# Analysing the Woo-Lam Protocol Using CSP and Rank Functions

**Siraj Shaikh and Vicky Bush**
Department of Computing,
University of Gloucestershire Business School,
Park Campus, Cheltenham Spa, GL52 2RH, UK
{sshaikh,vbush}@glos.ac.uk

*Designing security protocols is a challenging and deceptive exercise. Even small protocols providing straightforward security goals, such as authentication, have been hard to design correctly, leading to the presence of many subtle attacks. Over the years various formal approaches have emerged to analyse security protocols making use of different formalisms. Schneider has developed a formal approach to modelling security protocols using the process algebra CSP (Communicating Sequential Processes). He introduces the notion of rank functions to analyse the protocols. We demonstrate an application of this approach to the Woo-Lam protocol. We describe the protocol in detail along with an established attack on its goals. We then describe Schneider's rank function theorem and use it to analyse the protocol.*

*ACM Classification: C.2.2 (Communication/Networking and Information Technology – Network Protocols – Protocol Verification), D.2.4 (Software Engineering – Software/Program Verification – Formal Methods), D.4.6 (Operating Systems – Security and Privacy Protection – Authentication)*

## 1. INTRODUCTION

Over the years various formal approaches have emerged to analyse security protocols (Ryan *et al*, 2001), making use of different formalisms including logic (Gong *et al*, 1990; Syverson and van Oorschot, 1994) strand spaces (Thayer Fábrega, 1998), type theory (Gordon and Jeffrey, 2001), model-checking (Lowe, 1996; Mitchell *et al*, 1997) and some hybrid techniques (Meadows, 1996). A common principle in this research is the assumption of 'perfect encryption' that allows cryptography to be treated as a black-box and therefore flawless; this is formalised by Dolev and Yao (1983). Schneider (1996) has developed a formal approach to modelling security protocols using the process algebra CSP (Hoare, 1985). Schneider (1998) then introduces the notion of rank functions to analyse the protocols.

The purpose of this paper is to demonstrate an application of this approach to the Woo-Lam protocol (Woo and Lam, 1992). It is an authentication protocol with a history of established attacks. We describe the Woo-Lam protocol in Section 2, along with an attack in Section 2.1. We then describe Schneider's CSP approach and model the Woo-Lam protocol in CSP in Section 3. We introduce the rank function approach in relevant detail and apply the approach to the Woo-Lam protocol in Section 4. We discuss our experiences of this effort to conclude the paper in Section 5.

*Manuscript received: 12 April 2005*
Communicating Editor: Julio Cesar Hernandez

## 2. WOO-LAM PROTOCOL

Woo and Lam (1992) introduce a protocol that provides one-way authentication of the initiator of the protocol, *A*, to a responder, *B*. The protocol uses symmetric-key cryptography and a trusted third-party server, with whom *A* and *B* share long-term symmetric keys.

$$\begin{array}{lccl}
(1) & A \rightarrow B & : & A \\
(2) & B \rightarrow A & : & N_B \\
(3) & A \rightarrow B & : & \{N_B\}_{KAS} \\
(4) & B \rightarrow S & : & \{A,\{N_B\}_{KAS}\}_{KBS} \\
(5) & S \rightarrow B & : & \{N_B\}_{KBS}
\end{array}$$

**Figure 1: Woo-Lam protocol**

The protocol is shown in Figure 1 above where $\{m\}_k$ represents message *m* encrypted under key *k* and "," represents the concatenation operator. The keys $K_{AS}$ and $K_{BS}$ represent the long-term keys that *A* and *B* share with the trusted server *S*. The protocol goal is to authenticate *A* to *B* by using a fresh and unpredictable nonce, $N_B$, produced by *B*.

*A* starts the protocol by sending its identity to *B*. *B* replies by sending a freshly generated nonce $N_B$. *A* encrypts $N_B$ with key $K_{AS}$ and sends it back to *B*. *B* concatenates *A*'s reply with the identity of *A*, encrypts it with key $K_{BS}$ and sends it to the server *S*. *S* sends out $N_B$ back to *B* encrypted under $K_{BS}$. *B* compares the nonce it receives from *S* with the one it sent out to *A*. If they match, then *B* is guaranteed that the initiator of the protocol is in fact the principal claimed in the first step of the protocol.

### 2.1 An attack on the Woo-Lam protocol

The Woo-Lam protocol has been to shown to be susceptible to a few attacks (Woo and Lam, 1994), one of which is shown in Figure 2 below.

$$\begin{array}{llcl}
(1.1) & I(A) \rightarrow B & : & A \\
(1.2) & B \rightarrow I(A) & : & N_B \\
(1.3) & I(A) \rightarrow B & : & X \\
 & (2.1)\ I \rightarrow B & : & I \\
 & (2.2)\ B \rightarrow I & : & N_B' \\
 & (2.3)\ I \rightarrow B & : & \{N_B\}_{KIS} \\
(1.4) & B \rightarrow S & : & \{A,X\}_{KBS} \\
 & (2.4)\ B \rightarrow S & : & \{I,\{N_B\}_{KIS}\}_{KBS} \\
(1.5) & S \rightarrow B & : & \{N_B\}_{KBS}
\end{array}$$

**Figure 2: An attack on the Woo-Lam protocol**

The attack shows two simultaneous inbound authentication attempts initiated by an intruder *I*, where *I* is also considered as any other regular participant. *I* pretends to be *A* in one (1.*x*) and retains its own identity *I* for the other (2.*x*). *I* obtains nonces from *B* for both runs and encrypts the nonce $N_B$ intended for *A* with its own server key and returns it to *B*, retaining its original identity. When the nonce is returned by the server, it leads *B* to believe that it has authenticated *A*, whereas *A* has not even participated in either of the runs. The attack is complete.

The attack shown in Figure 2 demonstrates the difficulty in designing such protocols and emphasises the need for a formal and rigorous analysis of these protocols.

## 3. SCHNEIDER'S CSP APPROACH

In order to deal with the problem highlighted in the previous section, Schneider presents a formal framework that uses the process algebra CSP to model protocols. We present Schneider's CSP approach in detail, describing the relevant syntax for CSP and its trace semantics in Section 3.1. In Section 3.2 we model the participants in the Woo-Lam protocol as CSP processes and specify a network composed of these processes. We then present a trace specification that the network needs to satisfy for the protocol to hold correct. Finally we adopt a proof strategy to verify this network.

While we discuss this notation in detail relevant to our usage in this paper, we take for granted the reader's basic knowledge of CSP and its use by Schneider (1998) to model security protocols; in-depth treatments of CSP are provided by Hoare (1985) and, more relevantly, Ryan *et al* (2001).

### 3.1 CSP Events and Processes

A CSP system is modelled in terms of processes and events that these processes can perform, which are essentially instances of communication, usually involving a channel and some data value. Events may be atomic in structure or may consist of distinct components.

The CSP expression $a \rightarrow P$ describes a process $P$ with event $a$ in the interface of $P$. The process is initially able to perform $a$ and then behaves as $P$. The process *STOP* is the simplest CSP process that can be described; it has no event transitions and does not engage in any events. The *choice* operator $\square$ provides the option for running either of the two processes, $P$ and $Q$ for example, when put together as $P \;\square\; Q$. The *parallel* operator $\underset{A}{\|}$ is used to allow $P$ and $Q$ to run in parallel and synchronise on events in a set of events $A$. This would be written as $P \underset{A}{\|} Q$. If $P$ or $Q$ were to perform any events that are not in $A$ then they can do so independently without the need for any synchronisation. A process $P$ could be restricted on certain events $A$, expressed as $P \underset{A}{\|} STOP$ which means $P$ is not able to perform any events in $A$. The *interleaving* operator $|||$ is used to allow $P$ and $Q$ to run in parallel but with no interaction with each other. This is written as $P|||Q$. For a larger number of processes, an indexed form of the interleaving operator can be used. For a finite indexing set $I$ and process $P_i$ defined for each $i \in I$, $|||_{i \in I} P_i$ denotes the interleaving of all processes $P_i$. For the purpose of communication, a process may have channels on which it accepts inputs or produces output. The expression $c!v \rightarrow P$ describes a process that will output the value of $v$ on the channel $c$ and then behave as $P$. A process $P$ accepting an input $x$ on the channel $c$ is described as $c?x \rightarrow P(x)$ where the behaviour of $P$ after the input is described as $P(x)$, determined by the input.

#### *3.1.1 Trace Semantics*

The trace semantics in CSP allows us to capture the sequence of events performed by a communicating process as a trace and then use the trace to model the behaviour of the process. A trace is a sequence of events $tr$. A sequence $tr$ is a trace of a process $P$ if some execution of $P$ performs exactly that sequence of events. This is denoted as $tr \in traces(P)$, where $traces(P)$ is the set of all possible traces of $P$. An example of a trace could be $\langle a, b \rangle$ where event $a$ is performed followed by event $b$, whereas $\langle\rangle$ is an empty trace.

A concatenation of two traces $tr_1$ and $tr_2$ is written as $tr_1 \wedge tr_2$, which is the sequence of events in $tr_1$ followed by the sequence of events in $tr_2$. A trace $tr$ of the form $\langle a \rangle \wedge tr'$ expresses event $a$ followed by $tr'$, the remainder of the trace. A prefix $tr'$ of $tr$ is denoted $tr' \leqslant tr$. A non-contiguous subsequence of a trace is denoted by the symbol $\preccurlyeq$, for example, $\langle a,c,e \rangle \preccurlyeq \langle a,b,c,d,e \rangle$. The length $\#tr$ of a trace is the number of elements that it contains so that for example, $\#\langle a,b,d \rangle = 3$, whereas the set of events appearing in a trace $tr$ is denoted as $\sigma(tr)$. The projection operation, $tr \upharpoonright A$, is the maximal subsequence of $tr$, all of whose events are drawn from a set of events $A$. Another form of projection is on the set of channel names where $tr \Downarrow C$ provides the set of messages passed on a set of channels named $C$.

Trace semantics are used by Schneider (1996) to specify security properties for protocols as *trace specifications*. This is done by defining a predicate on traces and checking whether every trace of a process satisfies the *trace specification*. For a process $P$ and a predicate $S$, $P$ satisfies $S$ if $S(tr)$ holds for every trace $tr$ of $P$. More formally, $P$ **sat** $S \Leftrightarrow \forall\, tr \in \text{traces}(P) \bullet S(tr)$.

We use the above definition to specify a trace specification for a process, in terms of the occurrence of events in its traces. For some sets of events $R$ and $T$, the trace specification $R$ **precedes** $T$ is defined as

$$P \textbf{ sat } R \textbf{ precedes } T \Leftrightarrow \forall\, tr \in traces(P) \bullet (tr \upharpoonright R \neq \langle\rangle \Rightarrow tr \upharpoonright T \neq \langle\rangle)$$

where a process $P$ satisfies the predicate $R$ **precedes** $T$ if any occurrence of an event from $T$ is preceded by an occurrence of an event from $R$ in every trace $tr$ of $P$.

### *3.1.2 Schneider's model of the network*

Schneider (1998) models the protocol as a network where an arbitrary number of participants engage with each other along. The participants are modelled as CSP processes acting in parallel. An intruder process is also modelled alongside these participants, with capabilities as defined by Dolev and Yao (1983). These capabilities include blocking, replaying, spoofing and manipulating any messages that appear on any of the public channels in the network. In order to give the intruder complete control of the network, Schneider models the network such that all processes communicate with each other through the intruder, that is to say, the intruder becomes the medium. To express message transmission and reception for each process, Schneider introduces two channels, *send* and *receive*, which are public channels that all processes use to send and receive messages by. The events are structured as $send.i.j.m$ where a message $m$ is sent by source $i$ to destination $j$ on the channel *send* while $receive.j.i.m$ represents a message $m$ being received by $j$ from a source $i$ on the channel *receive*.

We consider a set of users $\mathcal{U}$ to represent all the participants that use the network and *Intruder* to denote the intruder process. For each participant $i \in \mathcal{U}$, a CSP process $USER_i$ represents the behaviour of the participant. We specify the complete network **NET** as

$$\textbf{NET} = (|||_{i \in \mathcal{U}} USER_i) \underset{(\text{send, receive})}{\|} Intruder$$

where all participants in $\mathcal{U}$ are forced to synchronise with *Intruder* on *send* and *receive* channels. In order to model the capabilities of the intruder according to the Dolev and Yao (1983) model, Schneider (1998) introduces a generates '$\vdash$' relation to characterise what messages may be generated from a given set of messages. The rules that define this relation are as follows, where $S$ is some set of messages, $m$ is a message and $k$ is some key

- $m \in S$ then $S \vdash m$
- $S \vdash m$ and $S \subseteq S'$ then $S' \vdash m$
- $S \vdash m_i$ for each $m_i \in S'$ and $S' \vdash m$ then $S \vdash m$
- $S \vdash m \wedge S \vdash k \Rightarrow S \vdash \{m\}_k$
- $S \vdash \{m\}_k \wedge S \vdash k \Leftrightarrow S \vdash m$
- $S \vdash m_1 .m_2 \Leftrightarrow S \vdash m_1 \wedge S \vdash m_2$
- $S \vdash m_1 \wedge S \vdash m_2 \Leftrightarrow S \vdash m_1 .m_2$

The relation can be extended to simulate further properties of cryptography or message extraction. We use this relation to specify a recursive definition of *Intruder* as follows

$$
\begin{array}{l}
Intruder(S) = \\
\quad send.i.j.m \rightarrow Intruder(S \cup \{m\}) \\
\quad \Box \\
\quad \Box_{\,i,j \in \mathcal{U}, S \vdash m} \; receive.i.j.m \rightarrow Intruder(S)
\end{array}
$$

The *Intruder* process is parameterised by a set of messages $S$ that denotes the set of messages in the possession of the intruder. The process is defined such that it has a choice: the first branch models the transmission of a message $m$, from a participant $i$ to participant $j$ on the channel send, after which the process behaves like the intruder with that additional message $m$. The second branch allows the intruder to send any message $m$ to any participant $i$ pretending to be some participant $j$, generated under $\vdash$ from $S$, after which the process remains with the same knowledge. The above definition of *Intruder* allows us to achieve two things, firstly, model the behaviour of an intruder in precise terms, such that it may (or may not) wish to block, spoof or manipulate some (or all) messages, and, secondly, allow the intruder to possess any initial public knowledge about the network such as participant identities and their respective public keys. Schneider (1998) denotes such a set of initial knowledge, *Initial Knowledge* ***IK***, and specifies *Intruder* such that *Intruder*(***IK***)

### *3.2 Modelling the Woo-Lam network*

We now model the Woo-Lam protocol in CSP as a network and specify the authentication property for this network as a trace specification. We then describe the proof strategy to verify this network for the given trace specification.

While modelling the different processes of a protocol, Schneider (1996) takes advantage of the extensibility of CSP to introduce additional control events known as signals. These signal events are then used in trace specifications to express the authentication goals of a protocol. We model the three participant roles in the Woo-Lam protocol in CSP below

$$
\begin{array}{llll}
User_A = \Box_b & send.A.b.A \rightarrow & User_B(n_B) = & receive.B.a.a \rightarrow \\
 & receive.A.b.n \rightarrow & & send.B.a.n_B \rightarrow \\
 & Running.A.b.n \rightarrow & & receive.B.a.\{n_B\}_{Kas} \rightarrow \\
 & send.A.b.\{n\}_{KAS} \rightarrow Stop & & send.B.s.\{a,\{n_B\}_{Kas}\}_{KBS} \rightarrow \\
 & & & receive.B.s.\{n_B\}_{KBS} \rightarrow \\
Server = & receive.S.b.\{a,\{n\}_{Kas}\}_{Kbs} \rightarrow & & Commit.B.a.n_B \rightarrow Stop \\
 & send.S.b.\{n\}_{Kbs} \rightarrow Stop & &
\end{array}
$$

In the model above, we specify a *Running.A.b.n* signal and introduce it in *A*'s run, indicating that *A* is aware of its involvement in a run with *b* and the nonce *n* being used as part of this run. We specify a corresponding $Commit.B.a.n_B$ signal on *B*'s behalf indicating that *B* has completed the protocol run and authenticated *a* using nonce $n_B$. The $Commit.B.a.n_B$ signal is placed at the end of *B*'s run as it is only when *B* receives the final message from *S* it can be assured of *A*'s involvement in the run. The entire network is composed of the above processes along with an Intruder process that is assumed to be in complete control of the network. In order to model it as such, we specify our **NET** as where $User_A$, $User_B$ and *Server* all communicate with each other only through an *Intruder* process and consider a specific run of the protocol between $User_A$ and $User_B$ using the nonce $N_B$.

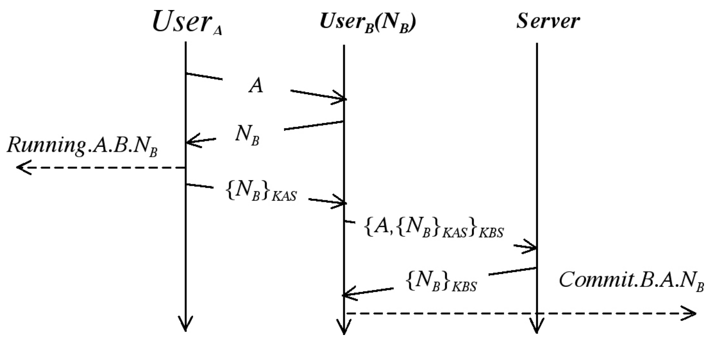

**Figure 3: A specific run of the Woo-Lam protocol involving *A* and *B* using nonce $N_b$**

$$\mathbf{NET} = (User_A \mid\mid\mid User_B(N_B) \mid\mid\mid Server) \mid\mid Intruder(\boldsymbol{IK})$$

We show this specific run of the protocol with appropriate signals in Figure 3 above.

## 3.3 Proof strategy

We now specify a trace specification that expresses the authentication property needed to be satisfied by **NET**. We use the signal events for the particular run shown in Figure 3. *B*'s authentication of *A* is now expressed as whenever $Commit.B.A.N_B$ appears in a trace of **NET** the corresponding $Running.A.B.N_B$ appears beforehand. This is formalised as

$$\mathbf{NET\ sat}\ Running.A.B.N_B\ \mathbf{precedes}\ Commit.B.A.N_B$$

In other words, if $Commit.B.A.N_B$ occurs then $Running.A.B.N_B$ precedes it. If **NET** could be proved to satisfy this specification, then the protocol is proved correct for the property of authentication. Once an authentication property is specified as a trace specification, Schneider (1998) adopts a simple proof strategy to verify **NET** against it: if the signal event $Running.A.B.N_B$ is prevented from occurring in **NET**, then the following signal event $Commit.B.A.N_B$ is not possible in **NET**. More formally $Commit.B.A.N_B$ should not appear in any trace *tr* of $\mathbf{NET} \underset{Running.A.B.NB}{\parallel} Stop$

$$\mathbf{NET} \underset{Running.A.B.NB}{\parallel} Stop\ \mathbf{sat}\ tr \upharpoonright Commit.B.A.N_B = \langle\rangle$$

If the above property is satisfied then $Commit.B.A.N_B$ can never occur in **NET** restricted on the $Running.A.B.N_B$ event and therefore authentication is verified.

## 4. ANALYSING THE WOO-LAM

In the previous section, we have built a CSP system along with a trace specification that the system needs to satisfy. To verify the system for such a specification, Schneider (1998) introduces the notion of a rank function. We describe the idea along with the central rank function theorem (Schneider, 1998) in Section 4.1. In Section 4.2, we construct a rank function for the Woo-Lam protocol and evaluate the different conditions provided in the theorem to judge the correctness of the protocol.

### 4.1 Rank functions

Consider the set of participant identities on the network to be $\mathcal{U}$, the set of nonces used by the participants in protocol runs as $\mathcal{N}$ and a set of encryption keys used as $\mathcal{K}$. The set of all such atoms is $\mathcal{A}$, where the atoms are defined as $\mathcal{A} = \mathcal{U} \cup \mathcal{N} \cup \mathcal{K}$. We consider a message space $\mathcal{M}$ to contain all the messages and signals that may appear during a protocol's execution, such that $m \in \mathcal{A} \Rightarrow m \in \mathcal{M}$. Schneider (1998) defines a rank function $\rho$ to map events and messages to integers $\rho \colon \mathcal{M} \to \mathbb{Z}$. The message space is then divided into two parts where

$$\mathcal{M}_{p-} = \{m \in \mathcal{M} \mid \rho(m) \leqslant 0\} \qquad \mathcal{M}_{p+} = \{m \in \mathcal{M} \mid \rho(m) > 0\}$$

The purpose of this partition of the message space is to characterise those messages that the intruder might get hold of without compromising the protocol – assigned a positive rank – and those messages that the enemy should never get hold of – assigned a non-positive rank. It is desirable for a process never to transmit a message of non-positive rank. For a certain process *P* to maintain positive rank, it is understood that it will never transmit a message with a non-positive rank unless it has previously received a message with a non-positive rank. More formally, for a process *P*,

$$P \ \textbf{maintains}\ \rho \Leftrightarrow \forall\, tr \in traces(P) \bullet \rho(tr \Downarrow receive) > 0 \Rightarrow \rho(tr \Downarrow send) > 0$$

In other words *P* will never transmit any message *m* of $\rho(m) \leqslant 0$ unless it has received some $m'$ of $\rho(m') \leqslant 0$ previously, with respect to some rank function $\rho$. It is not important who the message is received from or is sent to.

Schneider (1998) presents a general-purpose rank function theorem that ensures the messages that an *Intruder* gets hold of do not compromise the security property that the protocol provides. Considering that the communication channels are public – under the control of the *Intruder* – any message that flows through them should be of positive rank. If a message with non-positive rank flows through the channel then the intended secrecy of the message is compromised. A protocol is verified to be correct with regard to its security property, if it allows messages of only positive rank to be communicated through the channels.

RANK FUNCTION THEOREM

If, for sets *R* and *T*, there is a rank function $\rho \colon \mathcal{M} \to \mathbb{Z}$ satisfying

**R1)** $\forall\, m \in \boldsymbol{IK} \bullet \rho(m) > 0$

**R2)** $\forall\, S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet ((\forall m' \in \boldsymbol{S} \bullet \rho(m') > 0) \wedge \boldsymbol{S} \vdash m) \Rightarrow \rho(m) > 0$

**R3)** $\forall\, t \in T \bullet \rho(t) \leqslant 0$

**R4)** $\forall\, i \in \mathcal{U} \bullet User_i \underset{R}{\|} Stop$ **maintains** $\rho$

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

The theorem, the proof of which is available in (Schneider, 1998), states that if the rank function, and therefore the underlying **NET**, satisfies the four properties, then no messages of non-positive rank can circulate in **NET** $\underset{R}{\|}$ *Stop*. In particular, an intruder should not be able to generate any illegal messages from the messages it knows at the beginning of the protocol from the set ***IK***, nor from the messages it sees during the protocol execution, denoted by a set ***S***. Also, honest participants should not be able to generate any illegal messages unless they are sent one, that is, every honest process maintains $\rho$ while being restricted on $R$. The actual verification of the theorem conditions is performed manually for every rank function constructed for a protocol. Verifying different specifications may require different rank functions to be constructed for the same protocol. This is due to the different events that **NET** may be restricted on for different specifications – sets $R$ and $T$ will contain different events for different cases.

Note that although the theorem provides an assurance for a protocol once a rank function is constructed, there is no guarantee that a suitable rank function (that satisfies the theorem conditions) exists for every case. There may be cases where it is not possible to concur on a rank of a message or where an attack on the protocol is found. In most cases, however, the failure of a rank function to satisfy the conditions of the theorem signifies a flaw in the protocol and provides an insight into the workings of a protocol, which can lead to the discovery of an attack.

## 4.2 Constructing the rank function

We identify the ranks on the message space for our **NET** and construct the rank function shown in Figure 4 below. The rank function we have constructed assigns all user identities in the set $\mathcal{U}$ a positive rank. The identity of all users is assumed to be known to the *Intruder* and therefore could be impersonated by the intruder. All the nonces in the set $\mathcal{N}$ , including $N_B$, are assigned a positive rank. $B$ sends out $N_B$ in cleartext and therefore an *Intruder* can get hold of the nonce without further ado. The two shared keys used in the protocol, $K_{AS}$ and $K_{BS}$, are both assigned a non-positive rank as they are supposed to be private to $A$ and $B$. As the **NET** is restricted on the event $Running.A.B.N_B$, the three messages (see Figure 3) that follow this event, $\{NB\}_{KAS}$, $\{A,\{N_B\}_{KAS}\}_{KBS}$ and $\{N_B\}_{KBS}$, should not appear in the restricted **NET** either. We assign these three messages a non-positive rank along with signal event $Commit.B.A.N_B$, which logically follows these three messages.

$$\rho(\mathcal{U}) = 1 \text{ (including } A,\ B \text{ and } S) \qquad \rho(\mathcal{N}) = 1 \text{ (including } N_B)$$

$$\rho(\mathcal{K}) = \begin{cases} 0 & \text{if } k = K_{AS} \\ & \text{or } k = K_{BS} \\ 1 & \text{otherwise} \end{cases} \qquad \rho(\{m\}_k) = \begin{cases} 0 & \text{if } \{m\}_k = \{N_B\}_{KAS} \\ & \text{or } \{m\}_k = \{A,\{N_B\}_{KAS}\}_{KBS} \\ & \text{or } \{m\}_k = \{N_B\}_{KBS} \\ 1 & \text{otherwise} \end{cases}$$

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

$$\rho(Running.A.B.N_B) = 1 \qquad \rho(Commit.B.A.N_B) = 0$$

**Figure 4: A rank function for the Woo-Lam protocol**

Recall that the rank function theorem is defined in terms of general sets $R$ and $T$. For our analysis, we assign sets $R$ and $T$ to $Running.A.B.N_B$ and $Commit.B.A.N_B$ respectively

$$R = \{Running.A.B.N_B\} \qquad T = \{Commit.B.A.N_B\}$$

This corresponds to the proof strategy described in Section 3.3, where we need to check for the occurrence of $Commit.B.A.N_B$ in **NET** restricted on $Running.A.B.N_B$. We now consider each of the conditions of the rank function theorem and check whether our rank function satisfies them.

**R1)** $\forall\, m \in \boldsymbol{IK} \bullet \rho(m) > 0$

The set ***IK*** contains all the agent identities and a key $K_{IS}$ shared between $I$ and $S$. There is nothing in this set that is of non-positive rank. The condition is deemed satisfied.

**R2)** $\forall\, S \subseteq \mathcal{M}, m \in \mathcal{M} \bullet ((\forall m' \in S \bullet \rho(m') > 0) \wedge S \vdash m) \Rightarrow \rho(m) > 0$

This conditions checks whether a message of non-positive rank can be generated under the '$\vdash$' relation from a set of messages of positive rank. None of the messages identified as of positive rank, shown in Figure 4, let the Intruder generate any messages that are of non-positive rank. The three messages of non-positive rank, $\{N_B\}_{KAS}$, $\{A,\{N_B\}_{KAS}\}_{KBS}$ and $\{N_B\}_{KBS}$, are encrypted under keys $K_{AS}$ and $K_{BS}$ both of which are of non-positive rank. This prevents the Intruder from generating these messages as the Intruder has no way of acquiring these two keys. The condition is deemed satisfied.

**R3)** $\forall\, t \in T \bullet \rho(t) \leqslant 0$

This condition requires none of the events in $T$ to be of positive rank. The only event in set $T$ is the signal event $Commit.B.A.N_B$ of non-positive rank. This condition is deemed satisfied.

**R4)** $\forall\, i \in \mathcal{U} \bullet User_i \underset{R}{\|} Stop$ **sat** maintain positive $\rho$

For this condition to be satisfied every process in the **NET** needs to maintain positive $\rho$ while being restricted on the events in set $R$, where $R = \{Running.A.B.N_B\}$. We consider processes $User_A$, $User_B$ and $Server$, restrict them on $Running.A.B.N_B$ and check whether they maintain positive $\rho$. Since only $User_A$ can perform $Running.A.B.N_B$, the other two processes remain unaffected. The restriction on $User_A$ simplifies to

$$\begin{aligned} User_A \underset{Running.A.B.N_B}{||} Stop \;=\; \Box_b\;\; & send.A.b.A \rightarrow \\ & receive.A.b.n \rightarrow \\ & \text{if } b = B \wedge n = N_B \;\; \text{then} \;\; Stop \\ & \text{else } Running.A.b.n \rightarrow \\ & send.A.b.\{n\}_{KAS} \rightarrow Stop \end{aligned}$$

In the choice operator $\Box_b$, $b$ indicates the other participants that $User_A$ may communicate with. If participant $b = B$ and the nonce $n = N_B$ then we instruct $User_A$ to $Stop$. Any other participant instead of $B$ or even a different nonce then $N_B$ would allow $User_A$ to continue as normal.

Upon inspection, we observe that $User_A \underset{Running.A.B.N_B}{||} Stop$ fails to maintain positive $\rho$. In terms of protocol runs, consider a run where $A$ initiates the protocol with a participant other than $B$ (and $I$ intercepts) as shown in Figure 5 below

$$\begin{array}{lll} (1) & A \rightarrow I(C): & A \\ (2) & I(C) \rightarrow A: & N_B \\ (3) & A \rightarrow I(C): & \{N_B\}_{KAS} \end{array}$$

**Figure 5. A possible run of the protocol**

In this case, the process $User_A \underset{Running.A.B.N_B}{||} Stop$ behaves as follows

$send.A.C.A \rightarrow$
$receive.A.C.N_B \rightarrow$
if $b = B \wedge n_b = N_B$
    then $Stop$
    else $Running.A.C.N_B \rightarrow$
        $send.A.C.\{N_B\}_{KAS} \rightarrow Stop$

$A$ outputs the message $\{N_B\}_{KAS}$ while communicating with another participant $C$. The message $\{N_B\}_{KAS}$ is of non-positive rank as shown in Figure 4. This shows that $A$ does not maintain positive $\rho$ despite the restriction on it and the protocol, therefore, fails to meet the trace specification presented in Section 3.3. The theorem has been successful in finding a flaw in the Woo-Lam protocol that we demonstrated earlier in Section 2.

## 5. CONCLUSION

We have applied Schneider's rank function theorem to the Woo-Lam protocol to expose any flaws in the design and have managed to identify the attacks discussed earlier successfully. While this approach provides a formally meticulous analysis, the process of identifying a rank function is non-trivial. Once a rank function is constructed, however, it gives confidence in the soundness of the protocol design.

We recommend this approach for thorough investigation of similar protocols. The process of finding a rank function requires intuitive understanding of such protocols and focuses attention on relevant design aspects of these protocols.

## REFERENCES

DOLEV, D. and YAO, A.C. (1983): On the security of public key protocols. *IEEE Transactions on Information Theory* 29(2):198–208.

HOARE, C.A.R. (1985): Communicating sequential processes. Prentice-Hall.

GONG, L., NEEDHAM, R. and YAHALOM, R. (1990): Reasoning about belief in cryptographic protocols. *Proc. IEEE Symposium on Research in Security and Privacy*, Los Alamitos, California, USA, 234-248. IEEE Computer Society Press.

GORDON, A.D. and JEFFREY, A. (2001): Authenticity by typing for security protocols. *Proc. 14th IEEE Computer Security Foundations Workshop*, 145–159, IEEE Computer Society Press.

LOWE, G. (1996): Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Proc. of TACAS,* LNCS 1055:147–166, Springer-Verlag.

MEADOWS, C. (1996): The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131.

MITCHELL, J.C., MITCHELL, M. and STERN, U. (1997): Automated analysis of cryptographic protocols using Murj. Proc. *IEEE Symposium on Security and Privacy*, 141–151. IEEE Computer Society Press.

RYAN, P., SCHNEIDER, S., GOLDSMITH, M., LOWE, G. and ROSCOE, B. (2001): Modelling and analysis of security protocols. Addison-Wesley.

SCHNEIDER, S. (1996): Security properties and CSP. Proc. *IEEE Symposium Research in Security and Privacy*, Oakland, California, USA. IEEE Computer Society Press.

SCHNEIDER, S. (1998): Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering* 24(9):741–758.

SYVERSON, P.F. and van OORSCHOT, P.C. (1994): On unifying some cryptographic protocol logics. *Proc. IEEE Symposium on Research in Security and Privacy,* 14–28. IEEE Computer Society Press.

THAYER FÁBREGA, F.J. HERZOG, J.C. and GUTTMAN, J.D. (1998): Strand spaces: Why is a security protocol correct? *Proc. IEEE Symposium Research in Security and Privacy*, 24–34. IEEE Computer Society Press.

WOO, T.Y.C. and LAM, S.S. (1992): Authentication for distributed systems. *Computer* 25(1):39–52.

WOO, T.Y.C. and LAM, S.S. (1994): A lesson on Authenticated Protocol design. *Operating Systems Review* 28(3):24–37.

## BIOGRAPHICAL NOTES

Siraj Shaikh

*Siraj Shaikh is a Lecturer in Computer Networking with the Department of Multimedia and Computing, at the University of Gloucestershire, Cheltenham Spa, UK. He received a BSc (Honours) in Computing from Northumbria University, a MSc in Computer Networking from Middlesex University and has been a PhD student at University of Gloucestershire since 2003. His PhD research investigates the formal specification and analysis of authentication protocols using the CSP framework. His other research interests include design and performance analysis of security protocols, particularly IPSec,* *SSH and Kerberos, along with security for wireless networks and information security education.*

*Vicky Bush lectures at the University of Gloucestershire. She has also taught at the universities of Manchester and West of England. Her research interests centre round the use of formal methods applied in various contexts, such as demonstrating the equivalence of algorithms designed to run on machines with different numbers of processors, specifying computer programs and the analysis of security protocols. She is also interested in the possibilities of visualization to aid the teaching of programming.*

Vicky Bush