Verifying Security Protocols: an application of CSP

Steve Schneider and Rob Delicata

Department of Computing, University of Surrey

Abstract. The field of protocol analysis is one area in which CSP has proven particularly successful, and several techniques have been proposed that use CSP to reason about security properties such as confidentiality and authentication. In this paper we describe one such approach, based on theorem-proving, that uses the idea of a rank function to establish the correctness of protocols. This description is motivated by the consideration of a simple, but flawed, authentication protocol. We show how a rank function analysis can be used to locate this flaw and prove that a modified version of the protocol is correct.

1 Introduction

In their seminal paper [NS78], Needham and Schroeder proposed a way of using cryptographic mechanisms, such as public-key and shared-key encryption, in order to establish authentication guarantees across networks. Such mechanisms typically involve an exchange of messages between participants, and are known as authentication protocols. Participants carry out cryptographic operations particular to them (such as encrypting with a specific secret key) which are intended to provide guarantees as to their identity. Such protocols are designed to provide authentication even in insecure environments, where other parties can potentially interfere with messages over the network in various ways. For example, messages can be overheard, copied, blocked, replayed, diverted, duplicated, and spoofed.

As a motivating and running example, we will consider the following exchange of messages, which appears as a simple (flawed) authentication protocol in the Handbook of Applied Cryptography [MVV96]:

$$A \to B : n_A$$
$$B \to A : \{n_A, n_B\}_{K_{AB}}$$
$$A \to B : n_B$$

The aim of this protocol is for each of the participants to authenticate themselves to the other. In other words, each participant should know, by the end of the protocol, the identity of the other participant.

This protocol involves two participants, A and B, who share a symmetric cryptographic key K_{AB} (which can also be written K_{BA}) which is used by each of A and B to encrypt and decrypt messages to and from the other. The protocol

relies on the assumption that no party other than A or B knows this key. The protocol begins with A, acting as *initiator*, who invents a new random number (or *nonce*), n_A, and transmits it to B. This nonce is sent unencrypted, so any other agent could potentially eavesdrop and learn its value, or spoof some arbitrary nonce n_I to B as if it came from A. (As a result, B's receipt of the nonce does not carry any assurance that it originated from A.)

On receipt of the nonce n_A, B, as *responder*, performs a cryptographic operation that no other party can perform: by encrypting the nonce with K_{AB}. This message is then sent to A, who decrypts it using K_{AB}. If this decryption contains the nonce n_A then this provides a guarantee that n_A must have been received and encrypted by B, since B is the only other party that knows K_{AB}. This results in the authentication of B to A: A knows that she has been communicating with B, and not some malicious party pretending to be B. In order to achieve authentication in the other direction (A to B), B also includes a freshly generated nonce n_B in the encryption of the second message. A is able to decrypt this nonce and send it back, unencrypted, to B. On receipt of n_B, B has an assurance that it was A who received and returned the nonce, and hence was the other party involved in the protocol run.

The assurances are obtained by virtue of the fact that K_{AB} is known only to A and B, and hence evidence of its use provides evidence that A or B were involved in carrying out the encryption or decryption. Indeed, if A and B are only ever involved in one protocol run, then the protocol does provide the authentication required of it: A cannot reach the end of the run unless B is involved; and B cannot reach the end of the run unless A is involved.

However, agents can generally be involved in multiple protocol runs, possibly simultaneously, potentially with a variety of other participants, and in each case may assume the role of either initiator or responder (or, indeed, both). Under such circumstances, the protocol is susceptible to an *attack*: an exchange of messages after which one agent has reached a state where authentication appears to have been established, and yet where the party supposedly authenticated has not in fact been involved.

The attack (also given in [MVV96]) involves two runs, where A assumes the role of initiator in one run and responder in the other. In both runs A intends B to be the other party, but in fact the messages are being processed by some other agent $E(B)$, who A considers to be B. The runs, labelled α and β, are interspersed as follows:

$$
\begin{array}{ll}
\alpha: & A \rightarrow E(B) : n_A \\
\beta: & E(B) \rightarrow A : n_A \\
\beta: & A \rightarrow E(B) : \{n_A, n'_A\}_{K_{AB}} \\
\alpha: & E(B) \rightarrow A : \{n_A, n_A\}_{K_{AB}} \\
\alpha: & A \rightarrow E(B) : n_A
\end{array}
$$

The steps of the attack are as follows:

1. A initiates a run using nonce n_A, apparently with B; but the nonce is intercepted by $E(B)$.

2. $E(B)$ initiates a separate run with A (who thus takes the role of responder), apparently with B, using the same nonce n_A.
3. On receipt of the nonce n_A, A invents a responder's nonce n'_A and then returns it, together with n_A, encrypted under K_{AB}.
4. $E(B)$ intercepts this message and sends back exactly the same message to A as the response to the original nonce challenge n_A of the first run. A accepts the nonce n'_A as the nonce n_B provided by B.
5. A responds with the nonce n_A just received.

After this exchange of messages, A has reached the end of the protocol run, apparently with B, and hence the protocol is intended to provide an assurance that B was indeed the other participant. However, B has not been involved at all. Hence the protocol does not provide the assurances required of it.

Having identified the attack, it is possible to suggest corrections which will prevent it. In this example the attack was possible because the second message is symmetric in terms of initiator and responder, and contains no information about which participant created it. This allowed a situation in which A generated such a message and was later persuaded to accept it as if it came from the other party. Introducing the name of the participant who encrypted the message would prevent the attack above. This results in the revised protocol:

$$A \to B : n_A$$
$$B \to A : \{B, n_A, n_B\}_{K_{AB}}$$
$$A \to B : n_B$$

However, can we be confident that no other attacks are possible on the corrected protocol?

In order to obtain such confidence, it is necessary first to clarify several issues around the protocol:

- What kind of environment is the protocol designed for? In other words, what are the kinds of attacks that the protocol is designed to be resistant to? For example, on a broadcast network an attacker may be able to overhear and spoof messages, but be unable to block them.
- What level of authentication is the protocol designed to provide? For example, is it simply intended to establish that the authenticated agent is present (e.g. that a server is up), or that the authenticated agent knows who he is communicating with.
- Are the other participants assumed to be honest (i.e. attacks can only originate from outside the collection of protocol participants) or can they be dishonest?
- Can participants run arbitrarily many concurrent protocol sessions, or are there restrictions?

This kind of information should be included with any protocol description: the correctness of a protocol consists not only in the sequence of messages it describes, but also the environment it is designed for.

There have been a variety of approaches proposed for analysing and verifying security protocols [Mea92,Mil95,THG99,Low98,Pau98,CDL+99b,AG98,DFG00]. Such approaches do indeed incorporate such information into the models that they describe and analyse.

This paper is concerned with the application of CSP [Hoa85,Ros97,Sch99] to the verification of security protocols, and in particular with the *rank function* approach. There has already been significant experience of the application of CSP to communications protocols, and that experience provides a framework for the application of CSP to authentication protocols. Broadly speaking, there are three components of the approach:

- The requirements on the protocol are expressed either as a CSP process (to be refined by the implementation), or as **sat** specifications on the observable behaviours of the overall system: traces, failures, divergences. Such specifications describe the appropriate behaviour, and provide a basis for judging whether protocols exhibit correct behaviour or not.
- A protocol, although initially described in terms of message exchanges, is captured in CSP in terms of the behaviour of each participating agent, leading to an *agent-oriented* rather than a *message-oriented* viewpoint. Each participant in the protocol is described as a CSP process. This shift in viewpoint, away from message transmission and reception, and towards the individual agents considered in terms of their interactions with the rest of the system, is a key feature in the success of the approach when applied to authentication protocols, since it naturally focuses on where attacks might come from and hence how they should be prevented.
- Finally, the environment is also described as a CSP process. In communications protocols, this is generally an unreliable medium which might lose, reorder, or duplicate messages. The particular behaviour captured within the medium is precisely that behaviour that the protocol has been designed to overcome. For example, the traditional alternating bit protocol (see e.g. [Sch99]) is designed to provide reliable communication over a medium which can lose messages, and so the analysis of the protocol includes a CSP description of exactly such a medium (which non-deterministically either reliably communicates a message or else loses it). In the case of security protocols, we need to include the capabilities of possible attackers.

When all three components are in place: specification, environment, and protocol description, then the mature tools and techniques that CSP has to offer can be brought to bear on particular protocols, and whether or not they meet a particular specification.

This paper assumes a knowledge of CSP and, in particular, the notations of [Sch99].

The next section elaborates a theory for verifying authentication protocols based on this approach.

2 Verifying authentication protocols in CSP

Any authentication protocol is intended to run over a network which can be subject to particular kinds of attack. We take the approach of considering an attacker (synonymous terms include 'intruder', 'enemy', 'spy', and 'penetrator') in terms of *capabilities*, such as being able to intercept messages on the network, create new messages for passing on the network, redirecting messages, and so on. We will assume a single attacker, though in fact the attacker we will describe has the ability to behave as a collection of attackers.

2.1 The attacker

Since the aim is to prove that protocols are correct, we take a pessimistic point of view and assume an attacker with maximal capabilities. In the worst case, the attacker has complete control over all the messages in the network. If a protocol is secure even in such an environment, then it will be secure in any weaker, perhaps more realistic, environment. The only capabilities the attacker should not have are the ability to encrypt or decrypt messages without the appropriate keys. As a consequence, we assume there is enough redundancy in the cryptosystem so that each ciphertext can be produced in exactly one way. This restriction has become known as the *perfect encryption* assumption [PQ00].

We use the Dolev-Yao model, first proposed in [DY83], in which the attacker has complete control of the network and, to all intents and purposes, replaces the network. Thus, messages that are sent are automatically intercepted and held by the attacker. Messages that are received from the network must have come from the attacker. This simple model allows for the kinds of attacker behaviour described earlier. It allows for messages to be delivered normally, since one action the attacker can take is to deliver messages to the intended recipient unaltered. However, it also allows for messages to be misdirected, blocked, spoofed, reordered, and duplicated. Furthermore, the attacker can himself be in possession of some agent identities (names and associated cryptographic keys) and so appear to other agents on the network as a potential communication partner. In this way,

dishonest agents are encapsulated within the model. Any message that can be generated by the attacker, from what he has already observed and what he originally knows, can potentially be delivered to any other agent on the network, as if it came from any other agent.

The details of the CSP description of the attacker model will reflect the kind of environment the protocol is designed for. For example, if the protocol is intended to operate between two known honest participants, then the attacker might not itself control any agent identities. Furthermore, the precise cryptographic capabilities of the attacker will also be incorporated into the model, and this might be protocol-specific.

The overall network consists of a number of users connected to the communications medium, which is under the control of the attacker. The users will be modelled as CSP processes $USER_i$, where i is the agent's name. We will use a

$$\textsc{Pairing}\ \frac{S \vdash m_1 \qquad S \vdash m_2}{S \vdash m_1 \cdot m_2} \qquad \textsc{Unpairing}\ \frac{S \vdash m_1 \cdot m_2}{S \vdash m_1 \qquad S \vdash m_2}$$

$$\textsc{Member}\ \frac{}{S \vdash m}\ [m \in S] \qquad \textsc{Subset}\ \frac{S' \vdash m}{S \vdash m}\ [S' \subseteq S] \qquad \textsc{Transitive closure}\ \frac{\forall s' \in S' . S \vdash s' \qquad S' \vdash m}{S \vdash m}$$

$$\textsc{Encryption}\ \frac{S \vdash m \qquad S \vdash k}{S \vdash \{m\}_k} \qquad \textsc{Decryption}\ \frac{S \vdash \{m\}_k \qquad S \vdash k}{S \vdash m}$$

Fig. 1. Attacker inference rules

channel $trans.i$ for agent i to transmit messages intended for other users onto the network. An event $trans.i.j.m$ will correspond to agent i sending message m, intended for agent j. We will use a channel $rec.j$ for agent j to receive messages from the network. An event $rec.j.i.m$ corresponds to agent j receiving message m from the network, apparently from i. All message exchanges between protocol participants will use channels and events of this form.

We also need to define the kind of messages that can be passed around the network. This will depend on the protocol under analysis, since different protocols use different message constructions. For the example protocol introduced earlier, we will have three pairwise disjoint sets, $USER$, $NONCE$, KEY, which give the agent identities, nonces, and keys respectively. Furthermore, for each pair of distinct users i and j, there will be a shared key $k_{ij} = k_{ji}$ such that different pairs of agents have different shared keys. We will use the following space of messages, defined using BNF as the set $MESSAGE$:

$M_1, M_2 ::=$		messages
I	$(\in USER)$	agent identities
N	$(\in NONCE)$	nonces
K	$(\in KEY)$	keys
$M_1.M_2$		concatenation of messages
$\{M\}_K$		encryption of message M by key K

For this space of messages, we can define the attacker's capabilities in terms of the generation of new messages from those already possessed. We introduce a 'generates' relation $\vdash$, which relates a set of messages S to a message m that can be generated from S. It is defined to be the least relation closed under the inference rules of Figure 2.1. We are now in a position to describe the CSP model of the Dolev-Yao style attacker. It is given as the process $ENEMY$, defined as follows:

$$ENEMY(S) = trans?i?j?m \rightarrow ENEMY(S \cup \{m\})$$
$$\Box$$

$$\Box_{\substack{i \in USER \\ j \in USER \\ m \mid S \vdash m}} rec!i!j!m \rightarrow ENEMY(S)$$

The process $ENEMY(S)$ describes the possibilities available to an attacker in possession of the set of messages S. The first branch of the choice models the situation that a new message m can always be transmitted from any user to any other user, and this will be intercepted and added to the set of messages possessed by the attacker. The second branch of the choice describes that the attacker can provide any message m that can be generated from S to any user i, as if it came from any other user j. In this case the attacker's store of known messages S does not change.

The enemy will have some initial knowledge, including some nonces he can use, agents' identities, and cryptographic keys of agents that he controls. If the initial knowledge is given as the set IK, then $ENEMY$ — the environment that the protocol runs over — is given by

$$ENEMY = ENEMY(IK)$$

2.2 Specifying authentication

When two parties engage in a protocol run aimed at authenticating one to the other, the intention is that completion of the run by the authenticating party provides a guarantee that the other party had also participated in the run. Since specifications in CSP are defined in terms of events, we will introduce special signal events into the protocol runs at the points we wish to mark: completion of a protocol run, and participation in a run. The approach of introducing matching signals to specify authentication was introduced (not in the CSP context) by Woo and Lam [WL93]. These signals are introduced purely for the purposes of specification, to describe stages that protocol participants have reached, and they are used in the analysis and verification of the protocol. They are not events that the attacker can engage in.

In our example, we will introduce only two signals. Generally, others could be introduced depending on the authentication properties of interest.

Here we consider the property of the initiator authenticating the responder. This can be specified by introducing the following signals:

- $initdone.i.j.n$, which i performs after a protocol run as initiator involving j, and using n as the nonce.
- $respgo.j.i.n$, which j performs during a protocol run as responder apparently initiated by i with nonce n.

The set of all possible signals for this protocol and property is defined as follows:

$$\begin{aligned} SIGNAL = &\{initdone.i.j.n \mid i \in USER \wedge j \in USER \wedge n \in NONCE\} \\ &\cup \{respgo.i.j.n \mid i \in USER \wedge j \in USER \wedge n \in NONCE\} \end{aligned}$$

These signals will be inserted into the protocol runs. The intention is that an occurrence of the signal $initdone.A.B.n_A$ guarantees that (elsewhere in the

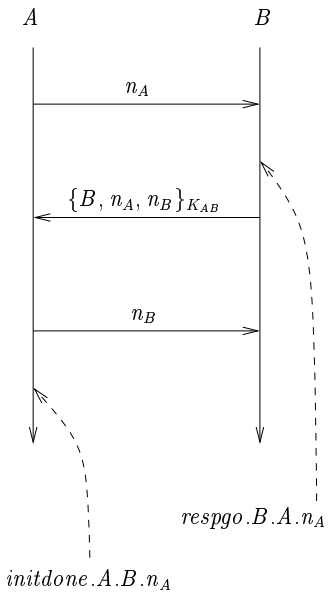

Fig. 2. Introducing matching signals

network, at B's location) the event $respgo.B.A.n_A$ has previously occurred, at least once. Thus the *respgo* signal must be inserted *before* the responder transmits his response to the first message, since it must be placed causally prior to the *initdone* message. The placing of the signals into the protocol is illustrated in Figure 2.

The inclusion of a specific nonce with the signal means that the agents must agree on the particular protocol run: A does not only authenticate B's presence, but also that B was engaged in the same protocol run.

The use of signals enables authentication to be expressed as a trace specification: that any occurrence of $initdone.A.B.n_A$ in any trace of the overall network must be preceded by some occurrence of $respgo.B.A.n_A$. This can be defined formally on traces, as:

$$respgo.B.A.n_A \text{ precedes } initdone.A.B.n_A$$

where

$$a \text{ precedes } b \mathrel{\widehat{=}} tr \upharpoonright a = \langle\rangle \Rightarrow tr \upharpoonright b = \langle\rangle$$

Observe that this specification allows arbitrarily many b events in response to a single a event. This has been termed *non-injective agreement* [Low97].

The inclusion of different information in the signals can give rise to different authentication requirements. For example, the removal of the nonce from the signals would allow interactions in which A's run could correspond to a different

run from B (i.e. one with a different nonce). However, there would still be a guarantee that B has been involved in some run apparently with A. An even weaker authentication would simply allow the signal $respgo.B$, not even requiring that B is engaged in a run apparently with A. This form of authentication might be appropriate if A simply requires some guarantee that B is alive. In practice different notions of authentication are appropriate to different situations, and the use of signals containing appropriate levels of detail allow these differences to be expressed. The various flavours of authentication are discussed in [Sch98,Low97].

2.3 Protocol participants

The protocol participants are also described as CSP processes. Here we will consider the modified version of the protocol where the responder's identity is included in the encryption of the second message. There are two possible roles in the protocol, and each of these will be described as a process.

An initiator run is parameterised by the identity of the initiating agent, the identity of the agent she wishes to authenticate, and the nonce used in the run. Thus we define $INIT_i(j,n)$ as a run of agent i using nonce n to authenticate j:

$$\begin{aligned} INIT_i(j,n) = {} & trans.i.j.n \rightarrow \\ & rec.j.i?\{j.n.y\}_{k_{ij}} \rightarrow \\ & trans.i.j.y \rightarrow \\ & initdone.i.j.n \rightarrow Stop \end{aligned}$$

Observe the use of pattern matching in the input of the second message: n, j, and k_{ij} are already fixed, and the input message must match these. However, any value for y can be accepted.

Similarly, $RESP_j(n')$ is a responder run for agent j, using nonce n' for the nonce that he generates. This is defined as follows:

$$\begin{aligned} RESP_j(n') = {} & rec.j?i?x \rightarrow \\ & respgo.j.i.x \rightarrow \\ & trans.j.i.\{j.x.n'\}_{k_{ij}} \rightarrow \\ & rec.j.i.n' \rightarrow Stop \end{aligned}$$

Observe that $RESP_j(n')$ is ready to run the protocol with anyone who requests.

In the most general case, an agent will be prepared to participate in any number of concurrent protocol runs in either role, which is expressible as an interleaving of runs. Our model must incorporate the fact that each run uses a different nonce, so we will use a collection of pairwise disjoint sets of nonces: N_j^I will be an infinite set of nonces that j can use on initiator runs; and N_j^R will be an infinite set of nonces that j can use on responder runs. A general agent is then given as:

$$USER_j = (|||_{n \in N_j^I} \Box_i INIT_j(i,n))$$

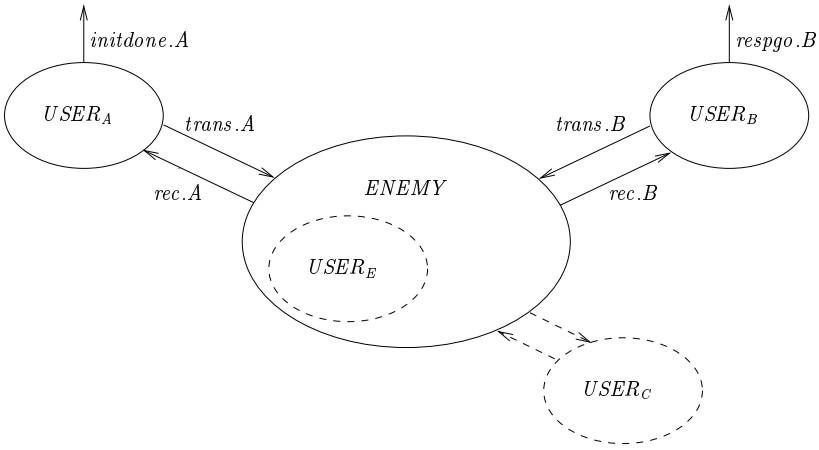

Fig. 3. The Dolev-Yao model in CSP

$$\begin{array}{l} ||| \\ (\Big|\Big|\Big|_{n \in N_j^R} RESP_j(n)) \end{array}$$

The resulting system is given by

$$SYSTEM = (\Big|\Big|\Big|_i USER_i) \,|[\{|\, trans, rec \,|\}]|\, ENEMY$$

This architecture is pictured in Figure 3.

To show that the protocol ensures that A authenticates B, we aim to establish that the following specification holds:

$$SYSTEM \textbf{ sat } respgo.B.A.n_A \textsf{ precedes } initdone.A.B.n_A$$

3 A theorem for verifying authentication

We will now introduce the *rank function* approach to verifying authentication protocols. In this approach we consider a restriction to the process $SYSTEM$ which prevents the occurrence of $respgo.B.A.n_A$, and then aim to establish that $initdone.A.B.n_A$ cannot occur. This approach is valid because

$$\begin{array}{l} SYSTEM \textbf{ sat } respgo.B.A.n_A \textsf{ precedes } initdone.A.B.n_A \\ \Leftrightarrow SYSTEM \,|[\, respgo.B.A.n_A \,]|\, Stop \textbf{ sat } tr \upharpoonright \{initdone.A.B.n_A\} = \langle\rangle \end{array}$$

We will associate a value, or *rank*, with each message that might occur in the restricted system, and aim to establish an invariant based on the message values: that only those with positive ranks can circulate in the restricted system.

We aim to define a rank function $\rho : MESSAGE \cup SIGNAL \to \mathbb{Z}$ with properties that enable us to do this.

Our required result will follow if we can establish the following properties for the rank function:

1. The attacker should not initially possess any messages of non-positive rank;
2. If the attacker only possesses messages of positive rank, then any messages he can generate should also be of positive rank;
3. The signal $initdone.A.B.n_A$ has non-positive rank;
4. Any agent, when restricted on $respgo.B.A.n_A$, does not introduce messages or signals of non-positive rank if it has not previously received any such messages.

The first two conditions between them ensure that the attacker cannot introduce any non-positive rank messages; and the fourth condition ensures that the protocol agents cannot do this either. Together these conditions ensure that no message or signal of non-positive rank can occur in the restricted system. Since the third condition requires that the signal we are concerned about should have non-positive rank, we can conclude that this signal indeed cannot occur.

These conditions are formalised in the rank function theorem, which is the heart of the approach:

Theorem 1. *If* $\rho : MESSAGE \cup SIGNAL \to \mathbb{Z}$ *is such that:*

1. $\forall\, m \in IK.\rho(m) > 0$
2. $\forall\, S \subseteq MESSAGE.(\rho(S) > 0 \wedge S \vdash m) \Rightarrow \rho(m) > 0$
3. $\rho(b) \leqslant 0$
4. $\forall\, i.(USER_i \,|[\, a \,]|\, Stop)$ **sat** $\rho(tr \upharpoonright rec) > 0 \Rightarrow \rho(tr) > 0$

then $(|||_i\, USER_i) \,|[\, trans, rec \,]|\, ENEMY$ **sat** a precedes b.

Here we have abused notation, and extended ρ to apply not only to messages and signals, but also to events, traces, and sets:

- $\rho(c.m) = \rho(m)$
- $\rho(tr) = \min\{\rho(s) \mid s \text{ in } tr\}$
- $\rho(S) = \min\{\rho(s) \mid s \in S\}$

Thus, if we can find a rank function ρ which meets the four conditions above, then we will have established that the system as described meets the corresponding authentication property expressed as a precedes b.

3.1 Preserving rank

The first three conditions of the rank function theorem can be checked independently of any CSP protocol description. However, the fourth condition requires verification of CSP processes against a specification. The benefits of using the

INTERLEAVING	$\dfrac{\forall\, i.(P_i \textbf{ sat } \mathsf{maintains}\ \rho)}{\left\|\right\|_i P_i \textbf{ sat } \mathsf{maintains}\ \rho}$
EXTERNAL CHOICE	$\dfrac{\forall\, i.(P_i \textbf{ sat } \mathsf{maintains}\ \rho)}{\Box_i P_i \textbf{ sat } \mathsf{maintains}\ \rho}$
PREFIXING	$\dfrac{P \textbf{ sat } \mathsf{maintains}\ \rho \qquad \rho(e) > 0}{e \rightarrow P \textbf{ sat } \mathsf{maintains}\ \rho}$
STOP	$\dfrac{}{\mathit{Stop} \textbf{ sat } \mathsf{maintains}\ \rho}$
INPUT	$\dfrac{\forall\, x.(\rho(f(x)) > 0 \Rightarrow P(x) \textbf{ sat } \mathsf{maintains}\ \rho)}{\mathit{rec}.i?j?f(x) \rightarrow P(x) \textbf{ sat } \mathsf{maintains}\ \rho}$

Fig. 4. Composition rules for $\mathsf{maintains}\ \rho$

CSP traces model is that a number of application-specific rules can be identified, and applied in this particular kind of verification. We are interested in the property $\mathsf{maintains}\ \rho$:

$$\mathsf{maintains}\ \rho \mathrel{\widehat{=}} \rho(tr \upharpoonright rec) > 0 \Rightarrow \rho(tr) > 0$$

Figure 4 identifies some compositional rules which are useful for establishing this property.

The last rule in this figure requires some explanation. It concerns input of a message which matches a particular pattern $f(x)$, with subsequent behaviour $P(x)$. If we can show that $P(x)$ **sat** $\mathsf{maintains}\ \rho$ whenever the input has positive rank, then we can conclude that the inputting process $\mathit{rec}.i.j?f(x) \rightarrow P(x)$ also maintains positive rank. We are not concerned with $P(x)$ for which $\rho(f(x)) \leqslant 0$, since in such cases the non-positive-rank message must have been introduced externally to the process, and so we do not need to consider whether $P(x)$ maintains positive rank.

3.2 Verifying the modified protocol

We aim to identify a rank function which meets the four conditions of the rank function theorem. In devising a rank function it is helpful to consider the sorts of messages that can legitimately pass on the network. Furthermore, the nature of the generates relation $\vdash$, and the CSP protocol descriptions, impose constraints on any putative function ρ.

For the fourth condition, we are required to show, for an arbitrary user C, that:

$$\mathit{USER}_C \,|[\, \mathit{respgo}.B.A.n_A \,]|\, \mathit{Stop} \textbf{ sat } \mathsf{maintains}\ \rho$$

We have that:

$$\begin{aligned} & USER_C \,|[\, respgo.B.A.n_A \,]|\, Stop \\ = & \Big|\Big|\Big|_n \, \Box_i (INIT_C(i,n) \,|[\, respgo.B.A.n_A \,]|\, Stop) \\ & ||| \Big|\Big|\Big|_n (RESP_C(n) \,|[\, respgo.B.A.n_A \,]|\, Stop) \end{aligned}$$

In order to show that this combination satisfies maintains ρ, the inference rules for interleaving and choice in Figure 4 mean that we have only to establish that each component separately maintains ρ. In other words, for each C, i, and n, we have to establish:

$$\begin{aligned} INIT_C(i,n) \,|[\, respgo.B.A.n_A \,]|\, Stop \textbf{ sat } \text{maintains } \rho \\ RESP_C(n) \,|[\, respgo.B.A.n_A \,]|\, Stop \textbf{ sat } \text{maintains } \rho \end{aligned}$$

There are a number of cases to consider:

Case $INIT_C$, $C = A, i = B, n = n_A$ In this case we have

$$\begin{aligned} & INIT_A(B, n_A) \,|[\, respgo.B.A.n_A \,]|\, Stop = \\ & \quad trans.A.B.n_A \rightarrow rec.A.B?\{B.n_A.y\}_{K_{AB}} \\ & \qquad \rightarrow trans.A.B.y \rightarrow initdone.A.B.n_A \rightarrow Stop \end{aligned}$$

We know from condition 3 that $initdone.A.B.n_A$ must have non-positive rank, since this is the signal whose non-occurrence we wish to establish. If we are to apply the rules for prefixing to establish that this process satisfies maintains ρ, then we require that the message input in step 2 of the protocol must have non-positive rank. This follows because the behaviour following a positive rank input must itself satisfy maintains ρ—and this is not possible because $initdone.A.B.n_A$ (necessarily non-positive rank) is performed.

Thus we obtain a constraint on the rank function we are searching for to establish correctness: that any message of the form $\{B.n_A.y\}_{K_{AB}}$ must have non-positive rank.

Case $RESP_B$ A second case which is of interest is that of agent B as responder. In this case we have that

$$\begin{aligned} & RESP_B(y) \,|[\, respgo.B.A.n_A \,]|\, Stop = \\ & rec.B?i?x \rightarrow \\ & \begin{cases} respgo.B.i.x \rightarrow trans.B.i.\{B.x.y\}_{K_{Bi}} & \\ \quad \rightarrow rec.B.i.y \rightarrow Stop & \text{if } i \neq A \text{ or } x \neq n_A \\ Stop & \text{if } i = A \text{ and } x = n_A \end{cases} \end{aligned}$$

The particular run with A and nonce n_A is blocked, but all other runs are allowed.

If the input message x has positive rank, and the first branch of the condition is followed, then we have that either $i \neq A$, or $x \neq n_A$. In this case the transmitted message $\{B.x.y\}_{K_{Bi}}$ should also have positive rank, since this protocol run should not introduce non-positive-rank messages.

$$
\begin{aligned}
\rho(i) &= 1 \\
\rho(n) &= 1 \\
\rho(k) &= \begin{cases} 0 \text{ if } k = k_{AB} \\ 1 \text{ otherwise} \end{cases} \\
\rho(m_1.m_2) &= \min\{\rho(m_1).\rho(m_2)\} \\
\rho(\{m\}_k) &= \begin{cases} 0 \quad \text{if } m = B.n_A.y \text{ and } k = k_{AB} \\ \rho(m) \text{ otherwise} \end{cases} \\
\rho(sig) &= \begin{cases} 0 \text{ if } sig = initdone.A.B.n_A \\ 1 \text{ otherwise} \end{cases}
\end{aligned}
$$

Fig. 5. A rank function for authentication

A candidate rank function The constraints arising from the two cases above give rise to the first attempt at a rank function. This is given in Figure 5. In this rank function, we give a rank of 0 to those messages and signals identified above as requiring non-positive rank, and also the shared key K_{AB}, which must remain out of the hands of the attacker to prevent him from constructing messages that should not circulate. Other atomic messages (nonces, agent identities, other keys) can have rank 1. Other compound messages essentially have the ranks dictated by their components: if a message's content has rank 0, then any encryption or concatenation of that message will likewise have a rank of 0.

It is straightforward to check that condition 2 holds for this rank function, and it is entirely reasonable to state that in the model the attacker does not start with any message of rank 0, as required by condition 1.

Lastly, we are required to show that condition 4 holds for all other cases. However, since these cases do not involve the important signals or protocol messages their proofs are all straightforward:

- $RESP_A$: only generates messages $\{A.x.n\}_{K_{Ai}}$, which are of positive rank;
- $INIT_A(i, n)$, where $i \neq B$ or $n \neq n_A$. In this case, the signal provided at the end of the run will have rank 1, so no message or signal of non-positive rank is produced;
- $INIT_B(i, n)$: only produces messages and signals of positive rank;
- $INIT_C$, $RESP_C$ $(C \neq A, B)$: only produce messages and signals of positive rank.

Thus the rank function is sufficient to establish that the corrected protocol indeed provides authentication of B to A.

It is instructive to see where this approach fails on the original flawed protocol. In that protocol there is no agent name included in the second message. When considering the case $INIT_C, C = A, i = B, n = n_A$, we will obtain the requirement that $\{n_A.y\}_{K_{AB}}$ must have non-positive rank for any y, since it is the input of such a message that leads to the performance of the non-positive-

rank signal $initdone.A.B.n_A$[1]. However, consideration of the case $RESP_A(n)$ finds that messages of the form $\{x.n\}_{K_{Ai}}$ are output, and so these will need to have positive rank, for any x and i[2]. But now there is a conflict on the message $\{n_A.n\}_{K_{AB}}$, which from the first case must have non-positive rank, but from the second case must have positive rank. Hence there can be no rank function for this version of the protocol: the constraints on any rank function are contradictory. The contradiction is avoided by introducing the name of the agent generating the message.

4 Discussion

4.1 Theorem-proving

In practice, of course, protocols tend to be more complicated than our running example, in a variety of ways. For example: the messages used in the protocol might be more complex, or simply much larger; there may be more messages involved in the protocol; the protocol could involve additional protocol agents, such as trusted third parties, or even entire groups of communicating agents; more complex combinations of cryptographic mechanisms might be used. All of these possibilities make the CSP modelling of the protocol a more difficult task, and the verification of candidate rank functions becomes more intricate and error-prone. Tool support is of great benefit in keeping track of the housekeeping involved in consideration of numerous cases, and in assisting in the construction of rank functions.

The constraints introduced by the rank function theorem can generally be used to derive a candidate rank function. Firstly, every message in IK must have positive rank. Secondly, any message derivable from a set of positive rank messages must also have positive rank. Thirdly, any output (message or signal) from a protocol step which follows only positive rank inputs must also be of positive rank. These three conditions allow the identification of a set S of messages and signals which must have positive rank. However, the signal required in condition 3 of the rank function theorem is required to have non-positive rank. If that signal is in the set S then no rank function can exist. Otherwise the function ρ which gives a rank of 1 to all messages in S, and a rank of 0 to all other messages, will be a suitable rank function.

The *RankAnalyser* tool [HS00,Hea00,HS04] provides a way of computing this rank function automatically for standard cases (where the protocol uses public-key or shared-key cryptography, nonces, agent names, and concatenation). The (infinite) message space is partitioned to a finite set of equivalence classes, and the set of messages and signals of positive rank is obtained by repeatedly applying protocol steps and generates rules (on the equivalence classes), starting from the attacker's initial knowledge IK.

[1] In the proof of the correct protocol, the corresponding requirement was that $\{B.n_A.y\}_{K_{AB}}$ must have non-positive rank

[2] Previously, it was required that $\{A.x.n\}_{K_{AB}}$ had positive rank

More generally, the PVS theorem prover [OSR93] has also been used to support rank function proofs of protocol correctness. Theorem provers such as PVS are well-suited to keeping track of all the unavoidable detailed housekeeping involved in the nuts and bolts of a protocol correctness proof. The traces model for CSP has been embedded in PVS, together with much of the consequent theory, including proof rules such as those of Figure 4 and the rank function theorem has been proved for this embedding [DS97]. Specific protocols can be modelled and verified, for example the recursive authentication protocol analysed in [BS97], demonstrating that this approach supports the full generality of an infinite message space, and arbitrary numbers of runs and protocol agents. More recently the CSP hierarchy of theories within PVS has been restructured [Eva03,ES04] to more easily allow extensions within the rank function framework, such as the introduction of (discrete) time [ES00], as well as consideration of other properties such as non-repudiation [Eva03].

The rank function approach has also been extended in other ways. It is able to incorporate algebraic properties of the cryptographic mechanisms into the analysis, provided they can be expressed appropriately within the model [Sch02]. For example, if Vernam encryption (exclusive-or) is used explicitly within a protocol, then the algebraic properties of exclusive-or should be taken into account in the analysis. This can be achieved by giving the algebraic identities that encapsulate exclusive-or on the message space, and checking that whenever two messages are equivalent then they should have the same rank. This approach is clearly limited since only *known* algebraic properties can be included in the model. Nonetheless their inclusion allows the protocol analyser to reason about the properties which a cryptosystem must satisfy if the protocol is to be implemented correctly.

Another extension concerns the verification of secrecy properties of protocols, particularly in situations where keys can be leaked to an attacker without compromising the security of past protocol runs. Such keys are temporary secrets: components of messages that are required to be unknown to the attacker at a particular point of the protocol, but can be disclosed later. The standard rank function approach cannot handle temporary secrets, because their rank should be non-positive at the point they are used, but positive because of the fact that the attacker learns them during the protocol run [DS04]. Temporal rank functions are a generalisation that take into account the time at which a message can first be learned by the attacker, enabling a finer way of analysing the relationships between messages. Use of temporal rank functions requires a generalisation of the rank function theorem, but they allow analysis of an additional class of secrecy properties not covered by the standard approach.

4.2 Model-checking

The use of CSP to describe and specify protocols naturally enables the use of model-checking for verification, and there has been a significant body of work using FDR [For03] in this area which began a decade ago [Low95,Ros95,LR96]. The approach constructs a CSP description of the protocol agents interacting over a Dolev-Yao style attacker as described earlier, and refinement-checks it

against authentication and secrecy properties expressed as CSP trace specifications in terms of the signal events which are inserted judiciously into the protocol runs. If the refinement check fails then FDR produces a (minimal-length) counterexample trace which corresponds to an attack on the protocol: a sequence of messages which lead to a failure of the authentication or secrecy property under consideration.

Since the construction of the model of the protocol is routine from the message-passing protocol description, Lowe has developed a tool, Casper [Low98], which translates a high-level protocol description into the corresponding CSP model, ready for FDR to analyse. The ease of use of this tool, together with the speed of the FDR analysis, means that the model-checking analysis should generally be the first to be carried out when considering a new protocol: simple flaws can be identified and corrected quickly, before too much effort is put into carrying out a rank function proof.

Of course, any CSP model which can be completely checked by FDR must have a finite number of states. This means that the number of protocol runs in the model, the number of agents, and the size of the message space, must necessarily be finite. Refinement failures will always correspond to attacks, but a successful refinement check on a finite model does not guarantee correctness in the presence of arbitrary concurrent runs—it may be that an attack requires more possibilities than have been included in the analysis. However, a collection of sophisticated techniques have been developed for enabling more general conclusions to be drawn from finite model-checking. For example, Lowe [Low99] has presented, for secrecy specifications, a list of conditions under which the correctness of just a single run of a protocol is sufficient to conclude the correctness of an unbounded number of runs of the same protocol. Hui and Lowe have shown how protocol messages in CSP models can be simplified without losing attacks (fault-preserving transformations) [HL01], thus enabling complex protocols to be reduced to a point where they can be analysed by FDR. Broadfoot and Roscoe have applied data independence techniques [BR99,BR02] which allow results about a finite number of runs to be lifted to arbitrary runs.

An extensive coverage of the use of CSP for modelling protocols, and both the model-checking and the rank function approaches to protocol analysis, is provided in [RSG+00].

4.3 Related approaches

In addition to the CSP approaches discussed above, a wide variety of formal techniques have been developed for protocol specification and analysis. These include approaches based on graph theory, induction, multiset rewriting, type-checking, and non-interference. Here we give a flavour of each.

In the strand space approach [THG99], a strand is a trace that represents either the execution of a legitimate protocol participant (an 'honest' strand) or the action of an attacker (a 'penetrator' strand). A strand space is a collection of strands equipped with a graph structure that represents both consecutive operations on the same strand (the behaviour of a single user) and the interaction

between strands (communication between users). Theorems have been developed on strand spaces which enable proofs that a protocol is correct, and tool support for the approach has been provided by Athena [SBP01], a program that is part model-checker and part theorem-prover. Some relationships have been identified [Hea02] between the rank functions used to verify protocols, and the structures (ideals) used in the strand spaces approach, and there are some similarities in the philosophies of the two approaches.

The inductive approach [Pau98] uses the theorem-prover Isabelle/HOL to support a theorem-proving approach to protocol verification. Protocols are coded directly in terms of event traces and rules that participants apply to 'received' messages in order to produce new messages. The possible actions of a 'Spy' are also specified by rules. A theory concerning the possible traces of the overall system is developed and the protocol is verified by establishing inductively that no trace violating the specification can ever occur. A particular achievement of this approach is its use in the verification of SET [Pau02], an electronic commerce protocol whose description runs to nearly 1000 pages.

Cervesato et al. [CDL+99a] have developed a way of specifying protocols using first-order multiset rewriting. This has become known as the MSR approach. Using MSR, protocols are specified by *roles* which represent the behaviour of protocol participants. Each role constitutes a series of rewrite rules which represent the actions of that particular user. The attacker, typically in the style of Dolev and Yao, is also defined via rewrite rules. Each rewrite rule that an attacker can apply corresponds to a deduction of the form $\vdash$ in the rank function approach. Recent work has sought to establish a correspondence between MSR and the strand space [CDM+00] and process algebraic approaches [BCLM03].

Abadi and Gordon have proposed the spi-calculus [AG98] as an extension to the π-calculus which includes cryptographic primitives. Protocols in the spi-calculus are modelled as processes — but the similarity with CSP ends here. The fundamental differences between CSP and nominal calculi mean that, in the spi-calculus, communication of secrets between parties is achieved via restriction and scope extrusion, and the nature of testing equivalence removes the need for an explicit attacker process. However, proving correctness via equivalence can be difficult. Abadi [Aba97] and, more recently, Gordon and Jeffrey [GJ01,GJ04] have therefore developed type-systems that enable authentication properties — expressed using signals — to be statically checked for a spi-calculus protocol model. The use of correspondence assertions (in the spirit of the *initdone* and *respgo* events) suggests a similarity between this approach and the rank function approach, and it is also interesting to consider that the 'trusted' and 'untrusted' secrecy types may be interpreted as non-positive and positive ranks, respectively.

The concept of non-interference has also formed the basis of protocol analysis techniques. These approaches generally impose a partition on protocol agents, with a group of 'high-level' privileged users distinguished from other 'low-level' users. Non-interference is achieved if the behaviour of a high-level user has no effect on what a low-level user can observe. For the purposes of protocol analysis this corresponds to the inability of an attacker (a high-level user) to induce

bad behaviour in the legitimate participants (the low-level users). A suite of tools have been developed that enable protocols to be reasoned about using non-interference. A high-level protocol description can be translated into the notation of the Security Process Algebra (SPA) using the CVS compiler [DFG00]. This SPA script is then amenable for analysis using the CoSec tool [FG97] which checks for the presence of non-interference. Similarities between non-interference and the concept of process equivalence in CSP have been established [RS00].

The above techniques, along with the CSP-based approaches, have much in common, most notably their basic assumption about the capabilities of the attacker. Indeed, in many cases it will be feasible to reason about a protocol using any one of these methods, and the results obtained from each will be broadly similar. As alluded to above, there is a growing body of research which aims to demonstrate fundamental similarities between these different approaches. In the end the choice of which technique to use will be guided by the previous experience of the protocol analyser. The advantage of applying CSP in this domain lies in the simplicity of the notation and the transparency with which protocols can be modelled. This transparency is essential for a model to be shown as an appropriate abstraction of a real protocol. Furthermore, the maturity of the language backs this up by allowing well-understood and powerful techniques to be brought to bear on the problem of verifying whether a given protocol model meets its intended goal.

References

[Aba97] M. Abadi. Secrecy by typing in security protocols. In *Proceedings of the Third International Symposium on Theortical Aspects on Computer Software*, number 1281 in Lecture Notes in Computer Science, pages 611–638, September 1997.

[AG98] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 1998. also DEC Research Report 149, 1998.

[BCLM03] S. Bistarelli, I. Cervesato, G. Lenzini, and F. Martinelli. Relating process algebras and multiset rewriting for security protocol analysis. In *WITS '03: Workshop on Issues in the Theory of Security*, 2003.

[BR99] P.J. Broadfoot and A.W. Roscoe. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(2/3), 1999.

[BR02] P.J. Broadfoot and A.W. Roscoe. Capturing parallel attacks within the data independence framework. In *Proceedings of the 15th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2002.

[BS97] J.W. Bryans and S.A. Schneider. CSP, PVS, and a recursive authentication protocol. In *DIMACS Workshop on Design and Formal Verification of Crypto Protocols*, 1997.

[CDL+99a] I. Cervesato, N.A. Durgin, P. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[CDL+99b] I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[CDM+00] I. Cervesato, N. Durgin, J.C. Mitchell, P. Lincoln, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[DFG00] A. Durante, R. Focardi, , and R. Gorrieri. A compiler for analyzing cryptographic protocols using non-interference. *ACM Transactions on Software Engineering and Methodology*, 9(4), 2000.

[DS97] B. Dutertre and S.A. Schneider. Embedding CSP in PVS: an application to authentication protocols. In *tpHOL*, 1997.

[DS04] R. Delicata and S.A. Schneider. Towards the rank function verification of protocols with temporary secrets. In *WITS '04: Workshop on Issues in the Theory of Security*, 2004.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

[ES00] N. Evans and S.A. Schneider. Analysing time dependent security properties in CSP using PVS. In *ESORICS*, volume 1895 of *LNCS*, 2000.

[ES04] N. Evans and S.A. Schneider. Verifying security protocols with PVS: Widening the rank function approach. *Journal of Logic and Algebraic Programming*, in press

[Eva03] N. Evans. *Investigating Security Through proof.* PhD thesis, Royal Holloway, University of London, 2003.

[FG97] R. Focardi, , and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), 1997.

[For03] Formal Systems (Europe) Ltd. FDR2 user manual, 2003.

[GJ01] A.D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of the 14th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

[GJ04] A.D. Gordon and A. Jeffrey. Types and effects for asymmetric cryptographic protocols. *Journal of Computer Security*, 12(3/4), 2004. Also in Proceedings of the 15th Computer Security Foundations Workshop. IEEE Computer Society Press, 2002.

[Hea00] J.A. Heather. *"Oh! Is it* really *you?"—Using rank functions to verify authentication protocols.* PhD thesis, Royal Holloway, University of London, 2000.

[Hea02] J.A. Heather. Strand spaces and rank functions: More than distant cousins. In *Proceedings of The 15th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2002.

[HL01] M.L. Hui and G. Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1/2), 2001.

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

[HS00] J.A. Heather and S.A. Schneider. Towards automatic verification of authentication protocols on unbounded networks. In *Proceedings of the 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.

[HS04] J.A. Heather and S.A. Schneider. A decision procedure for the existence of a rank function. *Journal of Computer Security*, in press

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

[Low97] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[Low98] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1/2), 1998.

[Low99] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2/3), 1999.

[LR96] G. Lowe and A.W. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 1996.

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

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

[MVV96] A.J. Menezes, P.C. Van Oorschott, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), 1978.

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

[Pau98] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2), 1998.

[Pau02] L. Paulson. Verifying the SET protocol: Overview. In *FASec 2002: Formal Aspects of Security*, 2002.

[PQ00] O. Pereira and J-J. Quisquater. On the perfect encryption assumption. In *WITS '00: Workshop on Issues in the Theory of Security*, 2000.

[Ros95] A.W. Roscoe. Modeling and verifying key-exchange protocols using CSP and FDR. In *Proceedings of the 8th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1995.

[Ros97] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1997.

[RS00] P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1/2), 2000. Also in Proceedings of the 12th Computer Security Foundations Workshop. IEEE Computer Society Press, 1999.

[RSG+00] P.Y.A. Ryan, S.A. Schneider, M.H. Goldsmith, G. Lowe, and A.W. Roscoe. *Modelling and Analysis of Security Protocols*. Addison-Wesley, 2000.

[SBP01] D.X. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2), 2001.

[Sch98] S.A. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 1998.

[Sch99] S.A. Schneider. *Concurrent and Real-time Systems: the CSP Approach*. Addison-Wesley, 1999.

[Sch02] S.A. Schneider. Verifying security protocol implementations. In *FMOODS'02: Formal Methods for Open Object-based Distributed Systems*, 2002.

[THG99] F.J. Thayer Fábrega, J.C. Herzog, and J.D. Guttman. Strand spaces: proving security protocols correct. *Journal of Computer Security*, 7(1), 1999.

[WL93] T. Woo and S. Lam. A semantic model for authentication protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.