and to extract the message x from this. As we shall see, we will be able to retain the result even under this weakening, though at some cost.

The property "$rec.b.a.p_b(n_b)$ `authenticates` $rec.a.USER.p_a(MESSAGE.n_b)$" is the appropriate version of property (4) for verify for NET in the absence of types, since the first message received by user b is of the form $p_b(y.a)$ for an arbitrary message y. The description of $USER_a$ to be analysed becomes

$$\begin{array}{l}USER_a \,|[\, rec.a.USER.p_a(MESSAGE.n_b) \,]|\, Stop = \\ \quad \Box_{i \in USER} \; trans.a!i!p_i(n_a.a) \rightarrow \\ \qquad\qquad rec.a.i?p_a(n_a.x) : p_a(MESSAGE \setminus (MESSAGE.n_b)) \rightarrow \\ \qquad\qquad\quad trans.a!i!p_i(x) \rightarrow Stop\end{array}$$

The rank function of Figure 8 is no longer appropriate, in that this process does not satisfy `maintains` ρ on a for that ρ. In particular, we require that ρ satisfies a generalisation of side-condition (1) listed above, that

$$\left.\begin{array}{l}\rho(p_a(n_a.x)) \geqslant 1 \\ \wedge\, p_a(n_a.x) \in (MESSAGE \setminus (MESSAGE.n_b))\end{array}\right\} \Rightarrow \rho(p_i(x)) \geqslant 1 \qquad (12)$$

However, this is not true for the ρ defined in Figure 8. For example, if n_c is some other nonce, then $x = n_b.n_c$ provides a counterexample.

$$\begin{aligned}
\rho_0(u) &= 1\\
\rho_0(t) &= 1\\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_b\\ 1 & \text{otherwise}\end{cases}\\
\rho_0(p_i) &= 1\\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = A \text{ or } i = B\\ 1 & \text{otherwise}\end{cases}
\end{aligned}$$

$$\begin{aligned}
\rho(r) &= \rho_0(r)\\
\rho(p_j(m)) &= \begin{cases} \rho(m) + 1 & \text{if } j = a \text{ and } m = y.n_b \text{ for some } y\\ \rho(m) & \text{otherwise}\end{cases}\\
\rho(s_j(m)) &= \begin{cases} \rho(m) - 1 & \text{if } j = a \text{ and } m = p_a(y.n_b) \text{ for some } y\\ \rho(m) & \text{otherwise}\end{cases}\\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}$$

Figure 9 New rank function for verification of Property (4)

This counterexample is excluded by the typing mechanism, which forces x to be some nonce other than n_b. Under this restriction on x line 12 is true.

The description of $USER_b$ is unchanged:

$$\begin{aligned}
USER_b \quad = \quad & rec.b.a?p_b(y.a) \rightarrow\\
& \quad trans.b!a!p_a(y.n_b) \rightarrow\\
& \qquad rec.b.a.p_b(n_b) \rightarrow Stop
\end{aligned}$$

though y ranges over $MESSAGE$ rather than $NONCE$.

A fresh rank function suitable for establishing $R4a$ and $R4b$ is given in Figure 9. It is also readily checked that it meets $R0$—$R3$.

5.1.4 A weak encryption mechanism

We observed that the rank function in Figure 8 respected the equations $E3$ and $E4$ on the message space, and hence that the verifications using that rank function remained correct even in the presence of those extra equations.

However, the rank function in Figure 9 does not respect either of these equations. This means that it is not well-defined on the message space in their presence. We may try to adapt the definition of the rank function: success would indicate that the authentication property held even in the presence of these extra equations. Alternatively, difficulty in adapting the definition might lead to the discovery of an attack which exploits one of the equations. This occurs in the case of $E3$, where the following trace of $NET \,|[\, rec.a.USER.p_a(MESSAGE.n_b) \,]|\, Stop$ (and thus of NET) arises. This trace contains the authenticating message $rec.a.b.p_b(n_b)$ but no message of the form $rec.a.USER.p_a(MESSAGE.n_b)$:

$$\begin{aligned}
&\langle trans.a.b.p_b(n_a.a), rec.b.a.p_b(n_a.a), trans.b.a.p_a(n_a.n_b),\\
&\quad rec.a.b.p_a(n_a.n_b.n_c), trans.a.b.p_b(n_b.n_c), rec.a.b.p_b(n_b)\rangle
\end{aligned}$$

This corresponds to the following attack:

$$\begin{array}{lcl} A \rightarrow B & : & p_b(n_a.a) \\ B \rightarrow I(A) & : & p_a(n_a.n_b) \\ I(B) \rightarrow A & : & p_a(n_a.n_b.n_c) \\ A \rightarrow I(B) & : & p_b(n_b.n_c) \\ I(A) \rightarrow B & : & p_b(n_b) \end{array}$$

This trace establishes the possibility that a takes $n_b.n_c$ to be b's nonce challenge, and so b's completion of the protocol does not provide authentication that a received b's nonce challenge b.

This attack may be regarded as implausible because even in the absence of typing a nonce may be known to contain a particular number of bits. However, even if this form of 'partial typing' allows some types to be distinguished (e.g. $NONCE$ from $NONCE.NONCE$), other types may still be confused. If we allow confusion only between $KEY(NONCE)$ and $NONCE$ (which is plausible since both are strings of bits of a particular length), then the following trace is possible under $E3$ and $E4$:

$$\langle trans.a.b.p_b(n_a.a), rec.b.a.p_b(n_a.a), trans.b.a.p_a(n_a.n_b),$$
$$rec.a.b.p_a(n_a.p_e(n_b)), trans.a.b.p_b(p_e(n_b)), rec.a.b.p_b(n_b)\rangle$$

This corresponds to the following attack:

$$\begin{array}{lcl} A \rightarrow B & : & p_b(n_a.a) \\ B \rightarrow I(A) & : & p_a(n_a.n_b) \\ I(B) \rightarrow A & : & p_a(n_a.p_e(n_b)) \\ A \rightarrow I(B) & : & p_b(p_e(n_b)) \\ I(A) \rightarrow B & : & p_b(n_b) \end{array}$$

Even though the enemy cannot obtain the nonce challenge n_b directly, commutativity and distributivity of encryption allow the enemy to alter the nonce challenge issued by b. Again, b's completion of the protocol run does not guarantee that a received the nonce challenge in the form transmitted by b.

These attacks are not possible under the assumption of full typing of messages: the fact that the rank function of Figure 8 respects $E3$ and $E4$ is enough to guarantee that there can be no such attacks. Both attacks above fail in the presence of full typing at the point where a should respond to b's nonce challenge; a detects a type mismatch and blocks further progress.

5.1.5 Proof of authentication property (7)

It is more usual for receipt of a message to authenticate transmission of a related message. We next turn our attention to verifying that b's completion of the protocol authenticates that a responded to the nonce challenge: in other words, that $rec.b.a.p_b(n_b)$ `authenticates` $\{trans.a.i.p_i(n_b) \mid i \in USER\}$. In this case the analysis begins with

$$\begin{array}{l} USER_a \,|[\{trans.a.i.p_i(n_b) \mid i \in USER\}]|\, Stop = \\ \quad \square_{i \in USER} \; trans.a!i!p_i(n_a.a) \rightarrow \\ \qquad\qquad rec.a.i?p_a(n_a.x) \rightarrow \\ \qquad\qquad\quad \left(\begin{array}{ll} trans.a!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_b \\ Stop & \text{if } x = n_b \end{array} \right) \end{array}$$

The rank function ρ defined in Figure 8 is again suitable. We have already established properties $R0$—$R3$ described for it, and the description of $USER_b$ is unchanged, so we also have $R4b$. It remains only to prove $R4a$.

R4a

For a given $x \in NONCE$, either $x \neq n_b$ or $x = n_b$.

In the first case it follows that $\rho(x) \geqslant 1$ and so $\rho(p_i(x)) \geqslant 1$ (for any i), and hence that $trans.a!i!p_i(x) \rightarrow Stop$ **sat** `maintains` ρ on a.

For the second case it is immediate that $Stop$ **sat** `maintains` ρ on a.

Hence we have that

$$\left(\begin{array}{ll} trans.a!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_b \\ Stop & \text{if } x = n_b \end{array}\right) \quad \textbf{sat} \quad \texttt{maintains } \rho \text{ on } a$$

This provides the antecedent for Rule `input` to yield

$$\begin{array}{l} rec.a.i?p_a(n_a.x) \rightarrow \\ \quad \left(\begin{array}{ll} trans.a!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_b \\ Stop & \text{if } x = n_b \end{array}\right) \end{array} \quad \textbf{sat} \quad \texttt{maintains } \rho \text{ on } a$$

And so from Rule `output` (since $\rho(p_i(n_a.a)) \geqslant 1$) we have for any i that

$$\begin{array}{l} trans.a!i!p_i(n_a.a) \rightarrow \\ \quad rec.a.i?p_a(n_a.x) \rightarrow \\ \quad \left(\begin{array}{ll} trans.a!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_b \\ Stop & \text{if } x = n_b \end{array}\right) \end{array} \quad \textbf{sat} \quad \texttt{maintains } \rho \text{ on } a$$

Finally we obtain from Rule `choice` that

$$\Box_{i \in USER} \begin{array}{l} trans.a!i!p_i(n_a.a) \rightarrow \\ \quad rec.a.i?p_a(n_a.x) \rightarrow \\ \quad \left(\begin{array}{ll} trans.a!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_b \\ Stop & \text{if } x = n_b \end{array}\right) \end{array} \quad \textbf{sat} \quad \texttt{maintains } \rho \text{ on } a$$

which completes the proof.

5.1.6 Failure of authentication property (5)

The previous verification shows that the protocol supports authentication that a responded to b's nonce challenge, but to some arbitrary user rather than specifically to b. The property "$rec.b.a.p_b(n_b)$ `authenticates` $trans.a.b.p_b(n_b)$ captures this stronger requirement. However, Lowe [Low95] has established that the protocol does not establish this requirement. It is instructive to see how this manifests itself in our framework.

Observe that the rank function in Figure 8 is inappropriate for $rec.b.a.p_b(n_b)$ authenticating $trans.a.b.p_b(n_b)$, since it is possible for $USER_a$ to transmit messages of rank 0, of the form $trans.a.i.p_i(x)$ for $i \neq b$, having received messages only of rank $\geqslant 1$. In other words, the process $USER_a \, |[\, trans.a.b.p_b(n_b) \,]| \, Stop$ does not satisfy `maintains` ρ on a. This tells us that the proofs above do not extend to cover this case. In itself it does not tell us that the protocol does not meet the authentication property, but the exercise of attempting to define a rank function may suggest an attack, since it highlights those messages which the enemy can generate which break the authentication property.

Lowe's attack corresponds to the following trace of $NET \,|[\, trans.a.b.p_b(n_b) \,]|\, Stop$ (and hence of NET):

$$\begin{array}{l}\langle trans.a.c.p_c(n_a.a), rec.b.a.p_b(n_a.a), trans.b.a.p_a(n_a.n_b) \\ \quad rec.a.c.p_a(n_a.n_b), trans.a.c.p_c(n_b), rec.b.a.p_b(n_b)\rangle\end{array}$$

which contains the authenticating message $rec.b.a.p_b(n_b)$ but not $trans.a.b.p_b(n_b)$, the message which is to be authenticated. This corresponds to the following attack, where user a's legitimate initiation of the protocol with c is used (by c, or equivalently by the enemy in possession of c's keys) to initiate a protocol run with b where b acts as if the other partner is a:

$$\begin{array}{lcl}A \rightarrow I & : & p_c(n_a.a) \\ I(A) \rightarrow B & : & p_b(n_a.a) \\ B \rightarrow I(A) & : & p_a(n_a.n_b) \\ I \rightarrow A & : & p_a(n_a.n_b) \\ A \rightarrow I & : & p_c(n_b) \\ I(A) \rightarrow B & : & p_b(n_b)\end{array}$$

Hence there can be no rank function ρ which meets $R1$—$R4$.

5.2 Verification of responder authentication

Authentication requirements commonly run in both directions: both the originator and the responder require some form of confidence about the origin of the messages they are receiving. We have analysed verification of the originator to some depth in the preceding sections. For the sake of completeness we will now examine authentication of the responder. In fact, the approach taken is the same as that taken above.

For authentication of the responder, the analysis is in the context of the first message of $USER_a$ having b as its destination. Hence the description of $USER_a$ will be that of the original description of Section 4.1.1 but with the first message restricted:

$$\begin{array}{lcl}USER_a & = & trans.a!b!p_b(n_a.a) \rightarrow \\ & & \quad rec.a.b?p_a(n_a.x) \rightarrow \\ & & \qquad trans.a!b!p_b(x) \rightarrow Stop\end{array}$$

The agent $USER_b$ has the original description of Figure 7.

We aim to verify "$rec.a.b.p_a(n_a.NONCE)$ **authenticates** $trans.b.a.p_a(n_a.NONCE)$". In other words, we must prove that

$$NET \,|[\, trans.b.a.p_a(n_a.NONCE) \,]|\, Stop \quad \textbf{sat} \quad tr \upharpoonright rec.a.b.p_a(n_a.NONCE) \qquad (13)$$

The rank function required for this verification is defined in Figure 10. The properties $R0$—$R3$ can easily be checked for this rank function. It remains to establish only $R4a$ and $R4b$.

$$\begin{aligned}
\rho_0(u) &= 1 \\
\rho_0(t) &= 1 \\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_a \\ 1 & \text{otherwise} \end{cases} \\
\rho_0(p_i) &= 1 \\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = a \text{ or } i = b \\ 1 & \text{otherwise} \end{cases} \\
\\
\rho(r) &= \rho_0(r) \\
\rho(p_j(m)) &= \begin{cases} 1 & \text{if } j = b \text{ and } m = n_a.a \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(s_j(m)) &= \begin{cases} 0 & \text{if } j = b \text{ and } m = p_b(n_a.a) \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}$$

Figure 10 Rank function for authentication of b to a

R4a

Observe that $USER_a \,|[\, trans.b.a.p_a(n_a.NONCE) \,]|\, Stop = USER_a$. Hence we aim to prove that $USER_a$ **sat** `maintains` ρ `on` a.

Firstly we see that $\rho(p_a(n_a.x)) = 0$ for any x, so it is vacuous that $\rho(p_a(n_a.x)) \geqslant 1 \Rightarrow \rho(p_i(x)) \geqslant 1$. Hence we deduce from an application of Rule `input`, Rule `output`, and Rule `stop` that

$$rec.a?j?p_a(n_a.x) \to trans.a!b!p_i(x) \to Stop \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ a$$

Finally, since $\rho(p_b(n_a.a)) = 1$, we may use Rule `output` to obtain

$$USER_a \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ a$$

as required.

R4b

The process to be analysed is

$$\begin{aligned}
& USER_b \,|[\, trans.b.a.p_a(n_a.NONCE) \,]|\, Stop \\
&\quad = \; rec.b?j?p_b(y.j) \to \\
&\qquad \begin{pmatrix} Stop & \text{if } y = n_a \text{ and } j = a \\ trans.b!j!p_j(y.n_b) \to rec.b?i.p_b(n_b) \to Stop & \text{otherwise} \end{pmatrix}
\end{aligned}$$

If $\rho(p_b(y.j)) \geqslant 1$ then either $y = n_a$ and $j = a$ or else $\rho(y.j) \geqslant 1$. In the second case, we obtain $\rho(p_j(y.n_b)) \geqslant 1$, and so

$$trans.b!j!p_j(y.n_b) \to rec.b?i.p_b(n_b) \to Stop \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ b$$

Hence we obtain that if $\rho(p_b(y.j)) \geqslant 1$ then

$$\left(\begin{array}{ll} \mathit{Stop} & \text{if } y = n_a \text{ and } j = a \\ \mathit{trans}.b!j!p_j(y.n_b) \rightarrow & \\ \quad \mathit{rec}.b?i.p_b(n_b) \rightarrow \mathit{Stop} & \text{otherwise} \end{array}\right) \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } b$$

This gives the conditions required for an application of Rule `input` to yield that

$$\mathit{USER}_b \,|[\, \mathit{trans}.b.a.p_a(n_a.\mathit{NONCE}) \,]|\, \mathit{Stop} \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } b$$

as required.

5.2.1 A weak encryption mechanism again

Once more we consider the effect of distributivity of encryption, this time in the presence of full typing of messages.

If $E3$ is also allowed, then the rank function of Figure 10 is no longer well-defined. In particular, we find that

$$\begin{aligned} \rho(p_b(n_a.a)) &= 1 && (14) \\ \rho(p_b(n_a).p_b(a)) &= 0 && (15) \end{aligned}$$

even though $p_b(n_a.a) = p_b(n_a).p_b(a)$.

An attempt to avoid this difficulty might define $\rho(p_b(n_a)) = 1$ but then there is a problem in the case where $j \neq a$:

$$\begin{aligned} \rho(p_b(n_a.j)) &= 0 \\ \rho(p_b(n_a).p_b(j)) &= 1 \end{aligned}$$

The difficulty exposed in lines 14 and 15 points to the fact that the presence of message $p_b(n_a.a)$ in the system might allow the enemy to construct a message of rank 0. In fact this turns out to be the case, and we find that the system $\mathit{NET} \,|[\, \mathit{trans}.b.a.p_a(n_a.\mathit{NONCE}) \,]|\, \mathit{Stop}$ allows the following trace:

$$\langle \mathit{trans}.a.b.p_b(n_a.a), \mathit{rec}.b.c.p_b(n_a.c), \mathit{trans}.b.c.p_c(n_a.n_b), \mathit{rec}.a.b.p_a(n_a.n_b)\rangle$$

This trace corresponds to the following attack:

$$\begin{array}{lcl} A \rightarrow I(B) & : & p_b(n_a.a) \\ I(C) \rightarrow B & : & p_b(n_a.c) \\ B \rightarrow I(C) & : & p_c(n_a.n_b) \\ I(B) \rightarrow A & : & p_a(n_a.n_b) \end{array}$$

The authentication property of line (13) no longer holds. Receipt of the response to a's nonce challenge issued to b does not authenticate that b treated the challenge as coming from a.

This again illustrates the necessity of establishing $R0$, that the rank function respects all of the equations on the message space. A verification uses an implicit assumption that there are no equations on the message space inconsistent with the rank function.

6 Lowe's fix

Lowe's observation in [Low96a] that the protocol does not have that $trans.a.b.p_b(n_b)$ authenticated by $rec.b.a.p_b(n_b)$ led to his suggestion that the name of user b be included in the second message of the protocol, described as follows:

$$\begin{aligned} A \to B &: p_b(n_a.a) \\ B \to A &: p_a(n_a.n_b.b) \\ A \to B &: p_b(n_b) \end{aligned}$$

6.1 Proof of authentication property (5)

In order to establish that a run of this protocol has $trans.a.b.p_b(n_b)$ authenticated by $rec.b.a.p_b(n_b)$ we use the CSP description

$$\begin{aligned} USER_a &= \Box_{i \in USER} \; trans.a!i!p_i(n_a.a) \to \\ &\qquad\qquad rec.a.i?p_a(n_a.x.i) \to \\ &\qquad\qquad\quad trans.a!i!p_i(x) \to Stop \end{aligned}$$

$$\begin{aligned} USER_b &= rec.b.a?p_b(y.a) \to \\ &\qquad trans.b!a!p_a(y.n_b.b) \to \\ &\qquad\quad rec.b.a.p_b(n_b) \to Stop \end{aligned}$$

The rank function described in Figure 11 is suitable for an application of Theorem 3.9: it meets properties $R0$—$R4$. In particular

$R4a$ $USER_a \, |[\, trans.a.b.p_b(n_b) \,]| \, Stop$ **sat** **maintains** ρ **on** a

$R4b$ $USER_b$ **sat** **maintains** ρ **on** b

The proofs follow the same pattern as those in section 5.1.5.

We will show in Section 6.3 that all of the authentication properties (1)–(8) follow for this protocol once property (5) is established. Since Lowe's amended protocol provides property (5) we will retain the amendment in the next section for analysis with regard to this property.

6.2 Multiple runs

All the analysis performed above has been on a system where there is but a single run of the protocol between a and b, and where a and b take the roles of initiator and responder respectively. While it is possible informally to generalise the verifications to systems with repeated runs of the protocol, it is also possible to describe in CSP the situation where agents perform multiple runs of the protocol and hence provide a formal verification. Analysis of a single run allows attention to be focussed on the two participants of the run. In the case where we have multiple runs, it is appropriate to model the two parties as able to engage in their other runs with any other parties, and restrict their behaviour only for the protocol run under analysis. This is the approach that we shall take.

$$
\begin{aligned}
\rho_0(u) &= 1\\
\rho_0(t) &= 1\\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_b\\ 1 & \text{otherwise}\end{cases}\\
\rho_0(p_i) &= 1\\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = A \text{ or } i = B\\ 1 & \text{otherwise}\end{cases}\\
\\
\rho(r) &= \rho_0(r)\\
\rho(p_j(m)) &= \begin{cases} 1 & \text{if } j = A \text{ and } m \in \mathit{MESSAGE}.n_b.b\\ \rho(m) & \text{otherwise}\end{cases}\\
\rho(s_j(m)) &= \begin{cases} 0 & \text{if } j = A \text{ and } m \in p_a(\mathit{MESSAGE}.n_b.b)\\ \rho(m) & \text{otherwise}\end{cases}\\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}
$$

Figure 11 Rank function for verification of Lowe's amended protocol

Rule `interleaves`

$$
\frac{P_1 \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i \qquad P_2 \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i}{P_1 \ ||| \ P_2 \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i}
$$

Rule `recursion`

$$
\frac{(\forall\, k.X_k \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i) \Rightarrow (\forall\, k.F_k(\vec{X}) \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i)}{\forall\, k.X_k \ \mathbf{sat}\ \mathtt{maintains}\ \rho\ \mathtt{on}\ i} \quad [\, \forall\, l.X_k \mathrel{\widehat{=}} F_k(\vec{A})\,]
$$

Figure 12 Further proof rules for " `maintains` ρ `on` i"

One issue to be addressed concerns the requirement to use a fresh nonce on every protocol run. This may be modelled in CSP by using an infinite sequence of nonces where $n_{a,k}$ and $n_{b,k}$ are used on the kth run of a and b respectively. The kth run of the protocol will be defined in terms of what occurs during that run, and when the $k+1$th run can commence.

This requires a proof rule for recursive definitions, which is found in Figure 12. Recursive definitions are of the form $X_k \mathrel{\widehat{=}} F_k(\vec{X})$, where each F_k is a function on CSP process expressions, containing instances of various X_l variables. The rule states that if it can be shown that each $F_k(\vec{X})$ **sat** `maintains` ρ `on` i from the inductive assumption that all of the X_k satisfy `maintains` ρ `on` i, then we may conclude that each X_k in the recursive definition does in fact satisfy `maintains` ρ `on` i.

6.2.1 Repeated runs

We first consider the situation where users may engage in one run at a time.

The users are defined in terms of a mutual recursion. In this case we model each user as being prepared either to initiate or to respond to a fresh run of the protocol whenever it is not already engaged in a particular run of the protocol.

The protocol will be investigated as to whether an arbitrary run provides authentication—such a run will be the kth run for some k.

Since we will use k to represent the kth run of the protocol, the mutual recursion will be indexed by l. The individual process variables X_l will be given as $USER_a(l)$ and $USER_b(l)$ for the two families of mutually recursive definitions defining users a and b respectively.

$$\begin{array}{rcl} USER_a(l) & = & \Box_{i \in USER} \; trans.a!i!p_i(n_{a,l}.a) \to \\ & & \qquad rec.a.i?p_a(n_{a,l}.x.i) \to \\ & & \qquad\qquad trans.a!i!p_i(x) \to USER_a(l+1) \\ & & \Box \; rec.a?j?p_a(y.j) \to \\ & & \qquad trans.a!j!p_j(y.n_{a,l}.a) \to \\ & & \qquad\qquad rec.a.j.p_a(n_{a,l}) \to USER_a(l+1) \end{array}$$

The description of $USER_b$ is entirely similar, but on $trans.b$ and $rec.b$ in place of $trans.a$ and $rec.a$.

$$\begin{array}{rcl} USER_b(l) & = & \Box_{i \in USER} \; trans.b!i!p_i(n_{b,l}.b) \to \\ & & \qquad rec.b.i?p_b(n_{b,l}.x.i) \to \\ & & \qquad\qquad trans.b!i!p_i(x) \to USER_b(l+1) \\ & & \Box \; rec.b?j?p_b(y.j) \to \\ & & \qquad trans.b!j!p_j(y.n_{b,l}.b) \to \\ & & \qquad\qquad rec.b.j.p_b(n_{b,l}) \to USER_b(l+1) \end{array}$$

Then $USER_a = USER_a(0)$ and $USER_b = USER_b(0)$.

As an example we will prove the equivalent of property (5): that $rec.b.a.p_b(n_{b,k})$ authenticates $trans.a.b.p_b(n_{b,k})$ for any given k. In other words, if b's kth run of the protocol is initiated by a and is between a and b, then b's receipt of the final message authenticates that a sent that message.

The description of $USER_b(k)$ will be restricted to reflect the fact that the analysis is with respect to this run. The descriptions of the other $USER_b(l)$s and all the $USER_a(l)$s will remain unchanged.

$$\begin{array}{rcl} USER_b(k) & = & rec.b.a?p_b(y.a) \to \\ & & \qquad trans.b!a!p_a(y.n_{b,k}.b) \to \\ & & \qquad\qquad rec.b.a.p_b(n_{b,k}) \to USER_b(k+1) \end{array}$$

An appropriate rank function is given in Figure 13.

This rank function meets $R0$—$R3$. We have only to prove $R4a$ and $R4b$ to establish that $rec.b.a.p_b(n_{b,k})$ authenticates $trans.a.b.p_b(n_{b,k})$ for k.

R4a

We consider $USER_a$ restricted on $trans.a.b.p_b(n_{b,k})$. In other words, we aim to prove for each l that

$$USER_a(l) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a \qquad (16)$$

$$
\begin{aligned}
\rho_0(u) &= 1 \\
\rho_0(t) &= 1 \\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_{b,k} \\ 1 & \text{otherwise} \end{cases} \\
\rho_0(p_i) &= 1 \\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = a \text{ or } i = b \\ 1 & \text{otherwise} \end{cases} \\
\\
\rho(r) &= \rho_0(r) \\
\rho(p_j(m)) &= \begin{cases} 1 & \text{if } j = a \text{ and } m \in \mathit{NONCE}.n_{b,k}.b \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(s_j(m)) &= \begin{cases} 0 & \text{if } j = a \text{ and } m \in p_a(\mathit{NONCE}.n_{b,k}.b) \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}
$$

Figure 13 Rank function for verification of the kth run

We have that

$$
\begin{aligned}
&\mathit{USER}_a(l) \,|[\, \mathit{trans}.a.b.p_b(n_{b,k}) \,]|\, \mathit{Stop} = \\
&\quad \Box_{i \in \mathit{USER}} \;\; \mathit{trans}.a!i!p_i(n_{a,l}.a) \to \qquad\qquad (17) \\
&\qquad\qquad \mathit{rec}.a.i?p_a(n_{a,l}.x.i) \to \\
&\qquad\qquad \begin{pmatrix} \mathit{Stop} & \text{if } i = b \text{ and } x = n_{b,k} \\ \mathit{trans}.a!i!p_i(x) \to & \\ \quad (\mathit{USER}_a(l+1) \,|[\, \mathit{trans}.a.b.p_b(n_{b,k}) \,]|\, \mathit{Stop}) & \text{otherwise} \end{pmatrix} \\
&\quad \Box \; \mathit{rec}.a?j?p_a(y.j) \to \\
&\qquad \mathit{trans}.a!j!p_j(y.n_{a,l}.a) \to \\
&\qquad\quad \mathit{rec}.a.j.p_a(n_{a,l}) \to (\mathit{USER}_a(l+1) \,|[\, \mathit{trans}.a.b.p_b(n_{b,k}) \,]|\, \mathit{Stop}) \qquad (18)
\end{aligned}
$$

We will apply Rule `recursion`. We begin by assuming (16) and aim to prove from this assumption that the definition given in lines 17 to 18 also satisfies `maintains` ρ `on` a.

The first branch of the choice, given in line 17, is shown to satisfy `maintains` ρ `on` a by an applying Rules `input` and `output`. We obtain firstly from our assumption and Rule `output` that

$$\mathit{trans}.a!i!p_i(x) \to (\mathit{USER}_a(l+1) \,|[\, \mathit{trans}.a.b.p_b(n_{b,k}) \,]|\, \mathit{Stop}) \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a$$

provided $\rho(p_i(x)) \geqslant 1$. It follows from an application of Rule `input` that

$$
\begin{aligned}
&\mathit{rec}.a?j?p_a(n_{a,l}.x.i) \to \\
&\begin{pmatrix} \mathit{Stop} & \text{if } i = b \text{ and } x = n_{b,k} \\ \mathit{trans}.a!i!p_i(x) \to & \\ \quad (\mathit{USER}_a(l+1) \,|[\, \mathit{trans}.a.b.p_b(n_{b,k}) \,]|\, \mathit{Stop}) & \text{otherwise} \end{pmatrix} \\
&\textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a
\end{aligned}
$$

provided $\rho(p_a(n_{a,l}.x.i)) \geqslant 1 \Rightarrow ((i = b \wedge x = n_{b,k}) \vee \rho(p_i(x)) \geqslant 1)$; and this side condition is easily checked. Another application of Rule `output` with the observation that $\rho(p_i(n_{a,l}.a) \geqslant 1$ for any i yields that the first branch of the choice satisfies `maintains` ρ `on` a.

A similar application of rules `input` and `output` establishes that the second branch of the choice satisfies `maintains` ρ `on` a. The condition required of ρ is that $\rho(p_a(y.j)) \geqslant 1 \Rightarrow \rho(p_j(y.n_{a,l}.a) \geqslant 1$; and this is easily checked.

Hence by an application of Rule `choice` we find that the process definition for the expression $USER_a(l) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop$ satisfies `maintains` ρ `on` a. Hence (16) follows from an application of Rule `recursion`, and so

$$USER_a \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a$$

as required.

R4b

We must consider the cases $l = k$ and $l \neq k$ separately.

We begin by assuming that $USER_b(l)$ **sat** `maintains` ρ `on` b for each l.

Case $k = l$:

$$\begin{aligned} USER_b(k) \quad = \quad & rec.b.a?p_b(y.a) \to \\ & \quad trans.b!a!p_a(y.n_{b,k}.b) \to \\ & \quad\quad rec.b.a.p_b(n_{b,k}) \to USER_b(k+1) \end{aligned}$$

Two applications of Rule `input` (both with empty side conditions) and an application of Rule `output` with the observation that $\rho(p_a(y.n_{b,k}.b) \geqslant 1$ yield that

$$\begin{aligned} & rec.b.a?p_b(y.a) \to \\ & \quad trans.b!a!p_a(y.n_{b,k}.b) \to \qquad\qquad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } b \\ & \quad\quad rec.b.a.p_b(n_{b,k}) \to USER_b(k+1) \end{aligned}$$

under the assumption that $USER_b(k+1)$ **sat** `maintains` ρ `on` b.

Case $k \neq l$: In this case the recursive equation is

$$\begin{aligned} USER_b(l) \quad = \quad & \Box_{i \in USER} \; trans.b!i!p_i(n_{b,l}.b) \to \\ & \qquad\qquad rec.b.i?p_b(n_{b,l}.x.i) \to \\ & \qquad\qquad\quad trans.b!i!p_i(x) \to USER_b(l+1) \\ & \Box \; rec.b?j?p_b(y.j) \to \\ & \qquad trans.b!j!p_j(y.n_{b,l}.b) \to \\ & \qquad\quad rec.b.j.p_b(n_{b,l}) \to USER_b(l+1) \end{aligned}$$

Here the proof is almost identical to that provided for $USER_a(l) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop$. The requirements on ρ required for the rules to apply are:

1. $\rho(p_b(n_{b,l}.x.i)) \geqslant 1 \Rightarrow \rho(p_i(x)) \geqslant 1$;
2. $\rho(p_i(n_{b,l}.b)) \geqslant 1$;
3. $\rho(p_b(y.j)) \geqslant 1 \Rightarrow \rho(p_j(y.n_{b,l}.b)) \geqslant 1$

The first two arise from the first branch of the choice, and the third arises from the second branch. They are all easly checked, bearing in mind that $l \neq k$ means that $n_{b,l} \neq n_{b,k}$. The required result follows from an application of Rule `choice` and finally of Rule `recursion`.

Since this is true for arbitrary k we deduce that on any protocol run receipt of the nonce challenge authenticates that it was appropriately sent by the other party.

6.2.2 Concurrent runs

We finally examine the situation where an agent may be engaged in multiple protocol runs at the same time. This is modelled in the description of $USER_a$ and $USER_b$ by allowing the $l+1$th run to be initiated at any point after the start of the lth run. In particular, $USER_i(l)$ described user i at the point where the lth run of the protocol is ready to be executed. As soon as it begins, the description enables the $l+1$th run concurrently with run l (and any earlier runs which are still proceeding).

The processes $USER_a(k)$ are described as follows:

$$\begin{array}{rcl} USER_a(k) & = & \Box_{i \in USER}\ trans.a!i!p_i(n_{a,k}.a) \rightarrow \\ & & \quad USER_a(k+1) \mid\mid\mid rec.a.i?p_a(n_{a,k}.x.i) \rightarrow \\ & & \qquad trans.a!i!p_i(x) \rightarrow Stop \\ & & \Box\ rec.a?j?p_a(y.j) \rightarrow \\ & & \quad USER_a(k+1) \mid\mid\mid trans.a!j!p_j(y.n_{a,k}.a) \rightarrow \\ & & \qquad rec.a.j.p_a(n_{a,k}) \rightarrow Stop \end{array}$$

The description of $USER_b$ is entirely similar, with b replacing a throughout.

$$\begin{array}{rcl} USER_b(k) & = & \Box_{i \in USER}\ trans.b!i!p_i(n_{b,k}.b) \rightarrow \\ & & \quad USER_b(k+1) \mid\mid\mid rec.b.i?p_b(n_{b,k}.x.i) \rightarrow \\ & & \qquad trans.b!i!p_i(x) \rightarrow Stop \\ & & \Box\ rec.b?j?p_b(y.j) \rightarrow \\ & & \quad USER_b(k+1) \mid\mid\mid trans.b!j!p_j(y.n_{b,k}.b) \rightarrow \\ & & \qquad rec.b.j.p_b(n_{b,k}) \rightarrow Stop \end{array}$$

Again we define $USER_a = USER_a(0)$ and $USER_b = USER_b(0)$.

We will once more establish property 5: that $rec.b.a.p_b(n_{b,k})$ authenticates $trans.a.b.p_b(n_{b,k})$ for any given k.

The description of $USER_b(k)$ will again be restricted to reflect the situation we are analysing. The descriptions of the other $USER_b(l)$s and all the $USER_a(l)$s remain unchanged.

$$\begin{array}{rcl} USER_b(k) & = & rec.b.a?p_b(y.a) \rightarrow \\ & & \quad USER_b(k+1) \mid\mid\mid trans.b!a!p_a(y.n_{b,k}.b) \rightarrow \\ & & \qquad rec.b.a.p_b(n_{b,k}) \rightarrow Stop \end{array}$$

We use the same rank function ρ defined in Figure 13. This means we already have $R0$—$R3$. Once again we have only to prove $R4a$ and $R4b$, in order for Theorem 3.9 to yield that the authentication property holds.

R4a

We have that

$$
\begin{array}{l}
USER_a(l) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop = \\
\quad \Box_{i \in USER} \; trans.a!i!p_i(n_{a,l}.a) \rightarrow \\
\qquad\quad USER_a(l+1) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop \\
\qquad\qquad ||| \;\; rec.a?j?p_a(n_{a,l}.x.i) \rightarrow \\
\qquad\qquad\qquad \left(\begin{array}{ll} Stop & \text{if } i = b \text{ and } x = n_{b,k} \\ trans.a!i!p_i(x) \rightarrow Stop & \text{otherwise} \end{array} \right) \\
\quad \Box \; rec.a?j?p_a(y.j) \rightarrow \\
\qquad USER_a(l+1) \,|[\, trans.a.b.p_b(n_{b.k}) \,]|\, Stop \\
\qquad\quad ||| \;\; trans.a!j!p_j(y.n_{a,l}.a) \rightarrow \\
\qquad\qquad rec.a.j.p_a(n_{a,l}) \rightarrow Stop
\end{array}
$$

It is straightforward to prove that

$$
\begin{array}{l}
rec.a?j?p_a(n_{a,l}.x.i) \rightarrow \\
\left(\begin{array}{ll} Stop & \text{if } i = b \text{ and } x = n_{b,k} \\ trans.a!i!p_i(x) \rightarrow Stop & \text{otherwise} \end{array} \right) \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a
\end{array}
$$

since $\rho(p_a(n_{a,l}.x.i)) \geqslant 1 \Rightarrow ((i = b \wedge x = n_{b,k}) \vee \rho(p_i(x)) \geqslant 1)$.

Under the inductive assumption that $USER_a(l+1)$ **sat** `maintains` ρ `on` a, Rule `interleaves` yields that

$$
\begin{array}{l}
USER_a(l+1) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop \\
||| \left(\begin{array}{lll} rec.a?j?p_a(n_{a,l}.x.i) \rightarrow & Stop & \text{if } i = b \text{ and } x = n_{b,k} \\ \quad trans.a!i!p_i(x) \rightarrow Stop & & \text{otherwise} \end{array} \right) \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a
\end{array}
$$

Rules `choice` and `output` together with the fact that $\rho(p_i(n_{a,l}.a)) \geqslant 1$ for any i, yield that the first branch of the choice satisfies `maintains` ρ `on` a.

Similar reasoning yields that the second branch of the choice also satisfies `maintains` ρ `on` a, and so we deduce that the process body of the equation for $USER_a(l) \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop$ satisfies `maintains` ρ `on` a. Hence we conclude from Rule `recursion` that

$$
USER_a \,|[\, trans.a.b.p_b(n_{b,k}) \,]|\, Stop \quad \textbf{sat} \quad \texttt{maintains } \rho \texttt{ on } a
$$

as required.

R4b

We must consider the case where $l = k$ separately from the case where $l \neq k$.

We begin with the inductive hypothesis that $USER_b(l)$ **sat** `maintains` ρ `on` b for each l.

Case $k = l$:

$$
\begin{array}{lcl}
USER_b(k) & = & rec.b.a?p_b(y.a) \rightarrow \\
& & \quad USER_b(k+1) \; ||| \;\; trans.b!a!p_a(y.n_{b,k}.b) \rightarrow \\
& & \qquad\qquad\qquad\qquad rec.b.a.p_b(n_{b,k}) \rightarrow Stop
\end{array}
$$

An application of Rule **input** (with empty side condition) and an application of Rule **output** with the observation that $\rho(p_a(y.n_{b,k}.b)) \geqslant 1$ yields that

$$\begin{array}{l} \mathit{trans}.b!a!p_a(y.n_{b,k}.b) \rightarrow \\ \quad \mathit{rec}.b.a.p_b(n_{b,k}) \rightarrow \mathit{Stop} \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ b \end{array}$$

Rule **interleave** with the inductive hypothesis on $USER_b(k+1)$ yields

$$\begin{array}{lll} USER_b(k+1)\ |||\ \ \mathit{trans}.b!a!p_a(y.n_{b,k}.b) \rightarrow & \textbf{sat} & \texttt{maintains}\ \rho\ \texttt{on}\ b \\ \qquad\qquad\qquad\qquad \mathit{rec}.b.a.p_b(n_{b,k}) \rightarrow \mathit{Stop} & & \end{array}$$

and so finally

$$\begin{array}{lll} \mathit{rec}.b.a?p_b(y.a) \rightarrow & & \\ \quad USER_b(k+1)\ |||\ \ \mathit{trans}.b!a!p_a(y.n_{b,k}.b) \rightarrow & \textbf{sat} & \texttt{maintains}\ \rho\ \texttt{on}\ b \\ \qquad\qquad\qquad\qquad \mathit{rec}.b.a.p_b(n_{b,k}) \rightarrow \mathit{Stop} & & \end{array}$$

Hence by Rule **recursion** we conclude that

$$USER_b(k) \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ b \tag{19}$$

Case $k \neq l$: In this case the recursive equation is

$$\begin{array}{lll} USER_b(l) & = & \Box_{i \in USER}\ \mathit{trans}.b!i!p_i(n_{b,l}.b) \rightarrow \\ & & \qquad USER_b(l+1)\ |||\ \mathit{rec}.b.i?p_b(n_{b,l}.x.i) \rightarrow \\ & & \qquad\qquad\qquad \mathit{trans}.b!i!p_i(x) \rightarrow \mathit{Stop} \\ & & \Box\ \mathit{rec}.b?j?p_b(y.j) \rightarrow \\ & & \quad USER_b(l+1)\ |||\ \mathit{trans}.b!j!p_j(y.n_{b,l}.b) \rightarrow \\ & & \qquad\qquad \mathit{rec}.b.j.p_b(n_{b,l}) \rightarrow \mathit{Stop} \end{array}$$

Again the proof is almost identical to that provided for $USER_a(l)\ |[\ \mathit{trans}.a.b.p_b(n_{b,k})\ ]|\ \mathit{Stop}$. The requirements on ρ required for Rules **input** and **output** to be applied are as in Section 6.2.1:

1. $\rho(p_b(n_{b,l}.x.i)) \geqslant 1 \Rightarrow \rho(p_i(x)) \geqslant 1$;
2. $\rho(p_i(n_{b,l}.b)) \geqslant 1$;
3. $\rho(p_b(y.j)) \geqslant 1 \Rightarrow \rho(p_j(y.n_{b,l}.b)) \geqslant 1$

and these have already been verified in the earlier proof.

Hence we obtain for any $l \neq k$ that

$$USER_b(l) \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ b$$

which together with line 19 yields that

$$USER_b \quad \textbf{sat} \quad \texttt{maintains}\ \rho\ \texttt{on}\ b$$

as required.

6.3 Further authentication

We continue our analysis of the protocol with Lowe's fix using the description in which each user can engage in multiple concurrent runs of the protocol—this is the most general description of the protocol.

Having established for this situation that

$$NET \;\; \mathbf{sat} \;\; rec.b.a.p_b(n_{b,k}) \;\; \texttt{authenticates} \;\; trans.a.b.p_b(n_{b,k}) \tag{20}$$

we have already observed that the range of other possibilities for $r3$ authenticating $s3$ (properties (5)–(8)) follow immediately by virtue of Lemma 3.2. We also aim to establish properties (1)–(4).

We will first consider property (4). We aim to prove that the final message received by b in the kth protocol run authenticates that a received b's second message $p_a(y.n_{b,k}.b)$ where y is the nonce appearing in the first message received by b in this protocol run. In order to perform the analysis, it is therefore necessary to name the nonce received on the first step by $USER_b$. This is justified as a form of Skolemisation: giving a name to an arbitrary variable and treating it as fixed. The authentication provided is for whichever nonce is received on the first step, so this is fixed as n_r. The way this is modelled is that the kth run of $USER_b$ accepts only n_r as the initial nonce. The nonce n_r could be any of the $n_{a,l}$.

The description of the kth protocol run of $USER_b$ is then given as follows:

$$\begin{aligned} USER_b(k) \;\; = \;\; & rec.b.a.p_b(n_r.a) \to \\ & \quad USER_b(k+1) \;|||\; trans.b!a!p_a(n_r.n_{b,k}.b) \to \\ & \qquad\qquad\qquad\qquad rec.b.a.p_b(n_{b,k}) \to Stop \end{aligned}$$

The description of the $USER_b(l)$ where $l \neq k$ is the same as in Section 6.2.2.

Then the rank function given in Figure 14 meets all of $R0$—$R4$, establishing that

$$NET \;\; \mathbf{sat} \;\; rec.b.a.p_b(n_{b,k}) \;\; \texttt{authenticates} \;\; rec.a.b.p_a(n_r.n_{b,k}.b) \tag{21}$$

by Theorem 3.9. The proof has the same structure as that of Section 6.2.2. Observe that the inclusion of the name b in the message allows the source of the message to be identified: the message $rec.a.b.p_a(n_r.n_{b,k}.b)$ rather than $rec.a.USER.p_a(n_r.n_{b,k}.b)$ is authenticated.

Observe that $\rho(n_r) = 1$, allowing for the possibility that the enemy can initiate the protocol run.

Finally we turn our attention to properties (1) to (3). Transitivity of authentication (Lemma 3.4) allows the property "$rec.b.a.p_b(n_{b,k})$ $\texttt{authenticates}$ $rec.a.b.p_a(n_r.n_{b,k}.b)$" already established to be used to verify these further properties. The strategy is illustrated in Figure 15. The benefits of this approach are that we have only to prove that a message of the form $r2$ authenticates the message of the form $s1$ ($trans.a.b.p_b(n_r.a)$), and transitivity will allow us to conclude that $r3$ authenticates $s1$ from the result that $r3$ authenticates $r2$. Both $r2$ and $s1$ are messages of $USER_a$, and so the rules in Figure 2 concerning authentication within a single process will be all that are required; the more involved strategy developed around Theorem 3.9 concerning authentication between messages of different users need not be deployed.

$$
\begin{aligned}
\rho_0(u) &= 1 \\
\rho_0(t) &= 1 \\
\rho_0(n) &= \begin{cases} 0 & \text{if } n = n_{b,k} \\ 1 & \text{otherwise} \end{cases} \\
\rho_0(p_i) &= 1 \\
\rho_0(s_i) &= \begin{cases} 0 & \text{if } i = A \text{ or } i = B \\ 1 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\rho(r) &= \rho_0(r) \\
\rho(p_j(m)) &= \begin{cases} \rho(m)+1 & \text{if } j = A \text{ and } m = n_r.n_{b,k}.b \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(s_j(m)) &= \begin{cases} \rho(m)-1 & \text{if } j = A \text{ and } m = p_j(n_r.n_{b,k}.b) \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}
$$

Figure 14 Rank function for verification of Property (4)

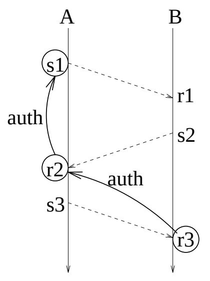

Figure 15 Authentication through transitivity

We aim to prove that "$rec.a.b.p_a(n_r.n_{b,k}.b)$ `authenticates` $trans.a.b.p_b(n_r.a)$" for each of a's protocol runs described by $USER_a(l)$. We begin by assuming this as the inductive hypothesis for each $USER_a(l)$.

$$
\begin{aligned}
USER_a(l) \;=\; & \Box_{i \in USER} \; trans.a!i!p_i(n_{a,l}.a) \rightarrow \\
& \qquad USER_a(l+1) \;|||\; rec.a.i?p_a(n_{a,l}.x.i) \rightarrow \\
& \qquad\qquad trans.a!i!p_i(x) \rightarrow Stop \\
& \Box \; rec.a?j?p_a(y.j) \rightarrow \\
& \quad USER_a(l+1) \;|||\; trans.a!j!p_j(y.n_{a,l}.a) \rightarrow \\
& \qquad\qquad rec.a.j.p_a(n_{a,l}) \rightarrow Stop
\end{aligned}
$$

Consider the first branch of the choice for a particular user i.

If $i \neq b$ or $n_{a,k} \neq n_r$ then

$$
\begin{aligned}
& rec.a?j?p_a(n_{a,k}.x.i) \rightarrow trans.a!i!p_i(x) \rightarrow Stop \\
& \quad \textbf{sat} \quad rec.a.b.p_a(n_r.n_{b,k}.b) \; \texttt{authenticates} \; trans.a.b.p_b(n_r.a)
\end{aligned}
$$

by Rule **auth.prefix.2** of Figure 2. Hence by Rule **auth.interleave** and another application of Rule **auth.prefix.2** we obtain that

$$\begin{array}{l} \mathit{trans}.a!i!p_i(n_{a,k}.a) \rightarrow \\ \quad \mathit{USER}_a(l+1) \;|||\; \mathit{rec}.a.i?p_a(n_{a,k}.x.i) \rightarrow \\ \qquad\qquad\qquad \mathit{trans}.a!i!p_i(x) \rightarrow \mathit{Stop} \\ \quad \textbf{sat} \;\; \mathit{rec}.a.b.p_a(n_r.n_{b,k}.b) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a) \end{array}$$

On the other hand, if $i = b$ and $n_{a,k} = n_r$ then

$$\begin{array}{l} \mathit{trans}.a!i!p_i(n_{a,k}.a) \rightarrow \\ \quad \mathit{USER}_a(l+1) \;|||\; \mathit{rec}.a.i?p_a(n_{a,k}.x.i) \rightarrow \\ \qquad\qquad\qquad \mathit{trans}.a!i!p_i(x) \rightarrow \mathit{Stop} \\ \quad \textbf{sat} \;\; \mathit{rec}.a.b.p_a(n_r.n_{b,k}.b) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a) \end{array}$$

follows immediately from Rule **auth.prefix.1**.

Hence by Rule **auth.choice** we obtain that

$$\begin{array}{l} \Box_{i \in \mathit{USER}} \;\; \mathit{trans}.a!i!p_i(n_{a,k}.a) \rightarrow \\ \qquad\quad \mathit{USER}_a(l+1) \;|||\; \mathit{rec}.a.i?p_a(n_{a,k}.x.i) \rightarrow \\ \qquad\qquad\qquad\qquad \mathit{trans}.a!i!p_i(x) \rightarrow \mathit{Stop} \\ \quad \textbf{sat} \;\; \mathit{rec}.a.b.p_a(n_r.n_{b,k}.b) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a) \end{array}$$

which covers the first half of the definition of $\mathit{USER}_a(l)$

For the second half, we obtain from several applications of Rule **auth.prefix.2** and Rule **interleave** that

$$\begin{array}{l} \mathit{rec}.a?j?p_a(y.j) \rightarrow \\ \quad \mathit{USER}_a(l+1) \;|||\; \mathit{trans}.a!j!p_j(y.n_{a,k}.a) \rightarrow \\ \qquad\qquad\qquad \mathit{rec}.a.j.p_a(n_{a,k}) \rightarrow \mathit{Stop} \\ \quad \textbf{sat} \;\; \mathit{rec}.a.b.p_a(n_r.n_{b,k}.b) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a) \end{array}$$

Thus we obtain that $\mathit{USER}_a(l)$ **sat** $\mathit{rec}.a.b.p_a(n_r.n_{b,k}.b)$ **authenticates** $\mathit{trans}.a.b.p_b(n_r.a)$ for arbitrary l, under the inductive hypothesis. Hence Rule **auth.recursion** yields that USER_a **sat** $\mathit{rec}.a.b.p_a(n_r.n_{b,k}.b)$ **authenticates** $\mathit{trans}.a.b.p_b(n_r.a)$.

Since $\mathit{NET} = \mathit{USER}_a \;|[\, \mathit{trans}.a, \mathit{rec}.a \,]|\; \mathit{NET}'$ where NET' is the network excepting user a, we may apply Rule **auth.parallel** to obtain that

$$\mathit{NET} \;\; \textbf{sat} \;\; \mathit{rec}.a.b.p_a(n_r.n_{b,k}.b) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a) \qquad (22)$$

Thus Lemma 3.4 applied to lines 21 and 22 yields

$$\mathit{NET} \;\; \textbf{sat} \;\; \mathit{rec}.b.a.p_b(n_{b,k}) \;\texttt{authenticates}\; \mathit{trans}.a.b.p_b(n_r.a)$$

which is property (1). As observed earlier, properties (2) and (3) are weaker than property (1) and follow immediately.

Observe that the approach that exploits transitivity could not have been used to establish property (4), since USER_a does not have $\mathit{trans}.a.b.p_b(n_{k,b})$ $(s3)$ authenticating $\mathit{rec}.a.b.p_a(n_r.n_{k,b}.b)$ $(r2)$. In other words, the reason why $\mathit{trans}.a.b.p_b(n_{k,b})$

authenticates $rec.a.b.p_a(n_r.n_{k,b}.b)$ is not contained entirely within user a (since the description $USER_a$ allows $trans.a.b.p_b(n_{k,b})$ in other situations), and consideration of user b's behaviour is required to establish it. This forces the use of Theorem 3.9. Authentication of intra-process messages can be established using the rules of Figure 2 only when the process in isolation is able to guarantee the property. As we have seen, this is the case for $r2$ authenticating $s1$, which is guaranteed by the description of $USER_a$ alone.

We have not explicitly established that $s3$ authenticates $r2$ (although it is true), because we are interested primarily in inter-process authentication; it is not an interesting property in its own right. We establish results concerning intra-process authentication only where they may be combined with inter-process authentication properties to yield the results we require.

7 Discussion

In this paper we have shown how the theory of CSP might be specialised to provide a theory for reasoning about authentication protocols. The process of verification requires the assumptions about encryption mechanisms and about the capability of a hostile agent to be made explicit. The theory includes a CSP model of the framework containing the protocol; rules for establishing authentication of messages within a single agent; and a general theorem for deducing authentication between agents from their properties with respect to a rank function ρ on messages, together with rules for deriving the required properties.

We find that construction of the rank function forces consideration of the precise reasons why a protocol is expected to work. In this respect, it should reflect the understanding of the protocol designer and make this understanding precise and explicit. Failed attempts to construct a rank function may also provide insight as to why a protocol does not provide authentication. Use of a weak encryption mechanism which allows $k(m_1.m_2) = k(m_1).k(m_2)$ would not provide authentication, as we saw in Section 5.2.1, and the difficulty in finding the rank function may lead to the discovery of the attack, as indeed happened here.

The CSP language, in common with other process algebras such as CCS, provides a language suitable for the description of protocols in a natural way. Abadi and Gordon [AbG96] observe that this kind of approach combines a precise and solid foundation for reasoning about protocols together with a clear relationship to implementations.

Another benefit of the process algebra approach is to identify precisely the authentication property required of a protocol. This may be left vague in the original formulation of the protocol, and performing the verification often gives information as to which properties actually hold, as well as pinning down precisely the properties which are provided by the protocol. The vital question as to whether the properties obtained are indeed those required are beyond the scope of the formal analysis itself, and must really be assessed according to the intended use of the protocol. Security properties may be captured as CSP specifications, or alternatively in terms of equivalences between processes, as is done in [AbG96], where a network built using the protocol is required to be equivalent to a network which describes the effect of a correct operation of the protocol: equivalence means that the protocol must operate correctly: that no context written in the process algebra can distinguish between the actual protocol and its specification.

Comparison with other approaches

The CSP approach put forward by this author in [Sch96] advocated the encapsulation of required properties in terms of the interactions between the protocol agents and their users. The intention is to separate out the required properties from the way of implementing them. Since a property of a system should be described in terms of its possible interactions with its environment, the internal events $trans.i$ and $rec.i$ should not appear in the property description. This approach requires extra events such as $a.connect\_to.b$ and $b.authenticated.a$ to appear in the protocol description as captured in $USER_a$ and $USER_b$. The descriptions might then be as follows:

$$
\begin{aligned}
USER_a &= a.connect\_to?i \rightarrow \\
&\quad trans.a!i!p_i(n_a.a) \rightarrow \\
&\quad\quad rec.a.i?p_a(n_a.x.i) \rightarrow \\
&\quad\quad\quad trans.a!i!p_i(x) \rightarrow Stop
\end{aligned}
$$

$$
\begin{aligned}
USER_b &= rec.b.a?p_b(y.a) \rightarrow \\
&\quad trans.b!a!p_a(y.n_b) \rightarrow \\
&\quad\quad rec.b?i.p_b(n_b) \rightarrow \\
&\quad\quad\quad b.authenticated.a \rightarrow Stop
\end{aligned}
$$

Our top level requirement would be that

$$NET \setminus (trans \cup rec) \quad \mathbf{sat} \quad b.authenticated.a \ \texttt{authenticates} \ a.connect\_to.b$$

The essential proof strategy will remain that proposed in this paper, but the high-level description of the property will be purely in terms of the interactions between the network and its environment. The proof rules will remain unchanged, though there will be an additional assumption required that messages of rank 0 are not introduced to protocol agents by the environment; this was guaranteed when such agents had no channels apart from $trans$ and rec.

The approach of including additional 'control' events into a protocol description is appropriate both for abstracting away the details of the protocol description, and also for providing a clearer understanding of what the protocol designer is attempting to achieve. This separation of concerns allows authentication specifications to be formulated independently of the details of any particular protocol. External events are introduced in other CSP approaches to protocol analysis [Ros95, Low96a] and elsewhere [WoL93]. In the case of the analysis performed here, additional external events such as $a.nonce\_challenge\_OK.b$ and $a.last\_message.b$ might be included to make finer distinctions between different flavours of authentication. This approach is simply an straightforward extension of the approach we have in fact taken in this paper, introducing special events are introduced to mark particular points in the run of the protocol which we have been pinpointing directly.

Another issue of interest is that Definition 3.1 is not concerned with matching up occurrences of events from T and R: once some event from R has occurred then this definition allows arbitrarily many events from T to occur without breaking authentication. Lowe [Low96b] discusses a stronger version of authentication which requires each authenticating event to correspond to a different authenticated event—he terms this requirement 'injectiveness'. Such a property is important for example in the authorisation of financial transactions. This property can be captured by strengthening the above definition as follows:

Definition 7.1

$$T \ \mathtt{injectively\ authenticates}\ R \quad = \quad \#(tr \upharpoonright R) \geqslant \#(tr \upharpoonright T)$$

□

It is immediate that T **injectively authenticates** $R \Rightarrow T$ **authenticates** R, since if $tr \upharpoonright R = \langle\rangle$ then $\#(tr \upharpoonright R) = 0$ and so $\#(tr \upharpoonright T) = 0$, and thus $tr \upharpoonright T = \langle\rangle$.

The techniques developed in this paper are concerned only with non-injective authentication. Their extension to deal with injectiveness is an area of ongoing research.

This approach contrasts with what Roscoe calls *intensional* specifications, which mean that the protocol works 'as expected' in some sense. Such properties cannot be expresses as CSP specifications independently of the protocol itself, and they really correspond to a recipe for providing the specification appropriate to the particular protocol. For example, Gollmann identifies a number of authentication properties in [Gol96]; the one closest to those we have considered here is $G4$, which states that "the origin of all messages in the protocol has to be authenticated." We would treat this as a conjunction of authentication properties: $r3$ authenticating $s1$, $r3$ authenticating $s3$ and $r2$ authenticating $s2$ (stipulating that authentication is required only when the protocol run is complete). Other examples are given in [Ros96], which gives the 'canonical' intensional specification as one where the interleavings of messages at different agents are in accordance with the messages in the protocol; and a slightly weaker on in [DOW92], which requires that whenever a participant completes its part of a protocol run then the other participant must have engaged in the sequence of events described by the protocol. These intensional properties can be formulated for any particular protocol, but not in CSP terms independently of any protocol. They correspond to recipes for producing a specification appropriate for a given protocol.

The approach of providing a rank function ρ as the core of a proof is complementary to the tools based approaches [Low95, Mil95, KMM94, Ros95] which search for attacks. The results of the latter kind of analysis provides useful information as to why a protocol is not correct, or alternatively gives a bald statement that no attack can be found. This is appropriate for debugging, but does not provide understanding as to why a protocol is correct. It is a claim of this paper that the rank function provides the basis for such an understanding, and it might be profitable to explore the interplay between the state exploration approaches and proofs. One problem concerns the relationship between the finite nature of the state space and the infinite possibilities for attacks (such as arise from such aspects as the possibility of arbitrary depths of encryption and combining of messages), and Lowe [Low96a] has begun to consider a proof strategy based on the general form of a protocol run for establishing when the absence of an attack on a finite state space really does imply that no attack is possible on the infinite state space. It seems that the rank approach might also be useful in this context.

Other approaches to direct proof of protocols, rather than the indirect route by establishing absence of attacks, tend to be based on formal languages for describing security properties, together with rules which support reasoning about statements in the language. Protocols are modelled as rules which allow the derivation of new statements from existing ones. The best known example of such a language is the BAN logic [BAN89], though the need for 'idealisation' of protocols into the logic means that the link between a protocol and its logical treatment is informal. The formal language described in [SyM96] contains lower-level primitives which relate more directly to the steps taken by a protocol, and supports reasoning concerning the knowledge of the intruder at particular stages. This language is used in conjunction with the NRL protocol analyser which is used to check reachability of negated requirements, so it is closer to the tools based approaches. However, an approach to proof reflecting that presented in this paper seems feasible.

Future directions

This paper has presented an approach to analysing and verifying authentication protocols, driven in part by consideration of the Needham-Schroeder protocol. The verification was done by hand, and the cryptographic mechanisms considered are straightforward: nonces, and public-key encryption. There is an obvious need to investigate the CSP handling of other security mechanisms such as timestamps, and to investigate more complicated protocols.

Some form of mechanical assistance for proof appears feasible, and this would bring benefits in managing the details of the proofs which can become cumbersome to track. A theory based around the rules given in the paper might be described in a theorem prover such as PVS [OSR93] which could then support or even discharge proofs for particular rank functions.

Acknowledgements

Thanks to Peter Ryan, Irfan Zakiuddin, Gavin Lowe, Cathy Meadows, and Abraham Sidiropoulos for comments on earlier forms of this work. It is also a pleasure to thank Bruno Dutertre for his work on checking the proofs with PVS and providing a number of corrections, underlining the necessity for mechanical assistance to make such proofs manageable.

Funding for this research was received from the Defence Research Agency.

References

[AbG96] M. Abadi and A.D. Gordon, *A calculus for cryptographic protocols: the spi calculus,* University of Cambridge, draft, 1996.

[BAN89] M. Burrows, M. Abadi, and R. Needham, *A logic of authentication,* Technical Report 39, SRC, DEC, 1989.

[DaS93] J. Davies and S. Schneider, *Recursion induction for real-time processes,* Formal Aspects of Computing 5(6), 1993.

[DOW92] W. Diffie, P.C. van Oorschot, and M.J. Wiener, *Authentication and key exchanges,* Designs, Codes and Cryptography 2, 1992.

[DoY83] D. Dolev and A.C. Yao, *On the security of public key protocols,* IEEE Transactions on Information Theory, 29(2), 1983.

[Gol96] D. Gollmann, *What do we mean by Entity Authentication?* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1996.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes,* Prentice-Hall, 1985.

[Kem89] R Kemmerer, *Analyzing encryption protocols using formal verification techniques,* IEEE Journal on Selected Areas in Communications, 7, 1989.

[KMM94] R Kemmerer, C. Meadows, and J. Millen, *Three systems for cryptographic protocol analysis,* Journal of Cryptology, 7(2), 1994.

[Low95] G. Lowe, *An attack on the Needham-Schroeder public-key authentication protocol,* Information Processing Letters, 56, 1995.

[Low96a] G. Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using FDR,* TACAS, LNCS 1055, 1996.

[Low96b] G. Lowe, *A hierarchy of authentication specifications,* in preparation, 1996.

[Mea92] C. Meadows, *Applying formal methods to the analysis of a key management protocol,* Journal of Computer Security, 1(1), 1992.

[Mil95] J. Millen *The interrogator model,* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1995.

[NeS78] R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers,* Communications of the ACM, 21(12), 1978.

[OSR93] S. Owre, N. Shankar, and J. Rushby, *The PVS specification language,* Computer Science Lab, SRI International, 1993.

[Ros95] A.W. Roscoe, *Modelling and verifying key-exchange protocols using CSP and FDR,* Proceedings of CSFW8, IEEE press 1995.

[Ros96] A.W. Roscoe, *Intensional specifications of security protocols,* Proceedings of CSFW9, IEEE press 1996.

[Sch96] S.A. Schneider, *Security Properties and CSP,* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1996.

[SyM96] P. Syverson and C. Meadows, *A formal language for cryptographic protocol requirements,* Designs, Codes and Cryptography 7, 1996.

[WoL93] T. Woo and S. Lam, *A semantic model for authentication protocols,* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1993.

A Proofs

A.1 Theorem 3.5

Proof of Theorem 3.5
ENEMY **sat** $(\mathit{INIT} \cup (tr \Downarrow trans)) \vdash tr \Downarrow rec$

Proof We prove easily by a mutual recursion induction that

$$\mathit{ENEMY}(S) \textbf{ sat } S \cup (tr \Downarrow trans) \vdash tr \Downarrow rec$$

The result follows from the fact that $\mathit{ENEMY} \mathrel{\widehat{=}} \mathit{ENEMY}(\mathit{INIT})$. □

A.2 Theorem 3.9

Proof of Theorem 3.9
If

R1: $\rho(INIT) \geqslant 1$

R2: $\rho(S) \geqslant 1 \wedge S \vdash m \Rightarrow \rho(m) \geqslant 1$

R3: $\rho(T) \leqslant 0$

R4: $\forall\, i.(USER_i \,|[\, S \,]|\, Stop$ **sat** `maintains` ρ `on` $i)$

then $NET \,|[\, S \,]|\, Stop$ **sat** $tr \upharpoonright T = \langle\rangle$

Proof Assume for a contradiction that $R1$—$R4$ hold, and also that $\neg(NET \,|[\, S \,]|\, Stop$ **sat** $tr \upharpoonright T = \langle\rangle)$. Then there is some trace $tr \in traces(NET \,|[\, S \,]|\, Stop)$ for which $tr \upharpoonright T \neq \langle\rangle$. Since $R3$ tells us that $\rho(t) = 0$ for any $t \in T$ we have that there are some messages in tr or rank 0 or less. Let tr_0 be the prefix of tr whose last message is the first message of tr of rank 0 or less. In otehr words, tr_0 is the trace up to the point where the first message of rank 0 occurs. By prefix closure of traces in processes, tr_0 is a trace of $NET \,|[\, S \,]|\, Stop$.

Now consider the last message of tr_0. It is either of the form $rec.i.x$ or $trans.i.x$ for some i and x, where $\rho(x) \leqslant 0$

Case $rec.i.x$. We have that tr_0 is a trace of $ENEMY$. Hence by Theorem 3.5 we have that $(INIT \cup (tr_0 \Downarrow trans)) \vdash tr_0 \Downarrow rec$ and so $(INIT \cup (tr_0 \Downarrow trans)) \vdash tr_0 \Downarrow x$. But by the definition of tr_0 we have $\rho(tr_0 \Downarrow trans) \geqslant 1$ since all messages in tr_0 apart from the last are of rank 1 or greater, so $R1$ and $R2$ yield that $\rho(x) \geqslant 1$, forcing a contradiction.

Case $trans.i.x$. Let $tr_i = tr_0 \upharpoonright \{trans.i, rec.i\}$. This is the subsequence of tr_0 in which $USER_i \,|[\, S \,]|\, Stop$ participates, so tr_i is a trace of $USER_i \,|[\, S \,]|\, Stop$. Hence by $R4$ we have `maintains` ρ `on` $i(tr_i)$. Expanding the definition we find $\rho(tr_i \upharpoonright trans.i) \geqslant \rho((tr_i \upharpoonright rec.i) \cup INIT)$, from which it follows that $\rho(x) \geqslant \rho((tr_i \upharpoonright rec.i) \cup INIT)$. However, by the definition of tr_0 and hence tr_i we have that $\rho(tr_i \upharpoonright rec.i \geqslant 1$, and from $R1$ we have that $\rho(INIT) \geqslant 1$, so we find that $\rho(x) \geqslant 1$, giving a contradiction.

In either case we find a contradiction, which establishes the theorem. $\square$