

# Strand Spaces and Rank Functions: More than Distant Cousins

James Heather  
Department of Computing  
University of Surrey  
Guildford  
GU2 7XH  
United Kingdom  
Email: j.heather@eim.surrey.ac.uk

## Abstract

*The strand spaces model and the rank functions model have both been used successfully to analyse and verify security protocols running on unbounded networks. At first sight, these two approaches appear rather different; however, close inspection reveals that there are strong links between strand spaces and rank functions.*

## 1. Introduction

*Strand spaces* and their associated theory constitute a highly effective model for analysing security protocols and proving that they satisfy certain properties. The model was first presented in [17], in which paper the authors give a proof of correctness of Lowe's fix of the Needham-Schroeder Public-Key Protocol [10]. They demonstrate that the techniques are sufficiently powerful to allow for attacks requiring unboundedly many concurrent runs of the protocol, thus enabling one to verify protocols running on an arbitrarily large network.

*Rank functions* can similarly be used for security protocol analysis. This approach was first suggested in [14], and improved upon in [4]. The process algebra CSP [6, 12, 15] is employed to construct the underlying model of the individual participants on a network, and of the hostile intruder; network messages are transmitted across CSP channels.

The link between the strand spaces model and the rank functions model is stronger than one might guess. This paper aims to explore the link in detail.

The structure of the paper is as follows. Section 2 introduces the rank functions model, and explains how it can be used for protocol verification.

In Section 3 we shall discuss the basic elements of the strand spaces model, presenting the related techniques that have been successfully used to analyse security protocols.

We shall show in Section 4, by means of an example taken from [16], how proofs of correctness may be constructed in the strand spaces model.

The correspondence between process traces and strands, and the similar approaches to proving results about which messages the enemy may or may not learn, make it unsurprising that the link between the rank function approach and strand spaces is quite strong. We shall conduct a detailed comparison of the two methodologies in Section 5.

## 2. Protocol analysis using rank functions

In this section, we give an introduction to rank functions, and a brief overview of how the theory presented in [14, 4] may be used to verify an authentication protocol.

Some knowledge of the traces model of CSP is assumed. For a detailed introduction to CSP, the reader is advised to consult [6, 12, 15].

For more information on cutting-edge techniques in security protocol analysis, see [13].

### 2.1. The network

The network considered in [5] consists of a (usually infinite) number of honest agents, and one dishonest enemy. The behaviour of an agent  $J$  is described as a CSP process  $USER_J$ , controlling  $J$ 's actions as initiator and responder on the network. This user process will vary according to the protocol under consideration; it will consist of communication along channels *trans*, representing transmission of a message, and *rec*, representing reception. For further clarification, the reader is directed to the example given in [5].

The process *SERVER* similarly controls the operation of a trusted server. If the protocol under consideration does not require a trusted server, then we shall simply set  $SERVER = STOP$ .

We shall write ‘ $\mathcal{U}$ ’ for the set of user identities on the network (including identities of any dishonest agents), and ‘ $\mathcal{N}$ ’ for the set of nonces that can be used in protocol runs.

The enemy is described by a CSP process that effectively operates as a communications centre for the entire network, in the style of the Dolev-Yao model [1].

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

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

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 it can generate from  $S$  to any agent  $i$  along its *rec* channel, remaining with the same information  $S$ . (We write ‘ $S \vdash m$ ’ to denote that the enemy may generate message  $m$  if he possesses every message in the set  $S$ . For a formal definition of this relation, see [5].)

The whole network is then

$$\mathbf{NET} = \left( \prod_{J \in \mathcal{U}} USER_J \right) \parallel SERVER \parallel ENEMY(INIT)$$

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 that 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  $P \text{ sat } S$  if all of the traces associated with  $P$  satisfy the predicate  $S$ .

**Definition 2.1.** For sets  $R, T \subseteq \mathcal{M}$ , we define the trace specification  $R$  **precedes**  $T$  as

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

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 [4], is a function

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

from the message space to the binary-valued set  $\{0, 1\}$ . In addition, we define

$$\begin{aligned} \mathcal{M}_{\rho^-} &= \{m \in \mathcal{M} \bullet \rho(m) = 0\} \\ \mathcal{M}_{\rho^+} &= \{m \in \mathcal{M} \bullet \rho(m) = 1\} \end{aligned}$$

If a rank function is understood, we shall just write ‘ $\mathcal{M}_-$ ’ or ‘ $\mathcal{M}_+$ ’. In addition, we shall lift  $\rho$  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 a rank of one will be something that the enemy might get his hands on; anything of zero rank will be unavailable to him.

## 2.4. The central theorem from [5]

For a process  $P$  to *maintain the rank* with respect to a rank function  $\rho$ , we mean that it will never transmit any message  $m$  with  $\rho(m) = 0$  unless it has previously received a message  $m'$  with  $\rho(m') = 0$ . 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**  $\rho$  if

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

**Theorem 2.4.** If, for sets  $R$  and  $T$ , there exists a rank function  $\rho : \mathcal{M} \rightarrow \{0, 1\}$  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.  $\forall J \in \mathcal{U} \bullet USER_J \parallel STOP \text{ maintains } \rho$

5.  $SERVER \parallel_{R} STOP$  maintains  $\rho$

then  $NET$  sat  $R$  precedes  $T$ .

The proof is omitted; the interested reader is advised to consult [14, 5]. For a demonstration of how Theorem 2.4 can be used to verify security protocols, see [5].

## 2.5. The minimal 1-set rank function $\rho_0$

An important question is that of how one should tackle the search for a rank function. The approach adopted in [5] is as follows.

We define the function  $\rho_0$  (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;
- 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  $\rho_0$  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  $\rho_0$  gives a rank of one only where absolutely necessary.

**Definition 2.5.** We write ' $S \rightarrow m$ ' (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 2.6.** We write ' $S \rightsquigarrow m$ ' (read ' $S$  gives  $m$ ') if  $S \vdash m$  or  $S \rightarrow m$ .

If  $\{m_0\} \rightsquigarrow m$  then we may write the more convenient ' $m_0 \rightsquigarrow m$ ', omitting the braces.

**Definition 2.7.** We further define

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

**Definition 2.8.** 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  $\rho_0$  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  $i$ th approximation to the set  $X$ .

If there exists a rank function, then  $\rho_0$  will be a rank function. Conversely, if  $\rho_0$  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.

## 2.6. Message fragments

This, as it stands, is not yet practical for finding and verifying a rank function by hand, or even mechanically, for it requires enumeration of the infinite set  $X$ . Part of the process given in [5] by which this is made manageable involves showing that in enumerating  $X$  one may restrict one's attention to protocol messages and their fragments, so that, for example, the rank of the message  $A.A.A.A.A$  would never be considered (unless the protocol required long concatenations of agent identities). The details are irrelevant to the purposes of this paper; but the notion of the *fragments* of a message will be needed later.

**Definition 2.9.** The *fragments* of a message  $m$  are those contained in the set  $frags(m)$ , defined recursively as:

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

(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 written in the form  $m_{t_1}.m_{t_2}$ .)

We extend the definition to cover sets of messages:

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

**Definition 2.10.** Let  $D$  be the set of all 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 we define  $\mathcal{M}^0 \subset \mathcal{M}$  to be the set of all fragments of all messages in  $D$ .

The subset  $\mathcal{M}^0$  is still infinite, because we have an infinite number of atoms; but it does not have arbitrarily large concatenations and encryptions.

## 2.7. Summary

A rank function verification of a security protocol involves the following steps.

1. Identify the set  $INIT$  of atoms that the enemy knows from the start.
2. Find the closure of the set  $INIT$  under operations that the enemy can perform; that is, under concatenation, deconcatenation, encryption using known keys, decryption using known keys, and picking up messages transmitted by honest agents. The resulting set is  $X$ , and  $\rho_0$  is the characteristic function of  $X$ .
3. Show that the crucial messages in  $T$  are not members of  $X$ .
4. Conclude from this that the protocol satisfies the specified security property.

## 3. Strand spaces

In this section, we introduce the central components of the strand spaces model presented in [17, 19, 16, 18], and give the theorems and propositions that allow us to construct proofs of correctness.

### 3.1. Basic definitions

A *strand* models the actions of an agent on the network, or an atomic action performed by the penetrator. (The *penetrator* in the strand space model corresponds to the enemy in our CSP model.) A *regular strand* corresponds to a trace of a single run of a  $USER_i$  process; it represents a run of the protocol from the point of view of one of the agents involved. A *penetrator strand* models an atomic action performed by the penetrator; for instance, concatenating two messages that he knows, or sending a message out over the network. The techniques available to the penetrator in the strand spaces model are comparable to those available to the enemy in the rank functions model.

**Definition 3.1.** We write ‘ $A$ ’ to denote the space of messages communicable across the network, corresponding to  $\mathcal{M}$  in the rank functions model. An element  $t \in A$  is called a *term*.

**Definition 3.2.** The set of cryptographic keys is denoted by ‘ $K$ ’; this set is a subset of  $A$ .

**Definition 3.3.** The atomic messages that are not keys form the set  $T$ . This set is a subset of  $A$ . The sets  $K$  and  $T$  are disjoint.

**Definition 3.4.** A *strand* is a sequence of signed terms. We write ‘ $+t$ ’ to represent transmission of a term  $t$ , with reception written as ‘ $-t$ ’. A general strand is denoted by ‘ $\langle \pm t_1, \dots, \pm t_n \rangle$ ’.

A strand representing an honest agent models the transmissions and receptions involving that agent in a single run of the protocol; it corresponds to a trace of a  $USER_i$  process.

**Definition 3.5.** The signed terms of a strand are called its *nodes*. The  $k$ th node of a strand  $s$  is denoted by ‘ $\langle s, k \rangle$ ’.

### 3.2. Strand spaces and bundles

A collection of strands may be considered as a graph, with two edge types,  $\Rightarrow$  and  $\rightarrow$ , representing respectively consecutive terms on the same strand, and communication between two strands.

**Definition 3.6.** If  $n_{i+1}$  immediately follows  $n_i$  on the same strand, then we write ‘ $n_i \Rightarrow n_{i+1}$ ’. We then use ‘ $\Rightarrow^*$ ’ to denote the transitive closure of  $\Rightarrow$ , so that  $n_1 \Rightarrow^* n_2$  whenever  $n_1 = \langle s, i \rangle$  and  $n_2 = \langle s, j \rangle$  with  $i < j$ .

**Definition 3.7.** If  $n_1 = +a$  and  $n_2 = -a$  for some term  $a \in A$  then we write ‘ $n_1 \rightarrow n_2$ ’.

**Definition 3.8.** A *strand space* is then a collection of strands considered as a graph ordered by  $\Rightarrow \cup \rightarrow$ .

A *bundle* in the strand space model corresponds to a trace of the whole network in the rank functions approach. It is a finite set of strands, ordered by  $\Rightarrow \cup \rightarrow$ , on which certain conditions are imposed to ensure that

- reception events never occur unless the corresponding transmission event has occurred;
- whenever an agent starts a protocol run, he starts from the beginning of the protocol;
- there is no backwards causation; that is, there are no loops in the graph.

**Definition 3.9.** If  $\mathcal{C} \subseteq (\rightarrow \cup \Rightarrow)$  is a finite set of edges, and  $\mathcal{N}$  is the set of nodes that appear on edges in  $\mathcal{C}$ , then  $\mathcal{C}$  will be called a *bundle* if

- whenever  $n_2 \in \mathcal{N}$  and  $n_2$  is a negative node, then there exists a unique  $n_1 \in \mathcal{N}$  with  $n_1 \rightarrow n_2$ ;
- whenever  $n_2 \in \mathcal{N}$  and  $n_1 \Rightarrow n_2$ , then  $n_1 \Rightarrow n_2 \in \mathcal{C}$ ;
- $\mathcal{C}$  is acyclic.

### 3.3. Penetrator strands

The analogue of the  $\vdash$  relation in the world of CSP is a type of strand known as a *penetrator strand*. A penetrator strand represents a deduction that the penetrator may make under  $\vdash$ , with different types of penetrator strand corresponding to different types of deduction. In addition, since the penetrator in the strand spaces model has no local state, there will be a type of penetrator strand to represent duplicating a term in order to be able to use it twice; and since every network message that is transmitted is always received by another strand, we introduce one final type of penetrator strand to model hearing and disregarding a message.

Just as our intruder starts by knowing everything in *INIT*, so the penetrator will have some initial knowledge. However, in the strand spaces model, only the keys known to the penetrator are stated: these keys form the set  $K_P$ .

In the definition that follows, we see that the penetrator may also, in a sense, say any message from the set  $T$  of atomic (non-key) messages. This means, effectively, that he is considered to know all agent identities, as we would expect; but the set  $T$  also contains all nonces. The strand spaces model is more intuitive than the rank functions approach on this count. Since a nonce is simply a random number, it is of course true to say that the penetrator knows every nonce in advance. However, since the probability of the penetrator guessing which nonce an agent has chosen is negligibly small, we shall wish to rule out attacks in which this occurs. We shall see later how this is achieved.

**Definition 3.10.** A *penetrator strand* is one of the following:

**M Text message**  $\langle +t \rangle$  for  $t \in T$ .

**F Flushing**  $\langle -x \rangle$  for  $x \in A$ .

**T Tee**  $\langle -x, +x, +x \rangle$  for  $x \in A$ .

**C Concatenation**  $\langle -x, -y, +xy \rangle$  for  $x, y \in A$ .

**S Separation**  $\langle -xy, +x, +y \rangle$  for  $x, y \in A$ .

**K Key**  $\langle +k \rangle$  for  $k \in K_P$ .

**E Encryption**  $\langle -k, -x, +\{x\}_k \rangle$  for  $x \in A, k \in K$ .

**D Decryption**  $\langle -\{x\}_k, -k^{-1}, +x \rangle$  for  $x \in A, k \in K$ .

This definition is parameterised by the set  $T$ .

Recall from Definition 3.9 that for each negative node  $n_2$  in a bundle there must be exactly one positive node  $n_1$  such that  $n_1 \rightarrow n_2$ ; that is, every reception of a message must be matched to exactly one transmission. The purpose of strands of type **F** is to allow the penetrator to ‘absorb’ transmissions of messages that he does not wish to use; strands of type **T** perform the corresponding *rôle* of replicating

messages that the penetrator wants to transmit more than once.

Recent work on strand spaces in [2, 3] has dropped this requirement of matched transmissions and receptions, allowing communications to be sent to zero nodes, one node or many nodes. Consequently, the need for penetrator strands of types **F** and **T** has been abrogated. This slightly simplifies the notation, but does not alter the expressive power of the language. Here, we follow the notation of [16] and keep these types of penetrator strand; but this decision does not affect the analysis.

### 3.4. Regular strands

A *regular strand* corresponds to a run of the protocol by an honest agent, or the actions of a trusted server. It will usually be, as is the case in our model, a specific instantiation of a strand template containing free variables. The formal definition of a regular strand is very simple.

**Definition 3.11.** A *regular strand* is any strand that is not a penetrator strand.

### 3.5. Subterms and ideals

In the rank functions model, we defined the *frags* operator to allow us to talk about the subcomponents of a message. The strand spaces equivalent is the concept of a *subterm*. The notation ‘ $t_1 \sqsubset t_2$ ’ will equate roughly to ‘ $t_1 \in frags(t_2)$ ’, except that whereas  $k \in frags(\{m\}_k)$ , we shall not have  $k \sqsubset \{m\}_k$  unless  $k \sqsubset m$ .

The  $\sqsubset$  relation is defined in terms of *ideals*. Ideals in the strand spaces model allow us to talk about all messages containing a particular submessage, when encryption is restricted to a particular set of keys.

**Definition 3.12.** Let  $k \subseteq K$  be a set of keys, and  $I \subseteq A$  a set of terms. Then we say that  $I$  is a *k-ideal* of  $A$  if

1.  $hg \in I$  and  $gh \in I$  whenever  $h \in I$  and  $g \in A$ ;
2.  $\{h\}_k \in I$  whenever  $h \in I$  and  $k \in k$ .

We write ‘ $I_k[h]$ ’ for the smallest  $k$ -ideal containing  $h$ . Similarly, if  $S$  is a set of terms, then ‘ $I_k[S]$ ’ denotes the smallest  $k$ -ideal that includes  $S$ .

**Proposition 3.13.**  $I_k[S] = \bigcup_{x \in S} I_k[x]$ .

*Proof.* Omitted; see [16]. □

**Definition 3.14.** We say that  $h \sqsubset_k m$  if  $m \in I_k[h]$ . When  $h \sqsubset_K m$ , we drop the subscript and write simply ‘ $h \sqsubset m$ ’, and say that  $h$  is a *subterm* of  $m$ .

### 3.6. Origination

This subterm relation allows us to talk about the first time that something is said on a strand. A term *originates* on a node if it is transmitted by that node without having been received, even as part of a larger message, by a previous node on the same strand. It will be *uniquely originating* if it originates on exactly one node in the strand space.

Unique origination is the mechanism alluded to above for modelling the assumption that nonces and keys cannot be simply guessed by the penetrator: if the penetrator guesses a nonce or a key, then it will not be uniquely originating. Usually, we shall disallow bundles in which there are nonces or keys that are not uniquely originating.

**Definition 3.15.** A term  $t$  is said to *originate* on a node  $n$  if

- $n$  is a positive node;
- $t \sqsubset \text{term}(n)$ ;
- whenever  $n'$  precedes  $n$  on the same strand—that is, whenever  $n' \Rightarrow^* n$ —we have  $t \not\sqsubset \text{term}(n')$ .

In addition,  $t$  is *uniquely originating* in a strand space  $\Sigma$  if there is a unique node  $n$  in  $\Sigma$  such that  $t$  originates on  $n$ .

### 3.7. Honesty

A node is an *entry point* to a set if it transmits a term without having already transmitted or received any term in the set.

**Definition 3.16.** If, for a node  $n$  of a strand  $s$ , and a set  $I \subseteq A$ ,

- $n$  is a positive node;
- the term of  $n$  is in  $I$ ;
- no previous node on  $s$ —that is, no  $n'$  with  $n' \Rightarrow^* n$ —has its term in  $I$

then we say that  $n$  is an *entry point* to  $I$ .

The idea of an *honest set* will be important to us. A set is honest relative to a given bundle if the penetrator can never break into the set except by pure fluke: either he guesses the right nonce, or he guesses the right key.

**Definition 3.17.** A set  $I \subseteq A$  is *honest* relative to a bundle  $\mathcal{C}$  if whenever a node of a penetrator strand  $s$  is an entry point to  $I$ , then  $s$  is a strand of type **M** or type **K**.

Although honesty is defined for sets in general, it is when the concepts of an honest set and an ideal are conjoined that they are most powerful. The main theorem from [16] gives conditions under which an ideal is honest.

**Theorem 3.18.** *Suppose*

1.  $\mathcal{C}$  is a bundle over  $A$ ;
2.  $S \subseteq T \cup K$ ;
3.  $k \subseteq K$ ;
4.  $K \subseteq S \cup k^{-1}$ .

*Then  $I_k[S]$  is honest.*

*Proof.* The proof is omitted; it may be found in [16].  $\square$

## 4. Example: the Otway-Rees Protocol

In this section, we give an overview of how the strand spaces machinery presented in this chapter may be used to verify security protocols.

In [16], the authors make use of the concept of an honest ideal in a strand space to prove guarantees about the Otway-Rees Protocol from [11]. Here we describe the key ideas involved in the proof; for the full details, the reader should consult [16].

The Otway-Rees Protocol may be described, in standard notation, as

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

The aim of the protocol is to provide mutual authentication of  $a$  and  $b$ , and to provide them with a fresh symmetric session key  $k$  known only to them and the trusted server  $s$ .

In the description, both  $m$  and  $na$  are nonces chosen by the initiator of the protocol. The initiator sends nonce  $m$ , his identity  $a$  and the responder's identity  $b$  to the responder, along with a component encrypted under a key shared between the initiator and the server. The responder forwards this component to the server, along with a similar component containing a nonce  $nb$  that he has chosen and encrypted under the key that he shares with the server. The server, having checked that each of  $m$ ,  $a$  and  $b$  is consistent throughout Message 2, generates a fresh session key  $k$ . He sends this session key back to the responder, once with the initiator's nonce  $na$  and encrypted under the key shared between the initiator and the server, and once with the responder's nonce  $nb$  and encrypted under the key that the responder and server share. Finally, the responder forwards the former onto the initiator to reveal  $k$  to him, and decrypts the latter to learn  $k$  himself.

Honest ideals are used in [16] to prove three guarantees about the Otway-Rees Protocol.

1. *Secrecy.* Session keys distributed by the server are never accessible to the penetrator, unless he possesses the shared key held by one of the agents for whom the server has generated the key.
2. *The initiator's guarantee.* If  $A$  has completed the protocol as initiator, believing that  $B \neq A$  was responder, then  $B$  has run the protocol as responder at least as far as Message 2, and the server has also completed the protocol. Furthermore,  $A$ ,  $B$  and the server agree on the value of  $m$ ;  $A$  and the server agree on  $na$  and  $k$ ; and  $B$  and the server agree on  $nb$ .
3. *The responder's guarantee.* If  $B$  has completed the protocol as responder, believing that  $A \neq B$  was initiator, then  $A$  has started the protocol as initiator, and the server has also completed the protocol. Furthermore,  $A$ ,  $B$  and the server agree on the value of  $m$ ; and  $B$  and the server agree on  $nb$  and  $k$ .

Note that it is not possible to guarantee that, at the end of the protocol, the initiator and responder agree on the session key  $k$ . However, the weakness is not as problematic as one might suppose, since the penetrator will know neither the session key that  $A$  accepts, nor the session key that  $B$  accepts.

In the next two sections, we shall consider these guarantees, giving an overview of how each is proved using an honest ideal.

#### 4.1. Secrecy

The aim is to prove that server-generated session keys are never revealed to the penetrator. It is sufficient to show that this is the case when the server believes that the initiator and responder are  $A$  and  $B$  respectively.

The procedure is to show that if the server generates session key  $K$  for agents  $A$  and  $B$  then the penetrator can never break into the set  $S = \{SH(A, S), SH(B, S), K\}$ .

Since the penetrator starts holding none of these keys, he cannot decrypt messages encrypted under any of these keys. For example, since he does not have access to  $SH(A, S)$ , it would be safe to allow him to see  $\{K\}_{SH(A, S)}$ . However, since he may hold other keys,  $\{K\}_j$  for some key  $j^{-1} \notin S$  should be kept secret from him.

We know how to construct the set of all messages that should be kept from the penetrator's grasp. Any message containing a member of  $S$  that is not encrypted under a key whose inverse is in  $S$  is out of bounds. If we let  $k = (K \setminus S)^{-1}$ , then the set of messages to be protected is the ideal  $I_k[S]$ . In [16],  $k = K \setminus S$ ; but since  $S$  contains only self-inverse keys, the two definitions are the same.

Proving that the Otway-Rees Protocol satisfies the secrecy requirement, therefore, involves demonstrating that, for bundles in an Otway-Rees strand space in which the

penetrator does not guess  $K$  (that is, in which  $K$  is uniquely originating), no element of  $I_k[S]$  is ever transmitted.

The honesty of this ideal follows from Theorem 3.18. A corollary of this theorem, stated and proved in [16], allows us to conclude that if there is a node that qualifies as an entry point for  $I_k[S]$ , then it must be a regular node.

A case analysis shows that a regular node cannot, in fact, be an entry point for  $I_k[S]$ . Thus, no node is an entry point for  $I_k[S]$  and the secrecy property holds.

#### 4.2. The initiator's and responder's guarantees

The first step in proving the authentication goals is to show that any occurrence of a message encrypted under  $SH(x, s)$ , where the penetrator does not already hold this key, originates on a regular strand.

This is done by setting  $S = \{SH(x, s)\}$  and  $k = (K \setminus S)^{-1} = K \setminus S$  and considering the ideal  $I_k[S]$ . (In [16], the authors have  $k = K$ , but it makes no difference to the argument.)

This ideal is, again, an honest ideal by Theorem 3.18. A second corollary of the theorem in [16] shows that, since the key in  $S$  never originates on a regular node—that is, is never given away by an honest principal or by the server—terms encrypted under  $SH(x, s)$  can originate only on a regular node.

The authentication guarantees can now be proved by appealing to the form of the messages involved.

In the case of the initiator's guarantee, one argues that if the initiator receives Message 4 with the encrypted component as  $\{N_A \cdot K\}_{SH(A, S)}$  then this component must have originated on a regular node; and from the structure of the messages produced by the participants, this must be a regular node on a server strand. This is either the first encrypted component or the second encrypted component of a Message 3; and in the latter case one may derive a contradiction regarding the unique origination of  $N_A$ , so it must be the former. But then the server must have received a corresponding Message 2, which also must have originated on a regular node; and so, by arguing back in this way, one may conclude that the responder ran the protocol with appropriate parameters at least as far as Message 2.

A similar argument shows that the Otway-Rees Protocol correctly ensures the validity of the responder's guarantee.

### 5. The link with rank functions

Having had a taste of how the power of strand spaces may be harnessed to prove correctness of security protocols, we now consider the connection between rank functions and strand spaces.

Meadows, in [9], compares several different approaches to security protocol analysis, including the strand spaces method and the rank functions model. She notes that the

set of messages of rank of zero is used in a similar way to that of an honest ideal in a strand space. We here seek to investigate this issue further, drawing out the nature of the correspondence in detail.

### 5.1. Verifying protocols using strand spaces

It should be noted that each guarantee about the Otway-Rees Protocol was proved by considering some honest ideal  $I_k[S]$  with the set  $S$  containing only atoms; and in particular that in each case we had  $k = (K \setminus S)^{-1}$ . This equality is no accident.

The intuitive purpose of Theorem 3.18 is to determine, given a set  $S$  of atoms that we wish to keep from the penetrator, the larger set of messages that it is dangerous to let him see, lest he should use a message in the larger set to deduce the value of one of the atoms in  $S$ . In constructing the  $k$ -ideal of  $S$ , we find all those messages that must be kept from the penetrator to ensure that we deny him access to  $S$ , assuming that he holds the inverses of those keys in  $k$  but holds no other keys.

Now, if we are attempting to keep everything in  $S$  from the penetrator, but are willing to allow him access to atoms outside  $S$ , then we already know which keys we should expect him to be able to learn, and which keys must remain secret. Any key  $k \in S$  will have to be kept from the penetrator, and so he will not be able to decrypt messages encrypted under  $k^{-1}$ ; and any key  $j \notin S$  will be one that the penetrator might learn, so messages encrypted under  $j^{-1}$  will be visible to him.

We can use this to state a corollary of Theorem 3.18 that gives conditions relating the sets  $S$  and  $k$  that must be met in order for the ideal to be honest. The corollary allows us to concentrate solely on finding an appropriate set  $S$  of atoms, and to have a suitable  $k$  constructed for us.

**Corollary 5.1.** *Suppose*

1.  $C$  is a bundle over  $A$ ;
2.  $S \subseteq T \cup K$ .

*Then  $I_{(K \setminus S)^{-1}}[S]$  is honest.*

*Proof.* We let  $k = (K \setminus S)^{-1}$ , and now claim that Theorem 3.18 applies. The first two conditions of the theorem are the same as those of the corollary. We have that  $K \setminus S \subseteq K$ , and  $K$  is closed under taking inverses, so  $(K \setminus S)^{-1} \subseteq K$ , and the third condition of Theorem 3.18 is satisfied. For the fourth,  $K = (K \cap S) \cup (K \setminus S) = (K \cap S) \cup ((K \setminus S)^{-1})^{-1} = (K \cap S) \cup k^{-1}$ , and so certainly  $K \subseteq S \cup k^{-1}$ . So all four conditions have been met, and the theorem thus assures us that  $I_{(K \setminus S)^{-1}}[S]$  is honest.  $\square$

The set  $k = (K \setminus S)^{-1}$  is the smallest set of keys that will satisfy the conditions of Theorem 3.18. The ideal of

Corollary 5.1 is the smallest honest ideal that includes the set  $S$ . As such, we shall refer to the set  $I_{(K \setminus S)^{-1}}[S]$  as the *minimal honest ideal* of  $S$ . We shall often drop the subscript and write simply ' $I[S]$ ' as a shorthand for ' $I_{(K \setminus S)^{-1}}[S]$ '.

**Remark 5.2.** Note that in the case that  $S \subseteq A$  but  $S \not\subseteq T \cup K$ , Corollary 5.1 will not apply. We shall still have cause to consider  $I[S]$ , but we shall have no guarantee that it is an honest set.

Proving a security property of a security protocol in the strand spaces model will usually involve identifying the set  $S$  of crucial atomic secrets, constructing its minimal honest ideal, and showing, *via* Theorem 3.18 or a suitable corollary, that the penetrator can never break into the set  $I[S]$ .

The approach can be summarised as follows.

1. Identify the set  $S$  of atoms that should remain secret.
2. Find the closure of the set  $S$  under operations that leave members of  $S$  unprotected; that is, under concatenation with arbitrary messages, and encryption under inverses of unprotected keys. The resulting set is  $I[S]$ .
3. Show that the penetrator can never break into the set  $I[S]$ , by demonstrating that the regular strands will not give out messages from  $I[S]$  unless they have already received a message from  $I[S]$ .
4. Conclude from this that the protocol satisfies the specified security property.

### 5.2. Verifying protocols using rank functions

A proof of correctness in the rank functions model is completed by finding some rank function  $\rho$  on the message space with the properties given in Theorem 2.4.

Although any such rank function will suffice, we have in this paper concentrated on the particular minimal 1-set rank function  $\rho_0$  that assigns a rank of one only when required by the conditions of the theorem. The procedure has been as follows.

1. Identify the set  $INIT$  of atoms that the enemy knows from the start.
2. Find the closure of the set  $INIT$  under operations that the enemy can perform; that is, under concatenation, deconcatenation, encryption using known keys, decryption using known keys, and picking up messages transmitted by honest agents. The resulting set is  $X$ , and  $\rho_0$  is the characteristic function of  $X$ .
3. Show that the crucial messages in  $T$  are not members of  $X$ .

4. Conclude from this that the protocol satisfies the specified security property.

There are two major differences between this procedure and that given for strand spaces. First, the actions of honest agents are considered earlier than in the strand spaces model. Closure of the set *INIT* includes closure under operations that honest agents may perform on messages. However, in the strand spaces model, closure of *S* to form  $I[S]$  does not take the actions of honest agents into account; regular strands are not considered until the third step. Second, the two procedures attack the problem from opposite points of view: whereas the strand spaces approach concentrates on messages that should remain secret, construction of  $\rho_0$  looks specifically for all messages that must be public.

### 5.3. Is the kernel of a rank function an honest ideal?

In light of the fact that the strand spaces model characterises all messages that must remain secret in terms of honest ideals, it is natural to ask whether anything can be said about the corresponding messages in the world of rank functions. More specifically, is the kernel  $\ker \rho$  of a rank function  $\rho$ —that is, the set of messages having a rank of zero—an honest ideal?

#### 5.3.1 The kernel of a general rank function

Recall that the minimal 1-set rank function  $\rho_0$  will not in general be the only rank function that can be found to prove a given protocol correct.

Suppose that we have found some rank function  $\rho$ , not necessarily the minimal 1-set rank function, that proves that our protocol satisfies a particular security property. What can be said of  $\ker \rho$ , the kernel of the rank function  $\rho$ ?

**Proposition 5.3.** *For any rank function  $\rho$  that satisfies the conditions of Theorem 2.4,  $\ker \rho$  is an ideal.*

*Proof.* We claim that  $\ker \rho = I[\ker \rho]$ .

Let  $k = (K \setminus \ker \rho)^{-1}$ . Now  $I[\ker \rho] = I_k[\ker \rho]$ , which is defined to be the smallest  $k$ -ideal that includes  $\ker \rho$ , so certainly we have  $\ker \rho \subseteq I[\ker \rho]$ .

Suppose that  $x \in I[\ker \rho]$ . We know from Proposition 3.13 that

$$x \in \bigcup_{m \in \ker \rho} I_k[m]$$

and so  $x \in I_k[m]$  for some  $m$  with  $\rho(m) = 0$ . But  $I_k[m]$  is the set formed by closure of  $\{m\}$  under concatenation with arbitrary messages, and encryption under keys from  $k$ ; that is, we can obtain  $x$  from  $m$  by some finite sequence of such concatenations and encryptions. Each of these operations preserves the property of having rank zero:

- If  $\rho(g) = 0$  then  $\rho(g.h) = 0$ . For  $\{g.h\} \vdash g$ , and so if  $\rho(g.h) = 1$  then the conditions on  $\rho$  force  $\rho(g) = 1$ , yielding a contradiction.

- If  $\rho(g) = 0$  and  $\rho(k^{-1}) = 1$  then  $\rho(\{g\}_k) = 0$ . If  $k \in k$  then  $k^{-1} \in k^{-1} = K \setminus \ker \rho$ , so  $k^{-1} \notin \ker \rho$  and we do indeed have that  $\rho(k^{-1}) = 1$ . But now  $\{\{g\}_k, k^{-1}\} \vdash g$ , so if  $\rho(\{g\}_k) = 1$  then the conditions on  $\rho$  would give  $\rho(g) = 1$ , again yielding a contradiction.

Thus  $\rho(x) = 0$ , and so  $x \in \ker \rho$ . This completes the proof that  $\ker \rho = I[\ker \rho]$ , and so  $\ker \rho$  is indeed an ideal.  $\square$

Recall from Remark 5.2 that, since  $\ker \rho$  does not contain only atoms, we shall have no guarantee from Corollary 5.1 that  $I[\ker \rho]$  is an honest set. However, closer inspection reveals that we may be assured of its honesty by considering the definition of a rank function.

Before continuing, it will be well to clarify exactly what we mean by asking whether  $\ker \rho$  is an honest set. Strictly speaking, the question makes no sense; for honesty is defined only for sets of terms in a strand space, and is defined in terms of nodes on strands, and these types have no meaning in the rank functions model. However, the analogues of these notions are not far to seek.

The equivalent, in the rank functions world, of a regular strand is a trace of a process. The process in question will be  $USER_J$  for some  $J \in \mathcal{U}$ , or else the process  $SERVER$  controlling the server; since a regular strand represents a single run of a protocol, the trace of the process will be a trace also representing a single run.

The analogue of a penetrator strand is an action performed by the enemy. Penetrator strands of type **C**, **S**, **E** and **D** correspond to deductions under  $\vdash$  of concatenation, deconcatenation, encryption and decryption respectively; type **T** disappears entirely since the enemy may transmit any message he knows any number of times; types **M** and **K** correspond to the enemy transmitting messages from the set *INIT*; the analogue of type **F** is reception of a message for which the enemy finds no further use.

Recall that an entry point to a set  $I$  is a node that transmits a term from the set without any preceding node on the same strand having previously received any term from  $I$ . For a regular strand, the rank functions equivalent is transmission of a message in  $I$  by a process representing an honest agent or the server, when the process has not previously received a message in  $I$  during that run. For a penetrator strand, it is immediately clear that the strand must be of type **C**, **S**, **E**, **D**, **M** or **K**, because **F** strands transmit nothing and **T** strands transmit terms that they have previously received; so the analogue will be either an enemy deduction of the form  $S \vdash m$  with  $s \notin I$  for every  $s \in S$  but with  $m \in I$ , or else a transmission of some element of *INIT*.

Finally, honesty is defined relative to a given bundle, and a bundle corresponds to a trace of the network. But we are interested only in whether sets are honest relative to every bundle of a particular strand space; or, in the rank functions

model, whether the equivalent property holds for every possible trace of the network.

This allows us to give the equivalent definition of honesty in the rank functions model. In the strand spaces model, a set  $I$  is honest if whenever a penetrator node is an entry point to  $I$  then it is of type **M** or of type **K**. We have seen that the type cannot in any case be of type **T** or type **F**; so the definition effectively states that the strand must not be of type **C**, **S**, **E** or **D**.

The following definition is analogous to Definition 3.17.

**Definition 5.4.** A set  $S \subseteq \mathcal{M}$  is *honest* if whenever the enemy can transmit an element  $m \in S$  then  $m \in \text{INIT}$ .

Now the statement that the kernel of a rank function is honest becomes almost trivial.

**Proposition 5.5.** For any rank function  $\rho$  that satisfies the conditions of Theorem 2.4,  $\ker \rho$  is honest.

*Proof.* We are required to show that the enemy can never learn any element  $m$  of  $\ker \rho$  unless  $m \in \text{INIT}$ .

In fact, we know that if  $m \in \text{INIT}$  then the conditions of Theorem 2.4 force  $\rho(m) = 1$ , so the statement is equivalent to saying that the enemy can never learn any element of  $\ker \rho$ .

But this is immediate. The enemy builds on his initial knowledge by deductions under  $\vdash$ , and by learning messages transmitted by honest agents and the server.

In the case of deductions, he can never break into  $\ker \rho$  by making a deduction  $S \vdash m$  from messages of rank one to learn a message of rank zero, for the conditions of the theorem insist that if  $S$  contains only messages of rank one then so must  $m$  have rank one.

In the case of transmissions by honest agents or the server, the enemy will never be able to use messages of rank one to persuade an honest agent or server to reveal a message with rank zero; for the condition that each such process must maintain the rank says exactly that if the process has received only messages of rank one then it will not transmit a message with a rank of zero.

The enemy's knowledge is restricted, then, to messages that have a rank of one; that is, messages that lie outside  $\ker \rho$ . Thus,  $\ker \rho$  is an honest set.  $\square$

The rank function, then, is a way of categorising which messages the enemy may learn. By definition,  $\ker \rho$  tells us the messages that will be inaccessible to him. It should be no surprise to learn that  $\ker \rho$  is always honest.

However, this is not the end of the matter. Theorem 3.18 gives conditions under which an ideal  $I_k[S]$  is honest; but it requires that  $S$  be a set of atoms. The most useful honest ideals are indeed of the form  $I_k[S]$  for some set  $S$  containing only atoms; and it is instructive to consider whether we can write  $\ker \rho$  in this form.

Unfortunately, we cannot. Consider the following protocol, proposed by Peter Ryan, and first published (as far as the author is aware) in [5]:

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

The purpose of the protocol is to authenticate the initiator to the responder, making use of keys shared with the trusted server. It has been proven correct, by use of rank functions, in [4].

Let  $\rho_0$  be the minimal 1-set rank function for Ryan's Protocol, and suppose that  $\ker \rho_0 = I_k[S]$  for some set  $S$  of atoms.

Consider the message  $\{A\}_{SH(S,B)}$ . It is clear that the enemy can never learn this message, for he does not possess the shared key to perform the encryption himself, and no message of this form appears anywhere in the protocol so he will not learn it from an honest agent or from the server. Since  $\rho_0$  is the minimal 1-set rank function, we must have  $\rho_0(\{A\}_{SH(S,B)}) = 0$  and hence  $\{A\}_{SH(S,B)} \in \ker \rho_0$ .

But if  $\{A\}_{SH(S,B)} \in I_k[S]$ , and  $S$  contains only atoms, then we must have  $A \in S$  and  $SH(S,B) \in k$ . But if  $A \in S$  then  $A \in I_k[S]$ , and so  $A \in \ker \rho_0$ . This is a contradiction, for we have  $A \in \text{INIT}$  and so  $\rho_0(A) = 1$ . Thus  $\ker \rho_0$  cannot be written in the form  $I_k[S]$  with  $S$  being a set of atoms.

We might ask whether this problem occurs simply because the message that we have considered is a message that cannot appear as a fragment of a protocol message; that is, a message outside  $\mathcal{M}^0$ . Can we rescue the situation by asking only that  $\ker \rho$  should agree with some  $I_k[S]$  on elements of  $\mathcal{M}^0$ ? Can we ensure that

$$(\ker \rho) \cap \mathcal{M}^0 = I_k[S] \cap \mathcal{M}^0$$

for some set  $S$  containing only atoms?

Again, the answer is no. Let us make a small change to the second message of Ryan's Protocol:

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

This alteration does not affect the correctness of the protocol. Nor does it affect the rank of  $\{A\}_{SH(S,B)}$ , for this component never appears except inside a Message 2 encrypted with  $SH(S,B)$ , which key the enemy does not hold. However, it does ensure that  $\{A\}_{SH(S,B)} \in \mathcal{M}^0$ ; and, by reasoning similar to that employed before, we see that we shall have  $A \in \ker \rho_0$  and yet  $\rho_0(A) = 1$ , yielding a contradiction.

### 5.3.2 The kernel of the minimal 1-set rank function

It is perhaps not surprising that the kernel of a general rank function cannot be written in a convenient form, for there might be any number of rank functions that satisfy the conditions of Theorem 2.4 for a given security protocol.

However, we might have held higher hopes for the minimal 1-set rank function. Since this rank function is in no way arbitrarily chosen, and has a clear structure, we might reasonably have expected that the link between the rank functions model and the strand spaces model would be at its strongest here.

But the argument above shows that this is not the case. Our counter-example to the general case comes in the form of the minimal 1-set rank function for a slightly altered version of Ryan’s Protocol.

### 5.4. Rank functions and secrecy specifications

The reason that the kernel of a rank function does not turn out to have the structure we hoped that it would is that an honest ideal in the strand spaces model does not serve quite the same purpose as the kernel of a rank function in the rank functions model.

The kernel of a rank function specifies those messages that the enemy should never be allowed to see. Recall that when considering an authentication specification we block agent  $A$  from starting a particular run of the protocol with agent  $B$  in order to discover whether the enemy can masquerade as  $A$ ; so that if the authentication specification is satisfied, the blocked protocol messages are never learnt by the enemy. The approach is to prove that if agent  $A$  never initiates the run then those messages never appear.

Honest ideals on strand spaces may be used without any necessity for blocking particular agents from engaging in particular runs. In the proof of the authentication guarantees for the Otway-Rees Protocol, it is nowhere claimed that the enemy never learns the messages that agent  $A$  generates in order to initiate the protocol with agent  $B$ . In fact, he does learn them; but not until agent  $A$  generates them and transmits them. The approach is not to prove that the messages never appear, but to prove that when they do appear they must have originated on a regular strand representing agent  $A$ .

The rank functions model can be used to prove secrecy properties as well as authentication properties; and, in this case, the kernel of the rank function takes on a *rôle* much like that of the honest ideal.

#### 5.4.1 Rank functions for secrecy

Where the rank functions theorem applies, it concludes that

$$\text{NET sat } R \text{ precedes } T$$

for particular sets  $R$  and  $T$ ; and the idea is to show that if we stop events in  $R$  from occurring then events in  $T$  can never happen. When considering authentication, we choose  $R$  to indicate that a particular agent has started a given *rôle* of a particular run, and  $T$  to state that another agent has finished the complementary *rôle* of the same run. For secrecy specifications, we need not block any agent activity: we wish simply to state that a certain message, almost always a session key, is never learnt by the enemy. We need, therefore, to let  $R = \emptyset$ , and choose  $T$  to indicate that the enemy has learnt the erstwhile secret. The statement that

$$\text{NET sat } \emptyset \text{ precedes } T$$

is just another way of saying that events from  $T$  never occur.

The definition of the process representing the enemy needs to be augmented slightly: it should allow the enemy, at any time, to engage in an event *says.x* for any message  $x$  that he knows. Then we let  $T = \{\text{says.K}\}$  for the key  $K$  that we wish to keep secret. As with authentication, we shall concentrate our attention on one specific run of the protocol. The key  $K$  will be a particular key that we shall expect to be allocated during this specific run; and, as with authentication, if we can show that the secrecy specification holds for this run, we shall conclude that it holds in general.

#### 5.4.2 Secrecy of session keys

In this context, with no blocking, we find that we can define a rank function whose kernel is an honest ideal of the form  $I_k[S]$ .

The enemy will still have access to the set *INIT* representing his initial knowledge. We shall augment *INIT* in this case with all possible session keys and nonces from all runs except for the one run that concerns us. The remaining atoms—long-term keys, and the secrets from the specific run of the protocol on which we shall be concentrating—will go to make up the set  $S$ .

We now define a rank function  $\rho^*$  such that

$$\rho^*(m) = \begin{cases} 0 & (m \in I[S]) \\ 1 & (m \notin I[S]) \end{cases}$$

If a rank function exists to prove the security specification,  $\rho^*$  will be such a rank function. For this rank function assigns a rank of zero only when absolutely necessary: it is the maximal 1-set rank function.

In this case, we have successfully constructed a rank function whose kernel is a minimal honest ideal of a set of atoms.

#### 5.4.3 Caveat lector

Although some work has been done on the completeness of the rank function theorem in terms of its application to

authentication protocols running on unbounded networks—that is, on the issue of whether every secure authentication protocol has an associated rank function to prove its security—the question remains open [4]. However, when we consider secrecy specifications, it is not difficult to see that the rank function theorem is not complete in any sense.

Consider the following protocol:

Message 1.  $a \rightarrow b : \{a.na\}_{PK(b)}$   
 Message 2.  $b \rightarrow a : \{b.na.k\}_{PK(a)}$   
 Message 3.  $a \rightarrow b : \{m\}_k.na$

The purpose of the protocol is to allow agent  $a$  to send some message  $m$  to  $b$ , under a session key  $k$  generated by  $b$ . First,  $a$  sends a nonce challenge to  $b$ : he requires  $b$  to decrypt  $na$  and send it back. In the second message,  $b$  obliges by returning  $na$ , along with a freshly generated session key  $k$ , encrypted under  $a$ 's public key. Finally,  $a$  sends his message to  $b$  encrypted with this session key, and appends the nonce  $na$  in cleartext.

The protocol correctly provides for secrecy of the message  $m$ . It is, in fact, a simple variant on Lowe's fixed version [7] of the Needham-Schroeder Public-Key Protocol [10]:

Message 1.  $a \rightarrow b : \{a.na\}_{PK(b)}$   
 Message 2.  $b \rightarrow a : \{b.na.nb\}_{PK(a)}$   
 Message 3.  $a \rightarrow b : \{nb\}_{PK(b)}$

There are two differences.

1. Rather than send back  $k$  (which stands in place of  $nb$ ), encrypted, to prove that the initiator knows the value, it is used to encrypt the secret message  $m$ . This has the same effect: no-one can recover  $k$  from  $\{m\}_k$ , but the responder can verify from the encryption that the sender knows  $k$ .
2. In the third message, the value of  $na$  is leaked. This does not allow an attack, because by this point the nonce has served its purpose. It is not leaked until after the initiator has verified that the responder knows  $na$ . The initiator will not use the same nonce value twice, and so the knowledge of the value of  $na$  is of no use to the enemy.

Although this protocol is secure, however, it cannot be proved so by appeal to rank functions. Suppose that we consider the run

Message 1.  $A \rightarrow B : \{A.N_A\}_{PK(B)}$   
 Message 2.  $B \rightarrow A : \{B.N_A.K\}_{PK(A)}$   
 Message 3.  $A \rightarrow B : \{M\}_K.N_A$

and look for secrecy of the message  $M$  in this run; that is, we aim to show that the enemy can never get hold of  $M$ . What rank shall we assign to the nonce  $N_A$ ?

The fact that nonce  $N_A$  is leaked in the third message forces us to set  $\rho(N_A) = 1$ . Formally, we have  $\rho(\{A.N_A\}_{PK(B)}) = 1$  because this message can be sent out by  $A$  without  $A$  needing to have received any inputs; then  $\rho(\{B.N_A.K\}_{PK(A)}) = 1$  because  $B$  can send this out having received only a message of rank one (namely,  $\{A.N_A\}_{PK(B)}$ ); and so  $\rho(\{M\}_K.N_A) = 1$  because  $A$  can transmit this having received only a message of rank one (namely,  $\{B.N_A.K\}_{PK(A)}$ ). Finally,  $\rho(N_A) = 1$  because  $\{M\}_K.N_A$  has rank one, and  $\{M\}_K.N_A \vdash N_A$ .

However, if  $\rho(N_A) = 1$ , then also  $\rho(M) = 1$ . For if  $\rho(N_A) = 1$  then  $\rho(\{B.N_A.K'\}_{PK(A)}) = 1$  for any  $K'$  chosen by the enemy. But now  $A$  can transmit  $\{M\}_{K'}.N_A$  having received only a message of rank one; this gives us that  $\rho(\{M\}_{K'}) = 1$ , and so  $\rho(M) = 1$ . So the enemy, it seems, can get hold of  $M$  after all.

How has this occurred? Let us try to turn the above argument into an attack.

Message 1.  $A \rightarrow C(B) : \{A.N_A\}_{PK(B)}$   
 Message 2.  $C(B) \rightarrow A : \{B.N_A.K'\}_{PK(A)}$   
 Message 3.  $A \rightarrow C(B) : \{M\}_{K'}.N_A$

This 'attack' does not work. The enemy cannot produce the second message, because  $N_A$  is not in his possession at this point: he learns it only later on, after the third message.

Having sent out nonce  $N_A$  in the first message, the initiator is willing to accept any key as the third component of the second message. But, of course, he will accept only one value of the key. The initiator will allow  $k = K$ ; he will allow  $k = K'$ ; but since he will not reuse the nonce  $N_A$ , there is no possibility of accepting  $\{B.N_A.K\}_{PK(A)}$  at one point and then accepting  $\{B.N_A.K'\}_{PK(A)}$  at another.

But the rank functions approach cannot cater for this. If, after receiving only inputs with a rank of one, an agent can *either* send message  $m$  out, *or* send message  $m'$  out, then the rank functions model insists on *both* messages  $m$  and  $m'$  having a rank of one. This causes the headache above: because agent  $A$ , acting as initiator, is willing to accept  $\{B.N_A.K\}_{PK(A)}$  in response to his first message, it follows that  $N_A$  must have a rank of one; because he is willing to accept  $\{B.N_A.K'\}_{PK(A)}$ , and because  $\rho(N_A) = 1$ , it follows that  $M$  must have a rank of one. The inability of rank functions to distinguish between mutually exclusive actions and compossible actions throws up a false attack.

This does not cause soundness problems for the rank function theorem, but it does impact on completeness: some protocols, at least for secrecy specifications, are secure, but cannot be proved secure using rank functions. Specifically, secrecy specifications for correct protocols that fail to meet Lowe's *no temporary secrets* requirement [8] will often not be verifiable using rank functions.

It would appear that authentication specifications are not significantly affected by this issue. With our present understanding, it seems that the rank functions theorem is bet-

ter suited to verifying authentication properties than secrecy properties. The parallelism with *STOP* over events in  $R$  in conditions 4 and 5 of Theorem 2.4 for authentication specifications appears to defeat the problem noted above; but this is the subject of ongoing research.

## Acknowledgements

The author owes a debt of gratitude to Steve Schneider for his helpful comments on this paper. Thanks are also due to Joshua Guttman, Steve Hailes, and Daniel Hill for reading an earlier draft.

Part of the work that resulted in this paper was conducted while the author was at Royal Holloway, University of London.

## References

- [1] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.
- [2] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication Tests. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Security Press, May 2000.
- [3] Joshua D. Guttman and F. Javier Thayer Fábrega. Protocol Independence through Disjoint Encryption. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 24–34, June 2000.
- [4] 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.
- [5] 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.
- [6] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.
- [7] Gavin Lowe. An attack on the Needham-Schroeder public key protocol. *Information Processing Letters*, 56:131–133, 1995.
- [8] Gavin Lowe. Towards a completeness result for model checking of security protocols. Technical Report 1998/6, Department of Mathematics and Computing Science, University of Leicester, 1998.
- [9] Catherine A. Meadows. Invariant Generation Techniques in Cryptographic Protocol Analysis. *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 159–167, June 2000.
- [10] 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.
- [11] Dave Otway and Owen Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, 1987.
- [12] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall International, 1998.
- [13] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *The Modelling And Analysis Of Security Protocols*. Addison Wesley, July 2000.
- [14] Steve A. Schneider. Verifying authentication protocols in CSP. *IEEE TSE*, 24(9), September 1998.
- [15] Steve A. Schneider. *Concurrent and real-time systems: the CSP approach*. John Wiley & Sons, 1999.
- [16] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. *Proceedings of 11th IEEE Computer Security Foundations Workshop*, June 1998.
- [17] 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.
- [18] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. *Proceedings of 12th IEEE Computer Security Foundations Workshop*, pages 72–82, June 1999.
- [19] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.