A Decision Procedure for the Existence of a Rank Function

James Heather* Steve Schneider†

Abstract

Schneider's work on rank functions [17] 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 that 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 briefly 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 [9, 11, 12, 10] 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 [14, 20, 19, 17]. 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 [17] 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 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 briefly 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.

For further information on cutting-edge techniques in security protocol analysis, see [16].

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 [17] may be used to verify an authentication protocol.

CSP notation will be explained as it is introduced. For a more detailed introduction to CSP, the reader is advised to consult [8, 15, 18].

*Department of Computing, University of Surrey, Guildford, Surrey, GU2 5XH, UK. Email: j.heather@eim.surrey.ac.uk

†Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, UK. Email: steve@cs.rhul.ac.uk

2.1. The network

The network considered in [17] 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 shall 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 [3]. 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 that 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 that 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 pass any message *m* that he 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 that 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 [17].) 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 2.1. 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 2.2. A rank function, as defined in [17], is a function

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

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

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

If a rank function is understood, we shall just write '$\mathcal{M}_-$' or '$\mathcal{M}_+$'. In addition, we shall 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 forever 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 always 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 shall 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 shall have cause to ensure that our rank function allows for any sleight of hand that 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 [17]

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 $\rho(m) \leqslant 0$ unless it has previously received a message m' with $\rho(m') \leqslant 0$. Essentially, this means that the process will never give out anything secret unless it has already received a secret message.

Definition 2.3. We say that P **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 2.4. *If, for sets R and T, there is a rank function $\rho : \mathcal{M} \to \mathbb{Z}$ satisfying*

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

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

The proof is omitted; the interested reader is advised to consult [17].

2.5. Example

Consider the three message version of Lowe's fixed version [10] of the Needham-Schroeder Public-Key Protocol [13]:

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

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

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

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

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.

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

$$\rho(sku) = \begin{cases} 0 & U = A \text{ or } B \\ 1 & \text{otherwise} \end{cases}$$

$$\rho(\{m\}_{pku}) = \begin{cases} 1 & U = A, m \in \mathcal{N}.N_B.B \\ \rho(m) & \text{otherwise} \end{cases}$$

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

$$\rho(m_1.m_2) = \min\{\rho(m_1), \rho(m_2)\}$$

A proof that this rank function satisfies the required conditions is given in [17].

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

In this section, we present new results that build on Schneider's theory in various ways, constructing from it a vastly more powerful and practical method of analysing security protocols.

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

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

Because of this, we may 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 $\rho : \mathcal{M} \to \mathbb{Z}$ that satisfies the conditions of the theorem, then the function $\rho' : \mathcal{M} \to \{0, 1\}$ defined as

$$\rho'(m) = 0 \quad \text{whenever} \quad \rho(m) \leqslant 0$$

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

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

We also wish to include the possibility of a trusted server participating in the protocol. The process *SERVER* that will describe the actions of this server will need to maintain the rank as well as the processes controlling the honest agents; we shall need to add a condition to this effect.

We thus include the server in the definition of **NET**:

$$\mathbf{NET} = \left(\left(\Big|\Big|\Big|_{J \in \mathcal{U}} USER_J \right) \,|||\, SERVER \right) \,\|\, \mathbf{ENEMY}$$

and the theorem then becomes:

Theorem 3.1. *If, for sets R and T, there exists a rank function $\rho : \mathcal{M} \to \{0, 1\}$ satisfying*

1. $\forall m \in INIT \bullet \rho(m) = 1$
2. $\forall S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet ((\forall m' \in S \bullet \rho(m') = 1) \wedge S \vdash m) \Rightarrow \rho(m) = 1$
3. $\forall t \in T \bullet \rho(t) = 0$
4. $\forall J \in \mathcal{U} \bullet USER_J \underset{R}{\|} STOP$ **maintains** ρ
5. $SERVER \underset{R}{\|} STOP$ **maintains** ρ

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

In the event that no server is involved in a protocol, then we shall have that $SERVER = STOP$; in which case this last condition will hold trivially.

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

It is not inconceivable that there would be attacks that 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 that 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 in this thesis are listed below.

Assumption 3.2. The sequence of messages passed in a protocol run is determined entirely by the identities of the agents involved and the choices of nonces, timestamps and session keys that the agents and server make. The protocol

is essentially a template containing free variables representing the initiator's identity, the responder's identity and any nonces, timestamps and session keys.

Implicit in this is that when an agent receives a new nonce, he must be willing to accept *any* value for the nonce. He may not, for instance, check that the other party's nonce (if any) differs from his nonce (if any), or keep a history of nonces that he has seen and check that an incoming nonce is one that he has not seen before.

Assumption 3.2 is slightly stronger than it need be, but formulating it in this way makes the exposition easier to follow. We could partition nonces into two different types, initiator nonces and responder nonces, thus preventing an agent from receiving his own nonce back from the other party in place of a fresh nonce.

Assumption 3.3. The protocol is intended to involve only two agents—the initiator and the responder—and possibly a trusted server.

Again, this assumption is not strictly necessary, but has been included to clarify the method. The work in this paper could be extended to cover protocols with any fixed, finite number of participants.

Assumption 3.4. An agent will never accept a value of one type when he is expecting a value of a different type. Even when he cannot decrypt the message he is receiving, he will still refuse it if the message does not have the exact form that he is expecting it to have.

Assumption 3.4 seems particularly unreasonable in the case of messages that the receiver cannot understand. How can he be expected to check the types of the values inside the encryption if he cannot decrypt it? However, Lowe and the two authors of this paper have demonstrated in [6] that a simple method of tagging each field with its intended type cheaply prevents an intruder from taking advantage of any type manipulation. The paper proves that any proof of correctness that relies on Assumption 3.4 will still hold true when the assumption is dropped, provided that the type tagging scheme is used in the implementation.

3.2.1 The old model

In the model presented in [17], the two agents A and B run together in parallel with the enemy. The users' alphabets are pairwise disjoint, so they are in fact interleaved; and then this large process is joined in parallel with the enemy:

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

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 $\mathcal{N}_U^I$ that U may use when acting as initiator (but he will choose each nonce at most once); and an infinite set of nonces $\mathcal{N}_U^R$ that 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 any 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 $U_J^I(ni)$ that describes a general user U acting as initiator, communicating with user J and using nonce ni, and similarly for $U_J^R(nr)$, we shall find that

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

As we shall see, although we look for a secure run specifically 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

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

3.2.3 The new server

The server may possibly want an infinite set $\mathcal{N}_S$ of nonces with which to play. 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.)

Suppose that we have some CSP process $SERV(ns)$ that describes how the trusted server should act when using

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

server nonce *ns* (if appropriate). We then define

$$SERVER = \Vert\Vert\Vert_{ns \in \mathcal{N}_S} SERV(ns)$$

If the server is not required to generate a nonce, then $SERV(ns)$ will be independent of *ns*; if no server is required at all, then $SERV(ns) = STOP$ and so also $SERVER = STOP$.

3.2.4 The new network

Our new network is, therefore,

$$\mathbf{NET} = (\mathbb{U} \,|||\, SERVER) \parallel \mathbf{ENEMY}$$

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 each user or server process, suitably restricted, maintains the rank?

Let us consider the case of the following protocol, suggested by Ryan:

Message 1.	$a \to b$	:	a
Message 2.	$b \to s$	:	$\{a.nb\}_{SH(s,b)}$
Message 3.	$s \to a$	:	$\{nb.b\}_{SH(s,a)}$
Message 4.	$a \to b$	:	nb

In this protocol, each key $SH(s,u)$ 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 may 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 required to check, therefore, 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 [17], 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 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 may set

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

and

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

We shall, 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 form of the pseudo-messages given above will not provide any guarantee that runs of *A* initiating with *B* are in one-to-one correspondence with runs of *B* responding to *A*; a rank function proof would here show simply that if *B* finishes the protocol as responder, believing that he was talking to *A*, then *A* has started the protocol with *B* at least once in the past. To allow for checking of one-to-one correspondence, we would need to establish that *A* had started a protocol run during which he believed that the nonce in use was N_B. Changing the form of the *initgo* messages to the form $initgo_{U,J,N}$ and setting

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

would be the easiest way to achieve this.)

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

$$\mathbb{I} = \Vert\Vert\Vert_{U \in \mathcal{U}} \left(\Vert\Vert\Vert_{ni \in \mathcal{N}_U^I} \Box_{J \in \mathcal{U}} U_J^I(ni) \right)$$

and

$$\mathbb{R} = \Vert\Vert\Vert_{U \in \mathcal{U}} \left(\Vert\Vert\Vert_{nr \in \mathcal{N}_U^R} \Box_{J \in \mathcal{U}} U_J^R(nr) \right)$$

then clearly

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

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

$$\mathbb{R}_0 = \Vert\Vert\Vert_{U \in \mathcal{U} \setminus \{B\}} \left(\Vert\Vert\Vert_{nr \in \mathcal{N}_U^R} \Box_{J \in \mathcal{U}} U_J^R(nr) \right)$$

so that

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

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

$$\mathbb{BR}_0 = \Big|||_{\substack{nr \in \mathcal{N}_B^R \\ nr \neq N_B}} \; \Box_{J \in \mathcal{U}} \; B_J^R(nr)$$

so that we have

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

and hence

$$\mathbf{NET} = \left(\mathbb{I} \;|||\; \mathbb{R}_0 \;|||\; \mathbb{BR}_0 \;|||\; \left(\Box_{J \in \mathcal{U}} \; B_J^R(N_B) \right) \;|||\; \mathit{SERVER} \right) \;\|\; \mathbf{ENEMY}$$

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

$$\mathbf{NET} = \Box_{J \in \mathcal{U}} \left(\left(\mathbb{I} \;|||\; \mathbb{R}_0 \;|||\; \mathbb{BR}_0 \;|||\; B_J^R(N_B) \;|||\; \mathit{SERVER} \right) \;\|\; \mathbf{ENEMY} \right)$$

It is a general law of CSP that, for a trace predicate W,

$$\Box_{i \in A} \; P_i \; \mathbf{sat} \; W \quad \Leftrightarrow \quad \forall \, i \in A \bullet P_i \; \mathbf{sat} \; W$$

—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

$$\forall \, J \in \mathcal{U} \bullet \left(\left(\mathbb{I} \;|||\; \mathbb{R}_0 \;|||\; \mathbb{BR}_0 \;|||\; B_J^R(N_B) \;|||\; \mathit{SERVER} \right) \;\|\; \mathbf{ENEMY} \right) \; \mathbf{sat} \; R \; \mathbf{precedes} \; T$$

In the case that $J \neq A$, the above proposition holds trivially. Agent B will never communicate with agent A using nonce N_B, and so will never engage in the event $trans.B.A.respdone_{B,A,N_B}$, satisfying the predicate vacuously. We need check, therefore, only that

$$\left(\left(\mathbb{I} \;|||\; \mathbb{R}_0 \;|||\; \mathbb{BR}_0 \;|||\; B_A^R(N_B) \;|||\; \mathit{SERVER} \right) \;\|\; \mathbf{ENEMY} \right)$$

satisfies the R **precedes** T predicate.

Here we use rank functions. We need to find a rank function ρ with the property that

$$\left(\mathbb{I} \;|||\; \mathbb{R}_0 \;|||\; \mathbb{BR}_0 \;|||\; B_A^R(N_B) \;|||\; \mathit{SERVER} \right) \; \mathbf{maintains} \; \rho$$

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

$$\forall \, i \in A \bullet P_i \; \mathbf{maintains} \; \rho$$

then

$$\Big|||_{i \in A} \; P_i \; \mathbf{maintains} \; \rho$$

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 may unpack the interleaved components and check that they individually maintain the rank. We check that

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

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

- $U_J^I(ni)$ for arbitrary $U \in \mathcal{U}, J \in \mathcal{U}, ni \in \mathcal{N}_U^I$
- $U_J^R(nr)$ for arbitrary $U \in \mathcal{U} \setminus \{B\}, J \in \mathcal{U}, nr \in \mathcal{N}_U^R$
- $B_J^R(nr)$ for arbitrary $J \in \mathcal{U}, nr \in \mathcal{N}_B^R \setminus \{N_B\}$
- $B_A^R(N_B)$
- $\mathit{SERV}(ns)$ for arbitrary $ns \in \mathcal{N}_S$

If these checks all succeed, then we shall have proved that the protocol satisfies initiator authentication even when agents are allowed to engage in multiple concurrent runs of the protocol.

$$
\begin{aligned}
U_J^I(ni) = {} & trans.U.J.initgo_{U,J} \\
& \rightarrow trans.U.J!U \\
& \rightarrow rec.S.U?\{nr.J\}_{SH(S,U)} \\
& \rightarrow trans.U.J!nr \\
& \rightarrow STOP
\end{aligned}
$$

Figure 1. Process describing the initiator in Ryan's Protocol

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 four smaller processes may be defined as in Figures 1, 2 and 3. Finding a rank function ρ such that each of these processes maintains ρ would provide a proof that Ryan's Protocol correctly authenticates the initiator of the protocol. The interested reader should consult [5] for details of how the *RankAnalyser* program can be used to do this automatically.

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 (unless, of course, the search helps us to find an attack on the protocol). We might form a strong suspicion—but no more—that there is none to be found.

Let us define the function ρ_0 (informally at first) to be the function that gives a rank of one to everything that *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;

$$
\begin{aligned}
U_J^R(nr) = {} & rec.U.J?J \\
& \rightarrow trans.U.S!\{J.nr\}_{SH(S,U)} \\
& \rightarrow rec.U.J?nr \\
& \rightarrow trans.U.J.respdone_{U,J,nr} \\
& \rightarrow STOP
\end{aligned}
$$

$$
\begin{aligned}
B_J^R(nr) = {} & rec.B.J?J \\
& \rightarrow trans.B.S!\{J.nr\}_{SH(S,B)} \\
& \rightarrow rec.B.J?nr \\
& \rightarrow trans.B.J.respdone_{B,J,nr} \\
& \rightarrow STOP
\end{aligned}
$$

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

Figure 2. Processes describing the responder in Ryan's Protocol

$$
\begin{aligned}
SERV(ns) = {} & rec.S?J?\{U.nr\}_{SH(S,J)} \\
& \rightarrow trans.S.U\{nr.J\}_{SH(S,U)}
\end{aligned}
$$

Figure 3. Process describing the server in Ryan's Protocol

- 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 may be certain that there is no rank function; for ρ_0 gives a rank of one only where absolutely necessary.

Definition 3.5. We write '$S \to m$' (read '*S leads to m*') if there is a process controlling one of the users, or the server, in the CSP description of the protocol that can transmit message m having taken inputs only from the message set S.

Definition 3.6. We shall write '$S \rightsquigarrow m$' (read '*S gives 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.

Definition 3.7. We further define

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

Definition 3.8. We make the inductive definition

$$\begin{aligned} X_0 &= INIT \\ X_{n+1} &= X_n{}' \end{aligned}$$

and write

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

Then ρ_0 is the characteristic function of the set X.

This is an inductive definition of the set X of all messages that must have a rank of one. A set X_i represents the ith approximation to 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 one or more members of T) then no rank function exists.

3.4. Reducing the size of the message space

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 should like it to be finite, and it is infinite on two counts:

1. the sets of users and nonces are infinite;
2. there is no bound on the length of messages that can be found in X, for $m \rightsquigarrow m.m \rightsquigarrow m.m.m.m \rightsquigarrow \cdots$.

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 shall 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.

Definition 3.9. The *fragments* of a message m are those contained in the set $frags(m)$, defined recursively as:

$$\begin{aligned} frags(a) &= \{a\} && (a \text{ an atom}) \\ frags(\{m\}_k) &= frags(m) \cup \{k, k^{-1}, \{m\}_k\} \\ frags(m_1 \ldots m_n) &= \left(\bigcup_{1 \leqslant i \leqslant n} frags(m_i) \right) \cup \\ &\qquad \{m_i \ldots m_j \mid 1 \leqslant i < j \leqslant n\} \end{aligned}$$

(For this last case, the message should be fully expanded so that n is as large as possible; that is, so that no m_t can be expressed in the form $m_{t_1}.m_{t_2}$.)

We extend the definition to cover sets of messages:

$$frags(S) = \bigcup_{m \in S} frags(m)$$

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 $\mathcal{M}$ that 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 3.10. *Assuming that $INIT \subseteq \mathcal{M}^0$, we have that*

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

This is non-trivial: in the case of $X \cap \mathcal{M}^0$, we perform all the primings and then take a finite subset; whereas in the case of Z, we restrict our attention to the finite subset after each priming.

Proof. We show by induction that $Z_i = X_i \cap \mathcal{M}^0$ for every i.

The base case is simple: when $i = 0$, $Z_i = INIT = X_i \cap \mathcal{M}^0$. For the inductive case when $i = k+1$, we may assume that the theorem holds for $i \leqslant k$.

It is clear that $Z_{k+1} \subseteq X_{k+1} \cap \mathcal{M}^0$ from the monotonicity of the priming operator. We are required to show that whenever $m \in X_{k+1} \cap \mathcal{M}^0$, we have also $m \in Z_{k+1}$.

Suppose, then, that $m \in X_{k+1} \cap \mathcal{M}^0$. If $m \in X_k \cap \mathcal{M}^0$ then $m \in Z_k$ by the inductive hypothesis, and so $m \in Z_k{}' \cap \mathcal{M}^0 = Z_{k+1}$ and we are done. If not, then $m \notin X_k$ but $X_k \rightsquigarrow m$.

There are now five subcases to consider according to the manner in which $X_k \rightsquigarrow m$. In each case, we show that $m \in Z_{k+1}$.

1. $X_k \to m$. Then there is a process which outputs m having taken inputs only from X_k. But a process takes inputs only from the set D of all messages that can appear in a protocol run, and $D \subseteq \mathcal{M}^0$. Thus $X_k \cap \mathcal{M}^0 \to m$, and, by our inductive hypothesis, $Z_k \to m$. Thus $m \in Z_{k+1}$.

2. $X_k \vdash m$ by encryption; that is, $m = \{m^*\}_{key}$ with $m^* \in X_k$, $key \in X_k$. Since $m \in \mathcal{M}^0$ and $\mathcal{M}^0$ is closed under fragmentation, we have that $m^* \in \mathcal{M}^0$ and $key \in \mathcal{M}^0$. By the inductive hypothesis, they are then in $X_k \cap \mathcal{M}^0 = Z_k$, and so $Z_k \rightsquigarrow m$. Thus $m \in Z_{k+1}$.

3. $X_k \vdash m$ by concatenation; that is, $m = m_1.m_2$ with each $m_j \in X_k$. Since $m \in \mathcal{M}^0$ and $\mathcal{M}^0$ is closed under fragmentation, each $m_j \in X_k \cap \mathcal{M}^0 = Z_k$, and hence $Z_k \rightsquigarrow m$. Thus $m \in Z_{k+1}$.

4. $X_k \vdash m$ by decryption; that is, $\{\{m\}_{key}, key^{-1}\} \subseteq X_k$. Now let j be the least integer such that $\exists\, m^* \in X_j \bullet \{m\}_{key} \in frags(m^*)$. We note that $j \leqslant k$ since $j = k$ will suffice if there is nothing smaller.

   If $j = 0$ then $m^* \in INIT$, and since $INIT \subseteq \mathcal{M}^0$ we know by fragmentation closure that $\{\{m\}_{key}, key^{-1}\} \subseteq \mathcal{M}^0$. In that case, we have that $\{\{m\}_{key}, key^{-1}\} \subseteq X_k \cap \mathcal{M}^0 = Z_k$, and so $Z_k \rightsquigarrow m$. Therefore, $m \in Z_{k+1}$.

   If $j > 0$ then $X_{j-1} \rightsquigarrow m^*$ by our choice of j. We split into two subcases:

   (a) If $m^* \in \mathcal{M}^0$ then by fragmentation closure $\{m\}_{key}$ and key^{-1} are both inside $\mathcal{M}^0$ and, by the inductive hypothesis, in Z_k; and then $Z_k \rightsquigarrow m$. Therefore, $m \in Z_{k+1}$.

   (b) If $m^* \notin \mathcal{M}^0$ then m^* is not a protocol message, and so $X_{j-1} \vdash m^*$. Now this cannot be by decryption, deconcatenation or concatenation because in each case this would contradict the choice of j above; so it is an encryption rule. But then $m^* = \{m_1\}_{key_1}$ with $\{m\}_{key} \in frags(m^*)$. It cannot be the case that $\{m\}_{key} \in frags(m_1)$, since this would again contradict the choice of j; and so $m = m_1$ and $key = key_1$, and $m \in Z_{j-1} \subseteq Z_{k+1}$.

5. $X_k \vdash m$ by deconcatenation; that is, $a_1 \ldots a_p.m \in X_k$ or $m.a_1 \ldots a_p \in X_k$. These two possibilities are treated in exactly parallel ways; here we take the former, and write $x = a_1 \ldots a_p.m$. Following a line of reasoning similar to that argued in case 4, we take j as the least integer such that $\exists\, m^* \in X_j \bullet x = frags(m^*)$; and we again note that $j \leqslant k$.

   Still running parallel to case 4, we see that if $j = 0$ then $m^* \in INIT \subseteq \mathcal{M}^0$, and by fragmentation closure $x \in \mathcal{M}^0$. Thus $x \in X_k \cap \mathcal{M}^0 = Z_k$, and so $Z_k \rightsquigarrow m$, in which case $m \in Z_{k+1}$.

   If $j > 0$ then $X_{j-1} \rightsquigarrow m^*$ by our choice of j. We split, once more, into two subcases:

   (a) If $m^* \in \mathcal{M}^0$ then by fragmentation closure x is inside $\mathcal{M}^0$ and, by the inductive hypothesis, in Z_k; and then $Z_k \rightsquigarrow m$. Thus $m \in Z_{k+1}$.

   (b) If $m^* \notin \mathcal{M}^0$ then m^* is not a protocol message and so $X_{j-1} \vdash m^*$. This cannot be by decryption, deconcatenation or encryption, or else the leastness of j is contradicted; and so it must be by concatenation. So we have a set $\{x_1, x_2, \ldots, x_r\} \subseteq X_{j-1}$ being concatenated to get $x_1.x_2 \ldots x_r = x$.

   Now m itself might be a concatenation; so let us split up m as far as possible, writing $m = m_1.m_2 \ldots m_g$ (with possibly $g = 1$).

We have

$$x_1.x_2 \ldots x_r = a_1.a_2 \ldots a_p.m_1.m_2 \ldots m_g$$

and we note that there will be some t, u, v such that

$$x_t = a_u \ldots a_p.m_1 \ldots m_v$$

which is to say that there will be some x_t which 'bridges the gap' between a and m. But now since $x_t \in X_{j-1}$ we can also deconcatenate x_t in X_{j-1} to get $m_1 \ldots m_v \in X_j$. In addition, since $x_{t+1} \ldots x_r = m_{v+1} \ldots m_g$, then by fragmentation closure we have that $\{x_{t+1}, \ldots, x_r\} \subseteq \mathcal{M}^0$. So $\{m_1 \ldots m_v, x_{t+1}, \ldots, x_r\} \subseteq X_j \cap \mathcal{M}^0 = Z_j \subseteq Z_k$. And by concatenation within Z_k, we have that $m = m_1 \ldots m_v.x_{t+1} \ldots x_r \in Z_{k+1}$.

This completes the induction. Finally,

$$\begin{aligned} Z &= \bigcup_{i=0}^{\infty} Z_i \\ &= \bigcup_{i=0}^{\infty} \left(X_i \cap \mathcal{M}^0\right) \\ &= \left(\bigcup_{i=0}^{\infty} X_i\right) \cap \mathcal{M}^0 \\ &= X \cap \mathcal{M}^0 \end{aligned}$$

□

3.4.2 Reducing the number of agents and nonces

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

Agents A and B we shall 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 shall 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 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 further definitions and results that will enable us to reduce the size of the network much further.

Definition 3.11. The *normal form* of a message m, written '$N(m)$', is the message obtained by applying a permutation on $\mathbb{N}$ to the C-indices (that is, the i in every C_i, $N^I_{C_i,j}$, $N^R_{C_i,j}$, $PK(C_i)$, $SK(C_i)$, $SH(x, C_i)$) 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, applying a permutation on $\mathbb{N}$ to 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 $\mathcal{N}^R_{D_k}$.

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

Figure 4. The normal form of messages

The examples given in the table in Figure 4 should make this clear.

We extend this definition to cover sets in the natural way: the normal form of a set S is

$$N(S) = \{N(m) \mid m \in S\}$$

Definition 3.12. We define the relation $\sim$ on $\mathcal{M}$ such that $m_1 \sim m_2 \Leftrightarrow N(m_1) = N(m_2)$. We further define $E(m) = \{v \mid m \sim v\}$.

Remark 3.13. The relation $\sim$ is an equivalence relation, and $E(m)$ is the equivalence class containing m.

Definition 3.14. 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 3.15. *If S is normal-closed then whenever $S \rightsquigarrow m$ and $m \sim v$ we have $S \rightsquigarrow v$.*

Proof. We may think of the transition from m to $N(m)$ as being the result of applying a permutation on $\mathbb{N}$ to the set of C-indices, and another permutation on $\mathbb{N}$ to the set of D-indices. This is also true of the transition from $N(v)$ to v; and since $N(m) = N(v)$, we have permutations on 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, we may replace each element of S with one from the same equivalence classes by applying the same permutations to this element—so that if every occurrence of C_i (including $N^I_{C_i,p}$,

$N^R_{C_i,q}$, $PK(C_i)$, $SK(C_i)$, $SH(x,C_i)$) in m becomes C_j in v, then we convert every C_i to C_j throughout S as well. S is normal-closed, so this new element will be in S too, and we shall have $S \rightsquigarrow v$. □

Definition 3.16. The transition from v to $N(v)$ is a permutation of indices. We shall 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 3.17. *If S is normal-closed then so is S'.*

Proof. Let $m \sim v$, with $m \in S'$. We are required to show that $v \in S'$.

Either $m \in S$ or $S \rightsquigarrow m$. But since S is normal-closed, in the former case $v \in S$ and in the latter case $S \rightsquigarrow v$ by Proposition 3.15. So, either way, $v \in S'$. Hence S' is normal-closed. □

Proposition 3.18. *For any message m and permutation of indices σ, we have $\sigma(\mathit{frags}(m)) = \mathit{frags}(\sigma(m))$*

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

$$\sigma(\mathit{frags}(m)) = \sigma(\{m\}) = \{\sigma(m)\} = \mathit{frags}(\sigma(m))$$

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

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

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 as follows:

$$\begin{aligned}
&\sigma(\mathit{frags}(m)) \\
&= \sigma(\mathit{frags}(m_1 \ldots m_n)) \\
&= \sigma\Big(\Big(\bigcup_{1 \leqslant i \leqslant n} \mathit{frags}(m_i)\Big) \cup \\
&\qquad \{m_i \ldots m_j \mid 1 \leqslant i < j \leqslant n\}\Big) \\
&= \Big(\bigcup_{1 \leqslant i \leqslant n} \sigma(\mathit{frags}(m_i))\Big) \cup \\
&\qquad \{\sigma(m_i \ldots m_j) \mid 1 \leqslant i < j \leqslant n\} \\
&= \Big(\bigcup_{1 \leqslant i \leqslant n} \mathit{frags}(\sigma(m_i))\Big) \cup \\
&\qquad \{\sigma(m_i) \ldots \sigma(m_j) \mid 1 \leqslant i < j \leqslant n\} \\
&= \mathit{frags}(\sigma(m_1) \ldots \sigma(m_n)) \\
&= \mathit{frags}(\sigma(m_1 \ldots m_n)) \\
&= \mathit{frags}(\sigma(m))
\end{aligned}$$

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

Corollary 3.19. *If S is normal-closed then so is $\mathit{frags}(S)$.*

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

If $m \in \mathit{frags}(S)$ then $m \in \mathit{frags}(z)$ for some $z \in S$. So $\sigma_m(m) \in \sigma_m(\mathit{frags}(z))$, and by Proposition 3.18, $\sigma_m(m) \in \mathit{frags}(\sigma_m(z))$.

If $m \sim v$, $\sigma_m(m) = N(m) = N(v) = \sigma_v(v)$ and so $\sigma_v(v) \in \mathit{frags}(\sigma_m(z))$.

Now, reversing the above process, we find that $v \in {\sigma_v}^{-1}(\mathit{frags}(\sigma_m(z)))$. Again, by Proposition 3.18, $v \in \mathit{frags}({\sigma_v}^{-1}(\sigma_m(z)))$.

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 $\mathit{frags}({\sigma_v}^{-1}(\sigma_m(z))) \subseteq \mathit{frags}(S)$, and so $v \in \mathit{frags}(S)$.

Therefore, $\mathit{frags}(S)$ is normal-closed. □

Remark 3.20. 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 3.21. So is $\mathcal{M}^0$ normal-closed. For $\mathcal{M}^0 = \mathit{frags}(D)$, and D is normal-closed as a consequence of Assumption 3.2. Corollary 3.19 tells us that $\mathit{frags}(D)$ 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 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 (one initiator, one responder, one covering actions of all hostile agents, one for all other honest agents) with just one initiating nonce and one responding nonce each. (If Assumption 3.3 were relaxed, we would need more than four agent identities to cover all possibilities.)

Corollary 3.22. *X_i, and hence Z_i, are normal-closed for each i.*

Proof. The proof is a simple induction on i. In the base case, $X_0 = \mathit{INIT}$, which is normal-closed. And whenever X_i is normal-closed, $X_{i+1} = X_i'$ is normal-closed by Corollary 3.17. So, by induction, X_i is closed for each 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 far-reaching consequences. Since each Z_i is normal-closed, we may represent each set by just keeping track of which equivalence classes are in the set. This gives us a finite representation of Z_i: we simply store the normal forms from the equivalence classes that are included in Z_i.

When we come to calculate Z_{i+1} from Z_i, we may 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. Completeness

It is not known whether Theorem 2.4 is *complete* in its most useful sense; that is, whether every protocol that is secure on an unbounded network has an associated rank function to prove its security. It remains a possibility that there are secure protocols that have no rank function.

Cervesato, Durgin, Lincoln, Mitchell and Scedrov have developed in [2] a way of specifying security protocols based on linear logic. They show, by importing standard results from linear logic and applying them to their multiset rewriting system, that correctness is an undecidable property of the class of security protocols that they can express. Further restrictions and bounds are presented by Durgin, Lincoln, Mitchell and Scedrov in [4].

If it could be shown that security was an undecidable property of the class of protocols considered in this paper, then we would know immediately that Theorem 2.4 was incomplete; otherwise, the test for existence of a rank function presented here would be a good decision procedure for establishing security.

Their results, however, appear not to apply to the class of protocols that we have been considering. Most of their work is concerned with secrecy, rather than authentication; and, crucially, their language allows for a much larger class of protocols than ours. The computation performed by an agent in our system is restricted to decryption, encryption, equality checks and nonce generation. In [2, 4] the rules by which the agents determine which messages to send out are substantially more complicated.

Work is in progress on the completeness question.

5. Conclusion

The results in this paper give a clear decision procedure by which one may determine, for a given protocol, whether or not there is a rank function to prove its correctness:

1. Describe the *rôles* of the protocol in CSP, and find appropriate sets R and T with which to specify correctness.
2. Enumerate the equivalence classes of the set $\mathcal{M}^0$.
3. Let $Z_0 = \mathit{INIT}$, representing Z_0 by the equivalence classes that it contains.
4. Repeatedly enumerate $Z_{i+1} = Z_i' \cap \mathcal{M}^0$, representing by equivalence classes, until the fixed point is reached: $Z_{k+1} = Z_k$ for some k. Declare $Z = Z_k$.
5. Check, for each $t \in T$, whether $t \in Z$.
6. If, for every $t \in T$, we have $t \notin Z$, then declare the protocol correct, and stop.
7. If, for some $t \in T$, we have $t \in Z$, then declare that no rank function exists.

If this procedure results in the conclusion that no rank function exists, because of some $t \in T \cap Z$, then an examination of the route by which t came to be in Z will almost invariably lead to the discovery of an attack on the protocol.

It makes sense to keep track, during the calculation of Z_{i+1}, of which messages (more accurately, which equivalence classes) were used to give which others. For example, when we establish that $N_A.N_B \in Z_{i+1}$ because $N_A \in Z_i$ and $N_B \in Z_i$, one should keep a record of the fact that $N_A.N_B$ was found in Z_{i+1} because $\{N_A, N_B\} \vdash N_A.N_B$ by concatenation. Afterwards, if desired, a tree may be constructed, giving a complete derivation of how each message in Z came to have a rank of one.

Acknowledgements

Thanks are due to Peter Ryan for comments and discussion on various sections of a preliminary version of this paper [7], and in particular for proposing the protocol of

Section 3.2.5. The authors are also grateful for Joshua Guttman's helpful comments on this work.

Much of the work involved in preparing this paper was done while the first author was at Royal Holloway, University of London.

References

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

[2] I. Cervesato, N. A. Durgin, P. D. Lincoln, John C. Mitchell, and Andre Scedrov. A Meta-notation for Protocol Analysis. *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 55–69, June 1999.

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

[4] N. A. Durgin, P. D. Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols (FMSP'99)*, The 1999 Federated Logic Conference (FLoC'99), July 1999.

[5] James A. Heather. *'Oh! ... Is it really you?'—Using rank functions to verify authentication protocols*. Department of Computer Science, Royal Holloway, University of London, December 2000.

[6] James A. Heather, Gavin Lowe, and Steve A. Schneider. How to avoid type flaw attacks on security protocols. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, June 2000.

[7] James A. Heather and Steve A. Schneider. Towards automatic verification of authentication protocols on an unbounded network. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, June 2000.

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

[9] Richard A. Kemmerer, Catherine A. Meadows, and Jonathan K. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

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

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

[12] Jonathan K. Millen. The interrogator model. *IEEE Computer Society*, Symposium on Research in Security and Privacy, 1995.

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

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

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

[16] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *The Modelling And Analysis Of Security Protocols*. Addison Wesley, July 2000.

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

[18] Steve A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999.

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

[20] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, May 1998.