Towards automatic verification of authentication protocols on an unbounded network

James Heather and Steve Schneider
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK

Abstract

Schneider's work on rank functions [14] provides a formal approach to verification of certain properties of a security protocol. However, he illustrates the approach only with a protocol running on a small network; and no help is given with the somewhat hit-and-miss process of finding the rank function which underpins the central theorem.

In this paper, we develop the theory to allow for an arbitrarily large network, and give a clearly defined decision procedure by which one may either construct a rank function, proving correctness of the protocol, or show that no rank function exists.

We discuss the implications of the absence of a rank function, and the open question of completeness of the rank function theorem.

1 Introduction

Security protocols can be insecure even under the assumption of perfect cryptographic mechanisms, because of the possibility of unexpected interactions between the agents involved and potentially hostile intruders. Formal approaches to verification of such protocols have focused either on attempting to find attacks, or else on direct proofs that attacks cannot occur. Attack-oriented approaches model protocols and intruder capabilities in terms of rules which transform messages, and analyse a system for possible attacks through a combination of algebraic reductions on messages and model-checking for reachability (see [5, 8, 9, 6] among others.) To keep the state space manageable, these approaches generally require analysis of a restricted model of the system, together with some justification for generalising the results to larger communication networks.

The complementary approach attempts to verify protocols directly, rather than in terms of the absence of attacks. BAN logics [1] provide one such approach, in which protocols are 'idealised' into statements within the logic. Alternatively, protocols might be modelled within a formal framework and then properties proven directly within that framework [11, 18, 16, 14]. Theorem proving approaches tend to be more time-consuming in establishing correctness, though there are techniques for automating aspects of the analysis, based on the formal models.

The approach taken in [14] is to use the process algebra Communicating Sequential Processes (CSP) to model protocols in a hostile environment, and to express security properties as specifications on CSP processes. Verification proceeds by the discovery of a *rank function*, a function that assigns a value to all possible messages within the system, which is essentially used as an invariant on the messages that can circulate. However, in practice the construction of a rank function with all the required properties is intricate and difficult to do by hand. This paper presents a decision procedure which either permits the automatic construction of a rank function, or demonstrates that no rank function exists.

The structure of the paper is as follows: Section 2.1 introduces the existing approach to using rank functions with CSP for protocol analysis; Section 3 presents the main results of the paper, which are concerned with handling the unbounded nature of the network by identifying a finite number of equivalence classes, and then with providing a procedure for constructing a rank function if one exists. Section 4 discusses the issue of completeness of the rank function approach, a question which remains open in the general case of a large network. The paper finishes with some concluding remarks.

2 Protocol analysis in CSP

In this section, we give an introduction to rank functions, and a brief overview of how the theory presented in [14] may be used to verify an authentication protocol.

0-7695-0671-2/00 $10.00

CSP notation will be explained as it is introduced. For a more detailed introduction to CSP, the reader is advised to consult [4, 12, 15].

2.1 The network

The network considered in [14] consists of two honest agents *A* and *B* and one dishonest enemy. The behaviours of agents *A* and *B* are described as CSP processes $USER_A$ and $USER_B$ respectively. These user processes will vary according to the protocol under consideration; they will consist of communication along channels *trans*, representing transmission of a message, and *rec*, representing reception. An example is given in Section 2.5.

We will write $\mathcal{U}$ for the set of user identities on the network, and $\mathcal{N}$ for the set of nonces that can be used in protocol runs.

The enemy is described by a CSP process which effectively operates as a communications centre for the entire network, in the style of the Dolev-Yao model [2]. All users communicate via the enemy, who may

- pass messages on normally (but take note of the contents of the message in the process);
- intercept messages and fail to deliver them;
- construct and deliver spurious messages purporting to come from anyone he pleases.

In this last case, he may send any message which he has already seen in the network or which he can produce using only messages which he has seen. For instance, if he has observed N_A and N_B as separate messages, then he may construct $N_A.N_A$ from them and deliver this concatenation. (This concatenation operator is defined to be associative.) We define a "generates" relation $\vdash$, writing $S \vdash m$ to denote that the enemy may construct message *m* if he possesses every message in the set *S*. If *m* and *n* are messages, *k* is a key, and k^{-1} is the inverse of *k*, then $\vdash$ is the smallest relation which satisfies

$$\begin{aligned} \{m,n\} &\vdash m.n \\ \{m.n\} &\vdash m \\ \{m.n\} &\vdash n \\ \{m,k\} &\vdash \{m\}_k \\ \{\{m\}_k, k^{-1}\} &\vdash m \end{aligned}$$

and also satisfies the closure conditions

$$\begin{aligned} m \in S &\Rightarrow S \vdash m \\ S \supseteq T \wedge T \vdash m &\Rightarrow S \vdash m \\ (\forall v \in T \bullet S \vdash v) \wedge T \vdash m &\Rightarrow S \vdash m \end{aligned}$$

The enemy (already in possession of a set of messages *S*) is then described by the recursive definition:

$$\begin{aligned} ENEMY(S) = \; & trans?i?j?m \rightarrow ENEMY(S \cup \{m\}) \\ & \Box \; \Box_{i,j,S \vdash m} \; rec!i!j!m \rightarrow ENEMY(S) \end{aligned}$$

Here the enemy can either receive any message *m* transmitted by any agent *i* to any other agent *j* along a *trans* channel, and then act as the enemy with that additional message; or it can pass any message *m* that it can generate from *S* to any agent *i* along its *rec* channel, remaining with the same information *S*.

The whole network is then

$$\mathbf{NET} = (USER_a \;|||\; USER_b) \parallel \mathbf{ENEMY}$$

where $|||$ represents independent concurrent execution, and $\parallel$ represents synchronised communication. It can also have an explicit interface: $\parallel_R$ requires synchronisation of its arguments on the set *R*.

For any given protocol, there will be a (possibly infinite) set of all atoms which could ever appear in a message of the protocol. This set will encompass all the user identities, nonces and keys, and any other types of atom used in the protocol (for instance, timestamps). From this set we can construct the *message space*, usually denoted by $\mathcal{M}$, which is the space of all messages which can be generated from these atoms.

We use *INIT* to denote the set of atoms known to the enemy right from the start. Some users will be under the control of the enemy, and hence their secret keys and all nonces that they might produce will be in *INIT*; other users will be free of enemy control, and so their secret keys and nonces will not be in *INIT*.

2.2 Authentication

For an authentication protocol to be correct, we usually require that a user *B* should not finish running the protocol believing that he has been running with a user *A* unless *A* also believes that he has been running the protocol with *B*. (For a discussion of different forms of authentication, see [14].) Conditions such as this can easily be expressed as trace specifications on **NET**, requiring that no event from a set *T* has occurred unless another event from a set *R* has previously occurred. A trace of a process is a record of the sequence of events it performs during an execution. Then P **sat** S if all of the traces associated with *P* satisfy the predicate *S*.

Definition 2.2.1. For sets $R, T \in \mathcal{M}$, we define the trace specification *R* **precedes** *T* as

$$\begin{aligned} & P \;\textbf{sat}\; R \;\textbf{precedes}\; T \Leftrightarrow \\ & \quad \forall tr \in traces(P) \bullet (tr \upharpoonright R \neq \langle\rangle \Rightarrow tr \upharpoonright T \neq \langle\rangle) \end{aligned}$$

and note that, since all processes are prefix-closed, this guarantees that any occurrence of $t \in T$ in a trace will be preceded by an occurrence of some $r \in R$.

2.3 Rank functions

Definition 2.3.1. A rank function, as defined in [14], is a function

$$\rho : \mathcal{M} \to \mathbb{Z}$$

from this message space to the set of integers. In addition, we define

$$\begin{aligned} \mathcal{M}_{\rho^-} &= \{m \in \mathcal{M} \bullet \rho(m) \leq 0\} \\ \mathcal{M}_{\rho^+} &= \{m \in \mathcal{M} \bullet \rho(m) > 0\} \end{aligned}$$

If a rank function is understood, we will just write $\mathcal{M}_-$ or $\mathcal{M}_+$. In addition, we will lift ρ to events concerned with the communication of messages along channels in the obvious way: $\rho(c.m) = \rho(m)$.

The point of a rank function will be to partition the message space into those messages that the enemy might be able to get hold of, and those messages that will certainly remain out of his grasp. Anything with positive rank will be something that the enemy might get his hands on; anything of non-positive rank will be unavailable to him.

Our approach will be to construct our message space so that authentication will correspond to certain messages being kept secret from the enemy. We will be looking to find a rank function which correctly assigns a positive rank to everything that the enemy may acquire, but which still manages to give a non-positive rank to the messages corresponding to our notion of authentication.

In addition, we will have cause to ensure that our rank function allows for any sleight of hand which the enemy may wish to perform. This, in particular, means that we must assign positive rank to anything that the enemy:

- can construct from what he already has;
- can persuade an agent to transmit on the network by feeding him with messages already in his possession;
- has in his possession from the start.

2.4 The central theorem from [14]

For a process P to *maintain the rank* with respect to a rank function ρ, we mean that it will never transmit any message m with $\rho(m) \leq 0$ unless it has previously received a message m' with $\rho(m') \leq 0$. Essentially, this means that the process will never give out anything secret unless it has already received a secret message.

Definition 2.4.1. We say that P **sat maintains** ρ if

$$P \textbf{ sat } \mathit{rec}.\mathcal{U}.\mathcal{U}.\mathcal{M}_{\rho^-} \textbf{ precedes } \mathit{trans}.\mathcal{U}.\mathcal{U}.\mathcal{M}_{\rho^-}$$

"Theorem 3.5" from [14]. *If, for sets R and T, there is a rank function $\rho : \mathcal{M} \to \mathbb{Z}$ satisfying*

- $\forall m \in INIT \bullet \rho(m) > 0$
- $\forall S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet$ $((\forall m' \in S \bullet \rho(m') > 0) \wedge S \vdash m) \Rightarrow \rho(m) > 0$
- $\forall t \in T \bullet \rho(t) \leq 0$
- $\forall J \bullet USER_j \underset{R}{\|} \mathrm{STOP}$ **sat maintains** ρ

then ***NET*** **sat** R **precedes** T.

The proof is omitted; the interested reader is advised to consult [14].

2.5 Example

Consider the three message version of Lowe's fixed version [6] of the Needham-Schroeder Public-Key Protocol [10]:

$$\begin{array}{lll} \text{Message } 1. & A \to B & : \{A, N_A\}_{pkb} \\ \text{Message } 2. & B \to A & : \{N_A, N_B, B\}_{pka} \\ \text{Message } 3. & A \to B & : \{N_B\}_{pkb} \end{array}$$

In order to verify that the protocol correctly authenticates the initiator on a small network with two honest agents A and B, we wish to ensure B can never receive Message 3 of the protocol from A unless A has started the protocol with B. We define

$$\begin{aligned} USER_A &= \mathit{trans}.A!i!\{A, N_A\}_{pki} \\ &\quad \to \mathit{rec}.A.i?\{N_A, x, i\}_{pka} \\ &\quad \to \mathit{trans}.A!i!\{x\}_{pki} \end{aligned}$$

$$\begin{aligned} USER_B &= \mathit{rec}.B.A?\{A, y\}_{pkb} \\ &\quad \to \mathit{trans}.B.A!\{y, N_B, B\}_{pka} \\ &\quad \to \mathit{rec}.B.A.\{y\}_{pkb} \end{aligned}$$

and set

$$\begin{aligned} R &= \{\mathit{trans}.A.B.\{A, N_A\}_{pkb}\} \\ T &= \{\mathit{rec}.B.A.\{N_B\}_{pkb}\} \end{aligned}$$

A suitable rank function for this protocol on this network is given below.

$$\begin{aligned} \rho(U) &= 1 \\ \rho(N) &= \begin{cases} 1 & N \neq N_B \\ 0 & N = N_B \end{cases} \end{aligned}$$

$$
\begin{aligned}
\rho(pku) &= 1 \\
\rho(sku) &= \begin{cases} 0 & U = A \text{ or } B \\ 1 & \text{otherwise} \end{cases} \\
\rho(\{m\}_{pku}) &= \begin{cases} 1 & U = A, m \in \mathcal{N}.N_B.B \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(\{m\}_{sku}) &= \begin{cases} 0 & U = A, m \in \{\mathcal{N}.N_B.B\}_{pka} \\ \rho(m) & \text{otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\}
\end{aligned}
$$

A proof that this rank function satisfies the required conditions is given in [14].

The rank functions theorem now assures us that we have **NET sat** *R* **precedes** *T*; that is, that *B* cannot finish the protocol believing that he is communicating with *A* unless *A* starts the protocol with *B*.

3 Developments

3.1 Restricting the rank function to $\{0, 1\}$

As can be seen from the statement of the theorem, the rank function is in fact only used to partition the message space. The actual value of $\rho(m)$ for any given *m* will not be of interest—we will only care whether $\rho(m) > 0$ or $\rho(m) \leq 0$.

Because of this, we can redefine rank functions as

$$\rho : \mathcal{M} \to \{0, 1\}$$

restricting the range to just two values. This does not affect the validity of the rank function theorem. For if there is a function $\rho : \mathcal{M} \to \mathbb{Z}$ which satisfies the conditions of the theorem, then the function $\rho_0 : \mathcal{M} \to \{0, 1\}$ defined as

$$\rho_0(m) = 0 \quad \text{whenever} \quad \rho(m) \leq 0$$

$$\rho_0(m) = 1 \quad \text{whenever} \quad \rho(m) > 0$$

must also meet the requirements. And if there is no function $\rho : \mathcal{M} \to \mathbb{Z}$ which fits the conditions then clearly there can be no $\rho : \mathcal{M} \to \{0, 1\}$ which works. The theorem then becomes:

"Theorem 3.5" revised. *If there is a rank function* $\rho : \mathcal{M} \to \{0, 1\}$ *satisfying*

- $\forall m \in INIT \bullet \rho(m) = 1$
- $\forall S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet$
  $((\forall m' \in S \bullet \rho(m') = 1) \wedge S \vdash m) \Rightarrow \rho(m) = 1$
- $\forall t \in T \bullet \rho(t) = 0$
- $\forall J \bullet USER_j \underset{R}{\|} \text{STOP}$ **sat maintains** ρ

then ***NET*** **sat** *R* **precedes** *T*.

Thus existence of a rank function on $\{0, 1\}$ is a necessary and sufficient condition for existence of a rank function on $\mathbb{Z}$. We may concentrate only on binary rank functions, assured that establishing existence or otherwise with this restricted codomain will carry over to $\mathbb{Z}$.

3.2 Multiple concurrent runs

So far, the theory can only prove that a small system running the protocol is secure. Although the intruder is fully general and can perform any sequence of actions that he could ever wish to perform, the other players are too restricted. *A* may only engage in one run of the protocol, and that as initiator; *B*, similarly, is allowed only one run, and this must be as responder, with *A* as initiator. No other user ever takes part in a protocol run (though the enemy can simulate such runs associated with other user names).

It is not inconceivable that there should be attacks which rely on there being more than two honest agents present, or on one or more agents engaging in more than one run of the protocol. These runs will not necessarily follow on one after the other; they may be run concurrently. We must, therefore, refine our model to allow for an arbitrary number of users each taking part in an arbitrary number of concurrent runs of the protocol, as initiator or responder, and communicating with any other agents they may choose.

All we insist on is that the honest agents must act in accordance with the rules of the protocol. Anything which constitutes a valid attempt to run the protocol as the designer intended will be allowed; but even in our fully general model, only the intruder will be allowed complete freedom of expression.

Our formal assumptions about protocols covered by this paper are listed below.

Assumption 3.2.1. The operation of the protocol is independent of the identities of the agents involved.

Assumption 3.2.2. The sequence of messages passed in a protocol run is determined entirely by the identities of the agents involved and the choices of nonces that the agents make.

Assumption 3.2.3. The protocol is intended to involve only two agents—the initiator and the responder—and possibly a trusted server.

3.2.1 The old model

In the model presented in [14], the two agents *A* and *B* (and possibly a server **S**) run together in parallel with the enemy.

The users' (and server's) alphabets are pairwise disjoint, so they are in fact interleaved; and then this large process is joined in parallel with the enemy:

$$\mathbf{NET} = (USER_A \;|||\; USER_B \;|||\; \mathbf{S}) \parallel \mathbf{ENEMY}$$

3.2.2 The new players

Let us suppose that we have an infinite[1] set $\mathcal{U}$ of all users, and that for each user $U \in \mathcal{U}$ we have an infinite set of nonces $\mathcal{N}_U^I$ which U may use when acting as initiator (but he will choose each nonce at most once); and an infinite set of nonces $\mathcal{N}_U^R$ which he will use (again, at most once each) when playing responder. All these nonce sets are disjoint.

(Depending on which protocol we are considering, we may find that an agent does not need to choose a nonce when acting as initiator, or possibly when acting as responder. This will not affect the analysis: $\mathcal{N}^I$ or $\mathcal{N}^R$ will be used in this case as an indexing set to produce an infinite interleaving of identical components. If a protocol requires the initiator or the responder to choose more than one nonce, then the model will have to be altered; but the alterations will be trivial and will not affect the essence of the discussion that follows.)

How will a general user U act? He may act as many times as he wishes—once for each nonce in $\mathcal{N}_U^I$—as initiator, each time communicating with an user of his choice; and, concurrently, as many times as he wishes—once for each nonce in $\mathcal{N}_U^R$—as responder, each time communicating with any user who chooses to contact him. So, assuming we have a process $U_J^I(n)$ which describes a general user U acting as initiator, communicating with user J and using nonce n, and similarly for $U_J^R(n)$, we will find that

$$U = \Big(\;|||_{n \in \mathcal{N}_U^I} \; \Box_{J \in \mathcal{U}} \; U_J^I(n) \Big) \;|||\; \Big(\;|||_{n \in \mathcal{N}_U^R} \; \Box_{J \in \mathcal{U}} \; U_J^R(n) \Big)$$

As we will see, although we specifically look for a secure run in the case that A initiates a run with B, the behaviours of A and B are no different from the behaviour of the other agents: the above description covers A and B. Thus, our entire network of users will be simply

$$\mathbb{U} = |||_{U \in \mathcal{U}} \left(\Big(\;|||_{n \in \mathcal{N}_U^I} \; \Box_{J \in \mathcal{U}} \; U_J^I(n) \Big) \;|||\; \Big(\;|||_{n \in \mathcal{N}_U^R} \; \Box_{J \in \mathcal{U}} \; U_J^R(n) \Big) \right)$$

3.2.3 The new server

The server may possibly want an infinite set $\mathcal{N}_S$ of nonces to play with. Again, providing such a set cannot cause any problem: we shall in any case need an infinite set for the indexed parallel operator so as to ensure that we allow for arbitrarily many server operations. (We could, if no nonces are needed, use $\mathbb{Z}$; but $\mathcal{N}_S$ will work just as well.) We therefore define

$$\mathbb{S} = |||_{n \in \mathcal{N}_S} \; \mathbf{S}(n)$$

where $\mathbf{S}(n)$ is the old (single-run) server process, using nonce n (if appropriate).

3.2.4 The new network

Our new network is, therefore,

$$\mathbf{NET} = (\mathbb{U} \;|||\; \mathbb{S}) \parallel \mathbf{ENEMY}$$

3.2.5 Analysing the new network

But how are we to analyse this large network? How can we hope to find a rank function ρ, and then show that **NET sat maintains** $\rho(tr)$?

Let us consider the case of the following protocol (suggested by Peter Ryan):

Message 1.	$A \rightarrow B$	:	A
Message 2.	$B \rightarrow S$	:	$\{A, N_B\}_{Ksb}$
Message 3.	$S \rightarrow A$	:	$\{N_B, B\}_{Ksa}$
Message 4.	$A \rightarrow B$	:	N_B

In this protocol, each key *Ksu* is a symmetric key shared between the server and user U. Only the agent acting as responder needs to choose a nonce, and we are hoping to authenticate the initiator. (The analysis would not be more difficult if the initiator also chose a nonce, but it would be longer and somewhat repetitive.) Since all the users are identical, we can simply check for correct authentication in a particular run of the protocol involving A and B, and a particular nonce $N_B \in \mathcal{N}_B^R$. If authentication cannot be faked in this run, then (since this run is arbitrarily chosen) it cannot be faked in any run.

We are therefore required to check that **NET sat** R **precedes** T for suitable R and T. We want to know that if B completes the protocol as responder using nonce N_B then A really did attempt to initiate the protocol with B. Thus, following [14], we might set

$$R = \{trans.A.B.A\}$$

and

$$T = \{rec.B.A.N_B\}$$

so that B cannot receive the appropriate fourth message of the protocol unless A has sent out the first message. However, we take this opportunity to improve on the model somewhat. To make the coding less protocol-specific (and

[1] Since we are working in the finite traces model, all our infinite sets will be countable.

hence easier to modify for analysing other protocols), we introduce pseudo-messages $initgo_{U,J}$ at the start of the protocol and $respdone_{J,U,N_J}$ at the end. The former will be sent to indicate that user U has attempted to initiate a protocol run with J; and the latter to inform us that J has successfully completed the protocol as responder, using nonce N_J, and (as far as he is aware) with U as initiator—and it will be noted that the form of these messages will not need to change if the protocol changes. Now, as long as we ensure that our initiator process U_J^I starts with an appropriate *initgo* and that our responder process $U_J^R(n)$ finishes with a correct *respdone*, we can set

$$R = \{trans.A.B.initgo_{A,B}\}$$

and

$$T = \{trans.B.A.respdone_{B,A,N_B}\}$$

We will, of course, need to augment the message space $\mathcal{M}$ to include all these pseudo-messages. In addition, the enemy must never be allowed to generate this pseudo-message—but this is automatic, since for every message m in T we have $\rho(m) = 0$.

The appropriate method is not to rush straight in looking for a rank function. We can drastically reduce the size of the components on which we need to find a rank function by some careful CSP manipulation. We first note that if we define

$$\mathbb{I} = \Big|\Big|\Big|_{U:\mathcal{U}} \left(\Big|\Big|\Big|_{n:\mathcal{N}_U^I} \Box_{J:\mathcal{U}} U_J^I(n) \right)$$

and

$$\mathbb{R} = \Big|\Big|\Big|_{U:\mathcal{U}} \left(\Big|\Big|\Big|_{n:\mathcal{N}_U^R} \Box_{J:\mathcal{U}} U_J^R(n) \right)$$

then clearly

$$\mathbb{U} = \mathbb{I} \,|||\, \mathbb{R}$$

Furthermore, we can separate B^R from the others by defining

$$\mathbb{R}_0 = \Big|\Big|\Big|_{U:\mathcal{U}\setminus\{B\}} \left(\Big|\Big|\Big|_{n:\mathcal{N}_U^R} \Box_{J:\mathcal{U}} U_J^R(n) \right)$$

so that

$$\mathbb{U} = \mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \left(\Big|\Big|\Big|_{n:\mathcal{N}_B^R} \Box_{J:\mathcal{U}} B_J^R(n) \right)$$

Finally, we split off the case where B responds using nonce N_B by writing

$$\mathbb{BR}_0 = \Big|\Big|\Big|_{n:\mathcal{N}_B^R\setminus\{N_B\}} \Box_{J:\mathcal{U}} B_J^R(n)$$

so that we have

$$\mathbb{U} = \mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \mathbb{BR}_0 \,|||\, \left(\Box_{J:\mathcal{U}} B_J^R(N_B) \right)$$

Now, since we are working exclusively in the traces model, choice distributes over interleaving. We can move this final choice outside everything else, to get

$$\mathbb{U} = \Box_{J:\mathcal{U}} \left(\mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \mathbb{BR}_0 \,|||\, B_J^R(N_B) \right)$$

It is a general law of CSP that

$$\Box_{i:A} P_i \ \textbf{sat}\ W(tr) \quad \text{iff} \quad \forall\, i : A \bullet P_i \ \textbf{sat}\ W(tr)$$

—or, in other words, a choice satisfies a trace predicate if and only if each branch of the choice satisfies the predicate. So we need to show that

$$\forall\, J : \mathcal{U} \bullet \left(\mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \mathbb{BR}_0 \,|||\, B_J^R(N_B) \right) \ \textbf{sat}\ R\ \textbf{precedes}\ T$$

If $J \neq A$ then the above holds trivially. B never communicates with A using N_B, so will never engage in the event $trans.B.A.respdone.B.A.N_B$, making the predicate vacuously true. We need only check, therefore, that

$$\left(\mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \mathbb{BR}_0 \,|||\, B_A^R(N_B) \right) \ \textbf{sat}\ R\ \textbf{precedes}\ T$$

Here we use rank functions. We need to find a rank function ρ such that

$$\left(\mathbb{I} \,|||\, \mathbb{R}_0 \,|||\, \mathbb{BR}_0 \,|||\, B_A^R(N_B) \right) \ \textbf{sat maintains}\ \rho(tr)$$

in order to show that the protocol is properly secure. We start by noting that if

$$\forall\, i : A \bullet P_i \ \textbf{sat maintains}\ \rho(tr)$$

then

$$\Big|\Big|\Big|_{i:A} P_i \ \textbf{sat maintains}\ \rho(tr)$$

For if the whole interleaving does not maintain the rank, it must be sending out something of rank zero without having accepted anything of rank zero. This must be because some component P_j of the interleaving has sent out something of rank zero; and if the entire interleaving has not taken in anything of rank zero, then nor has P_j; and so P_j does not maintain the rank either. Thus, we can unpack the interleaved components and check that they individually maintain the rank. We check that

- $\mathbb{I}$

- $\mathbb{R}_0$
- $\mathbb{BR}_0$
- $B_A^R(N_B)$

all maintain the rank. But these first three are all interleavings which can be further split; so, in fact, we need only check that the following processes maintain the rank:

- $U_J^I(n)$ for arbitrary $U \in \mathcal{U}, J \in \mathcal{U}, n \in \mathcal{N}_U^I$
- $U_J^R(n)$ for arbitrary $U \in \mathcal{U} \setminus \{B\}, J \in \mathcal{U}, n \in \mathcal{N}_U^R$
- $B_J^R(n)$ for arbitrary $J \in \mathcal{U}, n \in \mathcal{N}_B^R \setminus \{N_B\}$
- $B_A^R(N_B)$

If these checks all succeed, then we will have proved that the protocol satisfies initiator authentication even when multiple concurrent runs are allowed.

It should be noted exactly what we have achieved here. We have reduced the somewhat tricky problem of verifying a protocol running on an arbitrarily large network with multiple concurrent runs to the problem of verifying the protocol on a system with only a small number of runs. For to find a rank function suitable to prove correctness on the unbounded network, we now need to find a rank function only for the network where each of the four processes listed above engages in at most one run. This is a significant reduction; but we still, so far, have an arbitrary number of users and nonces to deal with.

For consideration of Ryan's Protocol, these smaller processes are defined as follows:

$$\begin{aligned} U_J^I(n) &= trans.U.J.initgo_{U,J} \\ &\quad \rightarrow trans.U.J.U \\ &\quad \rightarrow rec.S.U.\{N, J\}_{Ksu} \\ &\quad \rightarrow trans.U.J.N \\ &\quad \rightarrow \mathrm{STOP} \end{aligned}$$

$$\begin{aligned} U_J^R(n) &= rec.U.J.J \\ &\quad \rightarrow trans.U.S.\{J, n\}_{Ksu} \\ &\quad \rightarrow rec.U.J.n \\ &\quad \rightarrow trans.U.J.respdone_{U,J,n} \\ &\quad \rightarrow \mathrm{STOP} \end{aligned}$$

$$\begin{aligned} B_J^R(n) &= rec.B.J.J \\ &\quad \rightarrow trans.B.S.\{J, n\}_{Ksb} \\ &\quad \rightarrow rec.B.J.n \\ &\quad \rightarrow trans.B.J.respdone_{B,J,n} \\ &\quad \rightarrow \mathrm{STOP} \end{aligned}$$

$$\begin{aligned} B_A^R(N_B) &= rec.A.B.A \\ &\quad \rightarrow trans.B.S.\{A, N_B\}_{Ksb} \\ &\quad \rightarrow rec.B.A.N_B \\ &\quad \rightarrow trans.B.A.respdone_{B,A,N_B} \\ &\quad \rightarrow \mathrm{STOP} \end{aligned}$$

3.3 The minimal 1-set rank function ρ_0

We have not yet tackled the issue of exactly how to find a rank function when we have decided on the message space and user processes. We need to partition the message space somehow; but since we are looking for *a* rank function rather than *the* rank function, we have some latitude in how to proceed with the search. One might be forgiven for thinking that this looks like more of an art than a science!

In addition, a fruitless trial-and-error search for a rank function can never provide convincing evidence that no rank function exists. We might form a strong suspicion that there is none to be found, but no more.

Let us define the function ρ_0 (informally at first) to be the function which gives a rank of one to everything which *must* have rank one, and zero to everything else. For we recall that to be a suitable rank function we require that

- anything generable from messages of rank one should also have rank one;
- the user (and server) processes should not transmit messages of rank zero unless they have received a message of rank zero;
- everything in the enemy's initial knowledge should have rank one;
- anything in T should have rank zero.

The first three conditions provide us with everything that must have rank one: if a message is in the enemy's initial knowledge, or is generable from other messages of rank one, or can be given out by a user or server process that has received only messages of rank one, then that message must itself have rank one. Otherwise, it may have rank zero without risk of causing the function to fail on any of these three conditions. The fourth condition then becomes the crucial one: do the first three statements force the messages in T to have rank one? If not, then ρ_0 is a rank function. If $\rho_0(t) = 1$ for some $t \in T$, however, then we can be certain that there is no rank function; for ρ_0 only gives a rank of one where absolutely necessary.

More formally: we write $S \rightarrow m$ if there is a process controlling one of the users, or the server, in the CSP description of the protocol which can transmit message m having taken inputs only from the message set S, and $S \rightsquigarrow m$

if $S \vdash m$ or $S \to m$. (If $\{m_0\} \rightsquigarrow m$ then we may write the more convenient $m_0 \rightsquigarrow m$, omitting the braces.) We further define

$$S' = S \cup \{m \mid S \rightsquigarrow m\}$$

Then we let

$$X_0 = INIT$$

$$X_{n+1} = X_n{}'$$

and write

$$X = \bigcup_{i=0}^{\infty} X_i$$

Then ρ_0 is the characteristic function of the set X.

If there exists a rank function, then ρ_0 will be a rank function. Conversely, if ρ_0 is not a rank function (and the only point at which it may fail is by assigning rank one to some of T) then no rank function exists.

3.4 Reducing the size of $\mathcal{M}$

This will not yet be practical for finding and verifying a rank function by hand, or even mechanically. For in order to enumerate the set X we would like it to be finite, and it is infinite on two counts:

- the sets of users and nonces are infinite;
- $m \rightsquigarrow m.m \rightsquigarrow m.m.m.m \rightsquigarrow \ldots$.

But if we can somehow reduce the set X_0 to a finite size, and restrict the priming operation so that the sequence $X_0, X_1, X_2, \ldots$ converges to a finite set, then we will be able to construct the limit set X in a finite number of operations, and so establish in finite time whether a rank function exists.

This is what we set out to achieve in the next two sections.

3.4.1 A convergent formulation of priming

Let us define first what we mean by the *fragments* of a message:

$$\begin{aligned} fr(a) &= \{a\} \quad (a \text{ an atom}) \\ fr(\{m\}_k) &= fr(m) \cup \{k, k^{-1}, \{m\}_k\} \\ fr(m_1 \ldots m_n) &= \{fr(m_i) \mid 1 \le i \le n\} \cup \\ &\quad \{m_1 \ldots m_i \mid 1 < i \le n\} \cup \\ &\quad \{m_i \ldots m_n \mid 1 \le i < n\} \end{aligned}$$

(For this last case, the message should be fully expanded so that n is as large as possible; that is, so that no m_i can be written in the form $m_{i_1}.m_{i_2}$.)

Let D be the set of all messages, including the *initgo* and *respdone* messages, that could ever appear in a protocol run if no agent (including the enemy) ever behaves dishonestly. In other words, D is the set of all the messages that the designer intended ever to appear in a protocol run.

Now consider the subset $\mathcal{M}^0$ of $\mathcal{M}$ which contains all fragments of all messages in D. This subset is still infinite, because we have an infinite number of atoms; but it does not have the problem of arbitrarily large concatenations and encryptions. If we could reduce the number of atoms to finite, then $\mathcal{M}^0$ would be finite.

Now we note that, since $T \subset \mathcal{M}^0$, generating $X \cap \mathcal{M}^0$ would be sufficient to enable us to check whether ρ_0 is a rank function. For we are required to check whether $T \cap X = \emptyset$, and this is now equivalent to checking whether $T \cap X \cap \mathcal{M}^0 = \emptyset$.

But how can we enumerate this set? We write

$$Z_0 = X_0$$

$$Z_{n+1} = Z_n{}' \cap \mathcal{M}^0$$

and

$$Z = \bigcup_{i=0}^{\infty} Z_i$$

and give the following result:

Theorem 3.4.1. *Assuming that* $INIT \subseteq \mathcal{M}^0$*, we have that*

$$Z = X \cap \mathcal{M}^0$$

This is non-trivial: in the one case, we perform all the primings and then take a finite subset, whereas in the other, we restrict our attention to the finite subset after each priming.

The proof, which is here omitted but can be found in [3], shows by induction that $Z_i = X_i \cap \mathcal{M}^0$ for each i.

3.4.2 Reducing the size of $\mathcal{U}$ and $\mathcal{N}$

At this point, we apply a subtle renaming to the agents and nonces.

Agents A and B we will keep as A and B. To those zero or more other users who are under effective control of the enemy—that is, whose secret keys and nonces are in $INIT$—we shall assign names C_0, C_1, ... (There will usually be at least one of these, because we will wish to allow the enemy to use his own identity on the network as an honest agent.) To the remaining zero or more users whose secret keys and nonces the enemy does not

m	$N(m)$
$\{C_3.C_5.N^I_{C_3,7}.N^I_{C_3,2}\}_{pkc_2}$	$\{C_0.C_1.N^I_{C_0,0}.N^I_{C_0,1}\}_{pkc_2}$
$C_1.D_0.N^I_{D_4,1}.C_5.N^I_{D_3,3}$	$C_0.D_0.N^I_{D_1,0}.C_1.N^I_{D_2,0}$
$N^I_{C_4,1}.N^I_{C_3,2}.N^R_{C_3,6}.N^I_{C_5,4}$	$N^I_{C_0,0}.N^I_{C_1,0}.N^R_{C_1,0}.N^I_{C_2,0}$
$\{A.N_B\}_{Ksd_2}$	$\{A.N_B\}_{Ksd_0}$
$\{C_1\}_{pkc_0}$	$\{C_0\}_{pkc_1}$
$A.B.S$	$A.B.S$

Figure 1. The normal form of messages

initially know, we give names D_0, D_1, ... We rename the nonce sets accordingly: C_3 will have nonces in $\mathcal{N}^I_{C_3}$ ($= \{N^I_{C_3,0}, N^I_{C_3,1}, \ldots\}$) and $\mathcal{N}^R_{C_3}$.

We now give some definitions and results which will enable us to reduce the size of the network much further.

Definition 3.4.2. The *normal form* of a message m, written $N(m)$, is the message obtained by permuting the C-indices (that is, the i in every C_i, $N^I_{C_i,j}$, $N^R_{C_i,j}$, pkc_i, skc_i, Ksc_i) within m so that all the C-indices appear in numerical order (starting from zero), and similarly with the D-indices; and then, for each k, permuting the indices of the nonces within $\mathcal{N}^I_{C_k}$ (that is, the j in every $N^I_{C_k,j}$) so that these also appear in numerical order, and similarly for $\mathcal{N}^R_{C_k}$, $\mathcal{N}^I_{D_k}$ and $\mathcal{N}^R_{D_k}$.

The examples given in Figure 1 should make this clear.

For a set S, $N(S) = \{N(m) \mid m \in S\}$.

Definition 3.4.3. We define the relation $\sim$ on $\mathcal{M}$ such that $m_1 \sim m_2 \Leftrightarrow N(m_1) = N(m_2)$. We write $E(m) = \{v \mid m \sim v\}$.

Remark 3.4.4. $\sim$ is an equivalence relation, and $E(m)$ is the equivalence class containing m.

Definition 3.4.5. If a set $S \subseteq \mathcal{M}$ contains only entire equivalence classes—that is, whenever $m \in S$ and $m \sim v$ then $v \in S$—then it will be said to be *normal-closed*.

Proposition 3.4.6. *If S is normal-closed then whenever $S \rightsquigarrow m$ and $m \sim v$ we have $S \rightsquigarrow v$.*

Proof. We can think of the transition from m to $N(m)$ as being the result of applying a permutation of the C-indices, and another permutation of the D-indices. This is also true of the transition from $N(v)$ to v; and since $N(m) = N(v)$, we have permutations of the two index sets taking m to v.

Since the generation rules and the operation of the protocol treat all users in exactly the same way, then we can replace all elements of S with ones in the same equivalence classes by applying the same permutations to these elements—so that if all occurrences of C_i (including $N^I_{C_i,p}$, $N^R_{C_i,q}$, pkc_i, skc_i, Ksc_i) in m become C_j in v, then we convert all C_i to C_j throughout S as well. S is normal-closed, so these new elements will all be in S too, and we will have $S \rightsquigarrow v$. □

Definition 3.4.7. The transition from v to $N(v)$ is a permutation of indices. We will write this permutation of indices as σ_v, so that $N(v) = \sigma_v(v)$.

The transition from $N(v)$ to v is the inverse of this permutation, and is written as ${\sigma_v}^{-1}$, so ${\sigma_v}^{-1}(N(v)) = v$.

For a set S and a permutation of indices σ, we write $\sigma(S)$ as a shorthand for $\{\sigma(m) \mid m \in S\}$.

Corollary 3.4.8. *If S is normal-closed then so is S'.*

Proof. If $m \in S'$ then $m \in S$ or $S \rightsquigarrow m$. But since S is normal-closed, in the former case $v \in S$ and in the latter case $S \rightsquigarrow v$. So, either way, $v \in S'$. Hence S' is normal-closed. □

Proposition 3.4.9. *For any message m and permutation of indices σ, we have $\sigma(fr(m)) = fr(\sigma(m))$*

Proof. The proof is by induction on the structure of m. There are three cases to consider:

1. For the base case, if m is an atom then $\sigma(fr(m)) = \sigma(m) = fr(\sigma(m))$.

2. If $m = \{v\}_k$ then our inductive hypothesis is that the proposition holds for v. But then

$$\begin{aligned}
\sigma(fr(m)) &= \sigma(fr(\{v\}_k)) \\
&= \sigma(fr(v) \cup \{\{v\}_k, k, k^{-1}\}) \\
&= \sigma(fr(v)) \cup \\
&\quad \{\sigma(\{v\}_k), \sigma(k), \sigma(k^{-1})\} \\
&= fr(\sigma(v)) \cup \\
&\quad \{\{\sigma(v)\}_{\sigma(k)}, \sigma(k), \sigma(k)^{-1}\} \\
&= fr(\{\sigma(v)\}_{\sigma(k)}) \\
&= fr(\sigma(\{v\}_k)) \\
&= fr(\sigma(m))
\end{aligned}$$

3. If $m = m_1 \ldots m_n$ (fully expanded so that n is as large as possible) then our inductive hypothesis is that the proposition holds for each m_i. We show that the proposition holds for m by induction on n. When $n = 1$ the result is trivial; and when it holds for all concatenations

of length less than n then

$$\begin{aligned}
\sigma(fr(m)) &= \sigma(fr(m_1 \ldots m_n)) \\
&= \sigma(\{fr(m_i) \mid 1 \leq i \leq n\} \cup \\
&\quad \{m_1 \ldots m_i \mid 1 < i \leq n\} \\
&\quad \{m_i \ldots m_n \mid 1 \leq i < n\}) \\
&= \{\sigma(fr(m_i)) \mid 1 \leq i \leq n\} \cup \\
&\quad \{\sigma(m_1 \ldots m_i) \mid 1 < i \leq n\} \\
&\quad \{\sigma(m_i \ldots m_n) \mid 1 \leq i < n\}) \\
&= \{fr(\sigma(m_i)) \mid 1 \leq i \leq n\} \cup \\
&\quad \{\sigma(m_1) \ldots \sigma(m_i) \mid 1 < i \leq n\} \\
&\quad \{\sigma(m_i) \ldots \sigma(m_n) \mid 1 \leq i < n\}) \\
&= fr(\sigma(m_1 \ldots m_n)) \\
&= fr(\sigma(m))
\end{aligned}$$

Thus, by induction on n, our result holds for all concatenations.

Since the proposition holds in all three cases above, we conclude that it is true for all $m \in \mathcal{M}$. □

Corollary 3.4.10. *If S is normal-closed then so is $fr(S)$.*

Proof. We are required to show that whenever $m \in fr(S)$ and $m \sim v$, we have $v \in fr(S)$.

If $m \in fr(S)$ then $m \in fr(z)$ for some $z \in S$. So $\sigma_m(m) \in \sigma_m(fr(z))$, and by Proposition 3.4.9, $\sigma_m(m) \in fr(\sigma_m(z))$.

If $m \sim v$, $\sigma_m(m) = N(m) = N(v) = \sigma_v(v)$ and so $\sigma_v(v) \in fr(\sigma_m(z))$.

Now, reversing the above process, $v \in {\sigma_v}^{-1}(fr(\sigma_m(z)))$. Again, by Proposition 3.4.9, $v \in fr({\sigma_v}^{-1}(\sigma_m(z)))$.

But since ${\sigma_v}^{-1}$ and σ_m are permutations of the indices, ${\sigma_v}^{-1}(\sigma_m(z))$ is in the same equivalence class as z, and by the normal-closure of S we conclude that ${\sigma_v}^{-1}(\sigma_m(z)) \in S$. Then $fr({\sigma_v}^{-1}(\sigma_m(z))) \subseteq fr(S)$, and so $v \in fr(S)$.

Therefore, $fr(S)$ is normal-closed. □

Remark 3.4.11. We note that *INIT* is normal-closed. Recall that it contains all public keys, all agent identities, and the secret keys and nonces only of those agents under enemy control.

Remark 3.4.12. So is $\mathcal{M}^0$ normal-closed. For $\mathcal{M}^0 = fr(D)$, and D is normal-closed as a consequence of Assumption 3.2.1. Corollary 3.4.10 tells us that $fr(D)$ must be normal-closed as well.

For any given protocol, $\mathcal{M}^0$ will have a finite number of equivalence classes. For Assumption 3.2.2 tells us that the messages transmitted in a protocol run are parameterised only by agent identities and nonce choices; and the normalisation process effectively reduces these to four agent identities with just one initiating nonce and one responding nonce each.

Corollary 3.4.13. *X_i, and hence Z_i, is normal-closed for all i.*

Proof. The proof is a simple induction on i. In the base case, $X_0 = INIT$, which is normal-closed. And whenever X_i is normal-closed, $X_{i+1} = {X_i}'$ is normal-closed. So, by induction, X_i is closed for all i. $Z_i = X_i \cap \mathcal{M}^0$, which is the intersection of two normal-closed sets; and so Z_i is normal-closed as well. □

This has major consequences. Since all the Z_i are normal-closed, we can represent each set by just keeping track of which equivalence classes are in the set. This gives us a finite representation of Z_i.

When we come to calculate Z_{i+1} from Z_i, we can represent rules corresponding to $S \rightsquigarrow m$ (with $S \subseteq Z_i$) by treating it as if it were $N(S) \rightsquigarrow N(m)$. Although it will not in general be strictly true that $N(S) \rightsquigarrow N(m)$ whenever $S \rightsquigarrow m$, the normal closure of Z_i will ensure that $N(S) \subseteq Z_i \Rightarrow S \subseteq Z_i \Rightarrow Z_i \rightsquigarrow m \Rightarrow Z_i \rightsquigarrow N(m)$.

4 (Non-)completeness

The question will be asked: what if there is no rank function? We now have a sure way of constructing a suitable rank function if there is one, and of proving non-existence if there is not. But can we deduce anything about the security of the system when no rank function exists?

There are, in fact, two questions involved here. We should note that the rank function depends not only on the protocol under consideration but also on the network on which the protocol is to be run. Although we have concentrated here exclusively on an arbitrarily large network with multiple concurrent runs (because, for most purposes, we will want to reject a protocol that is not secure on this network; and if a protocol is secure in this case then it will be secure on any network), the statement of the central rank function theorem in Section 2.4 does not specify a type or size of network. We need to consider

1. whether the rank function theorem is complete in its most general sense: do all secure combinations of protocols and networks have associated rank functions; or can we find a protocol and a network such that, although the protocol is secure when run on that network, there is no rank function to demonstrate this?

2. whether the rank function theorem is complete in its most useful sense: do all protocols which are secure on an arbitrarily large network with multiple concurrent runs have rank functions; or can we find a protocol such that, although it is secure on this network, there is no rank function to prove the fact?

4.1 General completeness

The analysis of Section 3.2 does not show that if a protocol is secure running on a specific small network then it will be secure on a large network. It only shows that if there is a rank function for the protocol on the small network then the protocol is secure on the large network (because the rank function will apply to the large network as well). This gives an obvious approach to attacking general completeness: if we can construct a protocol which is secure on the small network, but not on the large network, then there can be no rank function for the protocol on the small network. If there were, it would also apply to the large network, contradicting the insecurity of the protocol on the large network.

Such a protocol is given below. Each user U has two public keys pku_1 and pku_2, corresponding to secret keys sku_1 and sku_2. The protocol instructs the user to leak one of these secret keys at the end of a run as initiator. One leaked secret key is not enough to breach security, and, because the protocol cannot contain nondeterministic choices, we cannot have a second run in which the second secret key of the same user is leaked. But we can construct the protocol so that one leaked secret key from each of two users constitutes a security flaw, by including a redundant message encrypted under one public key of each user:

$$\begin{array}{lll}
\text{Message } 1. & A \rightarrow B & : A \\
\text{Message } 2. & B \rightarrow A & : \{\{N_B, A, B\}_{pka_1}\}_{pkb_1} \\
\text{Message } 3. & B \rightarrow A & : \{\{N_B, A, B\}_{pka_1}\}_{pka_2} \\
\text{Message } 4. & A \rightarrow B & : N_B, ska_1
\end{array}$$

We then have an attack on the third run, after both A and B have played initiator and consequently leaked one public key each:

$$\begin{array}{lll}
\text{Message } \alpha.1. & A \rightarrow C & : A \\
\text{Message } \alpha.2. & C \rightarrow A & : \{\{N_C, A, C\}_{pka_1}\}_{pkc_1} \\
\text{Message } \alpha.3. & C \rightarrow A & : \{\{N_C, A, C\}_{pka_1}\}_{pka_2} \\
\text{Message } \alpha.4. & A \rightarrow C & : N_C, ska_1 \\
\text{Message } \beta.1. & B \rightarrow A & : B \\
\text{Message } \beta.2. & A \rightarrow B & : \{\{N_A, B, A\}_{pkb_1}\}_{pka_1} \\
\text{Message } \beta.3. & A \rightarrow B & : \{\{N_A, B, A\}_{pkb_1}\}_{pkb_2} \\
\text{Message } \beta.4. & B \rightarrow A & : N_A, skb_1 \\
\text{Message } \gamma.1. & I(A) \rightarrow B & : A \\
\text{Message } \gamma.2. & B \rightarrow I(A) & : \{\{N_B, A, B\}_{pka_1}\}_{pkb_1} \\
\text{Message } \gamma.3. & B \rightarrow I(A) & : \{\{N_B, A, B\}_{pka_1}\}_{pka_2} \\
\text{Message } \gamma.4. & I(A) \rightarrow B & : N_B, ska_1
\end{array}$$

However, although this protocol is fatally flawed when multiple runs are allowed, it must be secure for the small system which only allows one occurrence of an honest agent starting a protocol run as the initiator. For message $\gamma.4$ ever to appear, the intruder will have to be able to get hold of nonce N_B from one of message $\gamma.2$ and message $\gamma.3$. He will need either to hold ska_1, and one of ska_2 and skb_1, or to persuade another agent to decrypt one of these messages for him.

Honest agents only ever decrypt the third message of the protocol, and will only do so if all the fields of the encryption are appropriately set—which is to say that only A will decrypt it, and then only if he thinks he is communicating with B. A message 2 cannot be passed off as a message 3, because a message 3 is encrypted with two keys from the same agent, and message 2 is not. The intruder cannot use A as an oracle unless A really is trying to communicate with B, in which case there is no deception.

So the intruder will have to have discovered the appropriate keys. He can never get his hands on ska_2, since it is never transmitted; and he can only get ska_1 and skb_1 when both A and B have run the protocol as initiator. In other words, there must be at least two honest attempts to run the protocol as initiator before any attack can succeed. The protocol is, therefore, secure on the small system.

4.2 Completeness on a large network

This question is still open, and needs further consideration. If the theorem is not complete in this sense, then it will probably be on account of the fact that the rank function concept cannot distinguish between external choice and interleaving: if a process can output *either* message m_1 *or* message m_2 then we must have $\rho(m_1) = \rho(m_2) = 1$—exactly as if the process could output *both* message m_1 *and* message m_2. But no counterexample has been constructed.

However, by following this procedure, much information can be gained from the proof that there is no rank function. A derivation of the final message results: one can see exactly why the *respdone* message must have a rank of one. It has always been a simple matter, in our experience, to turn this derivation into an attack with each $S \rightarrow m$ corresponding to a part of a protocol run. We cannot say with certainty that one will *always* be able to construct an attack from this information—because, of course, this can only be true if the theorem is complete on a large network!

5 Conclusion

Until now, it has not been feasible to automate protocol analysis on an unbounded network. Analysis has been restricted either to protocols running on small networks, or to (at best) semi-automated proof techniques.

Lowe [7], Stoller [17], and Roscoe and Broadfoot [13] have all published significant results on finding bounds on the size of network that needs to be anaylsed in order to prove security on an unbounded network.

However, Lowe's result does not apply to, amongst others, protocols that make use of temporary secrets; Stoller

disallows nested encryptions; and Roscoe and Broadfoot require that each agent be engaged in at most one protocol run at any given time. Our approach does not suffer from any of these limitations.

In this paper, we have given a process which may be used to verify an authentication protocol on an unbounded network, and shown that it may be automated. We anticipate that this discovery will provide a way to avoid tedious manual proofs of protocol correctness without compromising the generality of the proof.

An implementation of this process is in progress.

Acknowledgements

Thanks are due to Peter Ryan for comments and discussion on various sections of this work, and in particular for proposing the protocol of Section 3.2.5.

References

[1] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, SRC DIGITAL, 1989.

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

[3] J. Heather and S. A. Schneider. Towards automatic verification of authentication protocols on an unbounded network. Technical Report 00-04, Royal Holloway, University of London, 2000.

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

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

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

[7] G. Lowe. Towards a completeness result for model checking of security protocols. Technical Report 6, Leicester University, 1998.

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

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

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

[11] L. C. Paulson. Proving properties of security protocols by induction. *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

[12] A. W. Roscoe. *The theory and practice of concurrency*. Prentice-Hall International, 1998.

[13] A. W. Roscoe and P. J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 1999.

[14] S. A. Schneider. Verifying authentication protocols in CSP. *IEEE TSE*, 24(9), September 1998.

[15] S. A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley, 1999.

[16] D. X. Song. Athena: a new efficient checker for security protocol analysis. *Proceedings of 12th IEEE Computer Security Foundations Workshop*, June 1999.

[17] S. D. Stoller. A bound on attacks on authentication protocols. Indiana University, Computer Science Dept, Technical Report 526, February 2000.

[18] F. J. THAYER Fábrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? *IEEE Symposium on Security and Privacy*, May 1998.