CSP, PVS and a Recursive Authentication Protocol

Jeremy Bryans and Steve Schneider
Department of Computer Science
Royal Holloway and Bedford New College
Egham, Surrey, TW20 0EX

1 Introduction

In this paper we consider the nature of machine proofs used in the CSP approach to the verification of authentication protocols.

In [Sch96], a general method is presented for the analysis and verification of authentication protocols using the process algebra CSP [Hoa85]. The CSP syntax provides a natural and precise way to describe such protocols in terms of the messages accepted and transmitted by the individual protocol participants. The CSP traces model provides a formal framework for reasoning about these protocols.

In the CSP method, authentication is considered to be message-oriented: $m2$ authenticates $m1$ if the receipt of $m2$ guarantees the previous transmission of $m1$, even in a hostile environment. To facilitate proofs, a notion of a rank function is developed in [Sch96]. This is an integer-valued function on the message space, such that all messages apart from $m1$ which could possibly circulate in the network have a rank greater than zero, and the message which provides authentication ($m2$ above) has a rank of zero or below. It is then sufficient to prove that no messages of rank zero or below are able to circulate when message $m1$ is blocked.

However, proving authentication is still an arduous task. The principal problem is in the complexity of the message space, which gives rise to a mass of detail in proofs, requiring a significant amount of detailed housekeeping. For this reason, the CSP traces theory has been embedded within PVS [DS97a, SOR93], and this description has been successfully used to mechanise several proofs of authentication properties [DS97b, BS97]. However, even with mechanical support the construction of proofs is far from easy, because of the inherent complexity involved in modelling all the possibilities of malicious action.

In this paper, we consider a novel authentication protocol, proposed in [BO97]. The protocol can be used in various ways: we take the purpose to be that of establishing an uncompromised chain of session keys between adjacent pairs of agents involved in the protocol run.

This protocol provides an interesting extension to the work cited above, because very few aspects of an individual protocol run are fixed in advance. There may be an arbitrary number of agents, and consequently there may be an arbitrary number of messages, which may grow to arbitrary lengths.

Despite the extra complexity of the protocol, adapting the techniques developed in [DS97a] to prove that the session keys are uncompromised turned out to be relatively straightforward, and the proofs of authentication were not significantly more complex. We present the rank function used, and show how PVS uses the rank function to prove the authentication property.

In [RS97], an attack is described on an implementation of this protocol and a correction is proposed. We go on to identify where the particular implementation decisions made compromised the protocol, and how the proof of authentication for the original definition of the protocol fails when applied to the faulty implementation. We also provide an analysis of the corrected protocol, and verify that the attack is no longer possible. Finally, we speculate on how failed proofs may lead us to discover attacks.

2 CSP

In [Sch96] a general framework for analysing security properties within the process algebra CSP is presented. Only a limited number of CSP operators are necessary. If a is a CSP event, A a set of events and P and Q CSP processes, then the prefix operator $a \rightarrow \mathrm{P}$ describes a process which performs an a and then behaves as process P. The choice operator $P \Box Q$ describes a process which offers a choice between process P and process Q, and it has an indexed form $\Box_{a \in A} P_a$, which offers a choice between all of the processes P_a. The choice is resolved by the first action to occur. The parallel operator $P \,|[\, A \,]|\, Q$ forces P and Q to synchronise on actions from the set A, but otherwise execute independently. The hiding operator $P \setminus A$ hides the events in set A, which means that no other process can participate in occurrences of these events. The atomic process *Stop* marks the termination of a process.

2.1 Traces

In [Sch96], the traces model is used as the basis for the proof rules presented. In this model, the semantics of a process P is defined to be the set of traces (sequences of events) that it may possibly perform. For example,

$$traces(a \rightarrow b \rightarrow Stop) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle\}$$
$$traces(a \rightarrow b \rightarrow Stop \,|[\, \{b\} \,]|\, b \rightarrow c \rightarrow Stop) = \{\langle\rangle, \langle a\rangle, \langle a, b\rangle, \langle a, b, c\rangle\}$$

A useful operator on traces is projection: If D is a set of events then the trace $tr \upharpoonright D$ is defined to be the maximal subsequence of tr all of whose events are drawn from D. If D is a singleton set $\{d\}$ then we overload notation and write $tr \upharpoonright d$ for $tr \upharpoonright \{d\}$. Message extraction $tr \downarrow C$ for a set of channel names C provides the maximal sequence of messages passed on channels C. Finally, $tr \Downarrow C$ provides the set of messages in tr passed along some channel in C. These may be described inductively on sequences, and the last by a set comprehension:

$$
\begin{aligned}
\langle\rangle \upharpoonright D &= \langle\rangle \\
(\langle d\rangle \frown tr) \upharpoonright D &= \begin{cases} \langle d\rangle \frown (tr \upharpoonright D) & \text{if } d \in D \\ (tr \upharpoonright D) & \text{otherwise} \end{cases} \\
\langle\rangle \downarrow C &= \langle\rangle \\
(\langle d\rangle \frown tr) \downarrow C &= \begin{cases} \langle m\rangle \frown (tr \downarrow C) & \text{if } \exists\, c \in C .\ d = c.m \\ (tr \downarrow C) & \text{otherwise} \end{cases} \\
tr \Downarrow C &= \{m \mid (tr \downarrow C) \upharpoonright m \neq \langle\rangle\}
\end{aligned}
$$

If tr is a sequence, then $\sigma(tr)$ is the set of events appearing in the sequence. The operator σ extends to processes: $\sigma(P)$ is the set of events that appear in some trace of P.

In the traces model, if $traces(Q) \subseteq traces(P)$ then we say that Q is a refinement of P, written $P \sqsubseteq Q$.

3 The protocol

In [BO97] an authentication protocol is proposed, which is further explained in [Pau97]. This protocol operates over an arbitrarily long chain of protocol agents, terminating with a key-server. We set out to verify that a run of the protocol establishes an uncompromised chain of session keys between adjacent pairs of agents. The protocol operates as follows, where $\mathsf{Hash}X$ is the hash of a message X and $\mathsf{Mac}_K X$ is the pair $\{\mathsf{Hash}\{K, X\}, X\}$. In the protocol description, K will be an agent's long-term shared key, and the hashed message $\mathsf{Hash}\{K, X\}$ (a *message authentication code*) will allow the server to check that message X originated with the owner of key K. K_x is the long-term key of agent X, K_{xy} is a session key between agent X and agent Y, N_x is a fresh nonce and null is a placeholder.

Agent A initiates a run by sending the following message.

1. $A \to B : \mathsf{Mac}_{K_a}\{A, B, N_a, \mathsf{null}\}$

Agent B responds by sending a similar message to agent C, but replacing the placeholder with A's entire message.

$$2.\ B \rightarrow C : \mathsf{Mac}_{K_b}\{B, C, N_b, \mathsf{Mac}_{K_a}\{A, B, N_a, \mathsf{null}\}\}$$

This step is repeated for each subsequent agent in the chain, and each agent adds new components to the message, and passes it on. This stage terminates when some agent specifies the server as the recipient. Suppose, for example, that C sends the message to the server.

$$3.\ C \rightarrow S : \mathsf{Mac}_{K_c}\{C, S, N_c, \mathsf{Mac}_{K_b}\{B, C, N_b, \mathsf{Mac}_{K_a}\{A, B, N_a, \mathsf{null}\}\}\}$$

The server now unpacks this message, and prepares session keys for each adjacent pair of agents in the chain. Considering the outer two levels of the protocol, we can see that agent C was called by agent B, and called agent S (the server). The server therefore generates the session keys K_{bc} and K_{cs}, prepares two certificates, and encrypts them with agent C's secret key: $\{K_{cs}, S, N_c\}_{K_c}$ and $\{K_{bc}, B, N_c\}_{K_c}$. In a sense, the key K_{cs} is redundant, because agent C has a key with the server already — its longterm key K_c. However, including it allows the final agent in the chain to be treated like any other agent.

Ignoring the first level of the message now, and considering the second and third levels, the server creates two certificates for agent B: $\{K_{bc}, C, N_b\}_{K_b}$ and $\{K_{ab}, A, N_b\}_{K_b}$, encrypted with agent B's secret key.

The third level of the message contains the placeholder null, which indicates to the server that this is the last level of the message. It therefore prepares only one further certificate: $\{K_{ab}, B, N_a\}_{K_a}$.

In the next step, the server returns the all certificates to the last agent on the chain:

$$\begin{aligned} 4.\ S \rightarrow C : &\{K_{cs}, S, N_c\}_{K_c}, \{K_{bc}, B, N_c\}_{K_c}, \{K_{bc}, C, N_b\}_{K_b}, \\ &\{K_{ab}, A, N_b\}_{K_b}, \{K_{ab}, B, N_a\}_{K_a} \end{aligned}$$

Agent C removes the relevant certificates and forwards the rest to agent B, which in turn passes the final one on to agent A.

$$\begin{aligned} &5.\ C \rightarrow B : \{K_{bc}, C, N_b\}_{K_b}, \{K_{ab}, A, N_b\}_{K_b}, \{K_{ab}, B, N_a\}_{K_a} \\ &6.\ B \rightarrow A : \{K_{ab}, B, N_a\}_{K_a} \end{aligned}$$

3.1 CSP Description

The datatype used to model the possible messages is given by

$$\begin{aligned} MESSAGE ::= {} & TEXT \mid NONCE \mid USER \mid KEY \mid \\ & MESSAGE.MESSAGE \mid \\ & encrypt(KEY, MESSAGE) \mid \\ & hash(KEY, MESSAGE) \end{aligned}$$

where $TEXT$, $NONCE$, and $USER$ are all primitive sets. The set KEY further subdivides into $SESSION$ and $LONGTERM$, representing two different ways in which keys are used: typically, session keys are only used for a single run of the protocol, whereas longterm keys are used repeatedly.

$$KEY ::= SESSION \mid LONGTERM$$

If m, m_1, m_2 are arbitrary messages, and k is a key, then the generates relation $\vdash$ for this message space is defined by the following rules:

- $m \in S$
- $(\forall\, m' \in S'.S \vdash m') \wedge S' \vdash m \Rightarrow S \vdash m$
- $S \vdash m_1 \wedge S \vdash m_2 \Rightarrow S \vdash m_1.m_2$
- $S \vdash m_1.m_2 \Rightarrow S \vdash m_1 \wedge S \vdash m_2$
- $S \vdash m \wedge S \vdash k \Rightarrow S \vdash encrypt(k, m)$
- $S \vdash k \wedge S \vdash encrypt(k, m) \Rightarrow S \vdash m$
- $S \vdash m \wedge S \vdash k \Rightarrow S \vdash hash(k, m)$

Observe that hashing is one-way: there is no rule which allows information to be extracted from a hashed message. Observe also that all keys in this protocol are symmetric: knowledge of a key allows any message encrypted with that key to be decrypted.

The protocol describes the required behaviour of each of its participants. We will use CSP processes to describe the behaviour of each of the participating agents. For simplicity, we will consider in this paper a single run of the protocol, though the approach extends naturally to multiple concurrent runs, as discussed in [Sch96].

We model an agent i as sending all of its messages on to the medium and receiving all its messages from the medium through the channels $trans.i$ and $rec.i$ respectively, as illustrated in Figure 1.

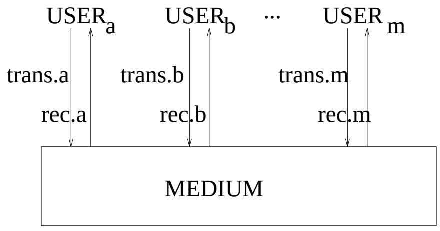

Figure 1: CSP model of the network

Messages on these channels have the type $USER.USER.MESSAGE$, where $USER$ is the set of all protocol agents names, $trans.i.j.m$ represents the transmission onto the medium of message m from $USER_i$ addressed to $USER_j$, and $rec.i.j.m$ represents the reception of message m by $USER_i$, labelled as coming from $USER_j$. The message m is drawn from the abstract data-type $MESSAGE$. The users are defined according to the protocol that we are analysing.

The initiator of the protocol, agent A, is described as

$$\begin{aligned} USER_A = {} & trans.A!B.mac(lt_A, A.B.N_A.null) \rightarrow \\ & rec.A.B?encrypt(lt_A, s.B.N_A) \rightarrow Stop \end{aligned}$$

where N_A is a fresh nonce. Agent A transmits an initiating request to agent B on channel $trans.A$ and awaits a reply on channel $rec.A$.

The freshness of N_A is modelled by the fact that it is not initially known by the enemy: $N_A \notin INIT$, where $INIT$ will be used to model the information known by the enemy at the start of the protocol run.

After sending out the initial request, the process is prepared to accept any message which is labelled as coming from B, is encrypted with its long term key and contains both its nonce challenge N_A and the agent B's identity. It will accept the key s as a session key generated by the server for private use between A and B.

Intermediate nodes along the chain all have the same form. If the node next up the chain from B is C then the appropriate description is as follows:

$$\begin{aligned} USER_B = {} & rec.B?i?mac(lt_k, i.B.N_x.m) \rightarrow \\ & trans.B!C!mac(lt_B, B.C.N_B.mac(lt_k, i.B.N_x.m)) \rightarrow \\ & rec.B.C?encrypt(lt_B, skup.C.N_B).encrypt(lt_B, skdown.i.N_B).x \rightarrow \\ & trans.B!i!x \rightarrow Stop \end{aligned}$$

Agent B receives a request from some agent, which it packages suitably and sends on to its successor (Agent C in this case.) It then receives a message consisting of a list of key certificates. The first two certificates contain sessions keys for communication with the agent immediately below (*skdown*) and above (*skup*). The rest of the message is passed to the agent i from whom the original request was received.

The server inputs a message which consists of a nested series of requests, and then outputs a message which is a concatenation of all of the key certificates encrypted for the appropriate agents.

$$SERVER = rec.S?i?m \rightarrow trans.S!i!response(m) \rightarrow Stop$$

The function *response* defined on the possible legitimate requests that may arrive at the server is defined inductively as follows:

$$response(mac(lt_i, i.j.N_i.null)) = encrypt(lt_i, s_{ij}.j.N_i)$$

$$\begin{aligned} &response(mac(lt_j, j.k.N_j.mac(lt_i, i.j.N_i.x))) = \\ &\quad encrypt(lt_j, s_{jk}.k.N_j).encrypt(lt_j, s_{ij}.i.N_j).response(mac(lt_i, i.j.N_i.x)) \end{aligned}$$

The session keys s_{ij} generated by the server are all fresh and unguessable: none appear in the set $INIT$.

The protocol operates in a hostile environment. This is also modelled within CSP in order to facilitate analysis. The approach taken is to provide a CSP description of the Dolev-Yao model [DY83]. In this model, it is assumed that the medium is under the complete control of the enemy, which can block, re-address, duplicate and fake messages.

The network description consists of a set of user processes which execute the protocol, an intruder process and a medium which carries all the messages.

As is pointed out in [Sch96], the medium and intruder can be rewritten as a single process $ENEMY$:

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

This is the description we shall use through this paper: $ENEMY = ENEMY(INIT)$, where $INIT$ does not contain N_A, N_B, K_a, K_b or K_{ab}.

Although this description looks simple, it is powerful enough to model all aspects of the Dolev-Yao model, in that it can block, duplicate, re-order or fake messages. All attacks involving these operations are therefore possible within the model.

4 Authentication

In the CSP traces model, properties are given as predicates on traces, and a process P satisfies a specification S if all of its traces satisfy S:

$$P \textbf{ sat } S \Leftrightarrow \forall\, tr \in traces(P).S$$

In the traces model, we say that P is a refinement of Q (written $(Q \sqsubseteq P)$ if $traces(Q) \subseteq traces(P)$, and from this definition it follows that

$$P \sqsubseteq Q \wedge P \textbf{ sat } S \Rightarrow Q \textbf{ sat } S$$

We use this message-oriented approach in defining authentication: a set of messages T $\mathsf{authenticates}$ a set of messages R if the receipt of a message in set T guarantees the previous transmission of a message in set R. As a predicate on traces, this is defined

$$T \;\mathsf{authenticates}\; R \quad = \quad tr \restriction R = \langle\rangle \Rightarrow tr \restriction T = \langle\rangle$$

If it is not possible for a trace tr to contain a message from the set T without also containing a message from the set R, then we can be sure that a message from set R was transmitted onto the network before T could be received.

In this paper, the property we choose to prove is that the message

$$rec(b, i, crypto(longterm(lt(b)), conc3(S, Ia, Nb)))$$

authenticates

$$\{trans(s, j, x.crypto(longterm(lt(b)), conc3(S, Ia, Nb)).y)\}$$

where S is an arbitrary session key. That is, if agent B receives, from anywhere, a message encrypted with his long-term key, and containing a session key S, a neighbour's identity and his original nonce challenge, he can be sure that that message originated from the server.

The following lemma is an immediate consequence of the definition.

Lemma 1

$$P \textbf{ sat } T \;\mathsf{authenticates}\; R \Leftrightarrow P \,|[\, R \,]|\, Stop \textbf{ sat } tr \restriction T = \langle\rangle$$

This follows from the fact that the process $P \,|[\, R \,]|\, Stop$ is unable to perform any events from the set R. Thus to prove that

$$P \textbf{ sat } T \;\mathsf{authenticates}\; R$$

it is sufficient to prove that $P \,|[\, R \,]|\, Stop$ **sat** $tr \restriction T = \langle\rangle$. This is the approach we will use in paper.

The CSP traces model has a sound and complete set of rules for proving that processes satisfy specifications, which could be used here, but we prefer to develop a set of rules specific to our application, which will enable us to reason at a more appropriate level of abstraction. Those used in this paper are given in Figure 2.

The soundness of the rules follows from the trace semantics of the operators, and the formal definition of T authenticates R. They have been proven in PVS [DS97a]. We may give informal justification of their soundness by considering that occurrence of an event from T is intended to provide evidence that some event from R previously occurred. Hence a process *fails* to satisfy T authenticates R only when some event from T occurs before some event from R.

Rule `auth.stop` is therefore sound because *Stop* cannot perform any events at all, and so cannot perform some T before some R.

Rule `auth.prefix.1` is sound because if the very first event a performed by $a \rightarrow P$ is an event from R, then it is not possible for an event from T to occur before an event from R. This is independent of the nature of the subsequent process P, which therefore has no restrictions placed on it by the rule—the rule is applicable for any process P.

Rule `auth.prefix.2` is most useful when the event a is not in R, since otherwise `auth.prefix.1` is applicable. In this case it states that if the first event is not in T, then occurrence of a is irrelevant to authentication of R by T, and such authentication is guaranteed for $a \rightarrow P$ whenever it is guaranteed for P.

Rule `auth.choice` states that if each branch of a choice guarantees the authentication property T authenticates R, then so does the entire choice—since whenever some event from T occurs, it must have been performed by one of the arms of the choice, and that choice must previously have performed some event from R.

Rule `auth.parallel` states that if a single component P of a parallel combination is able to guarantee that T authenticates R, and it is involved in all occurrences of events from T and R, then that is enough to ensure that the entire parallel combination $P \,|[\, A \,]|\, Q$ guarantees it: since P will not allow any event from T to occur before an event from R occurs. There are no restrictions on the rest of the system Q, so the rule holds for any process description Q.

Rule `auth.stop`

$$\frac{}{\mathit{Stop}\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}$$

Rule `auth.prefix.1`

$$\frac{}{a \rightarrow P\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R} \quad [\ a \in R\ ]$$

Rule `auth.prefix.2`

$$\frac{P\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}{a \rightarrow P\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R} \quad [\ a \notin T\ ]$$

Rule `auth.choice`

$$\frac{\forall\, j\,.\,V_j\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}{\Box_j\ V_j\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}$$

Rule `auth.parallel`

$$\frac{P\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}{P\ |[\ A\ ]|\ Q\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R} \quad [\ (R \cup T) \subseteq A\ ]$$

Rule `auth.interleaves`

$$\frac{\begin{array}{l} P\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R \\ Q\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R \end{array}}{P\ |||\ Q\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R}$$

Rule `auth.recursion`

$$\frac{\begin{array}{l} (\forall\, k.X_k\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R) \Rightarrow \\ \qquad (\forall\, k.F_k(\underline{X})\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R)) \end{array}}{\forall\, k.X(k)\ \mathbf{sat}\ T\ \mathsf{authenticates}\ R} \quad [\ \forall\, k.X_k \mathrel{\widehat{=}} F_k(\underline{X})\ ]$$

Figure 2: Proof rules for authentication

Rule `auth.interleaves` states that if both components of an interleaved combination can guarantee T $\mathsf{authenticates}$ R, then the combination itself can. This follows from the fact that if some event from T occurs, then it must have been performed by one of the component processes, which must have previously performed an event from R.

Finally, the rule `auth.recursion` for mutually recursive processes states that if the property T $\mathsf{authenticates}$ R is preserved by recursive calls—if each variable X_k **sat** T $\mathsf{authenticates}$ R then so does each function $F_k(\underline{X})$ applied to the variables—then the processes defined by the mutual recursion satisfy the property T $\mathsf{authenticates}$ R.

4.1 A key theorem

We obtain an extremely specialised theorem that applies to authentication properties on this specific description NET of the network. This theorem is at the heart of the proof strategy presented in this paper. It provides a sufficient list of conditions whose achievement guarantees that NET **sat** T `authenticates` R.

Theorem 1 $\rho : MESSAGE \to \mathbb{Z}$ is such that

C1: $\forall\, m \in INIT.\rho(m) > 0$

C2: $(\forall\, m' \in S.\rho(m') > 0)) \wedge S \vdash m \Rightarrow \rho(m) > 0$

C3: $\forall\, m \in T.\rho(m) \leq 0$

C4: $\forall\, i.(USER_i \,|[\, R \,]|\, Stop$ **sat** $\mathsf{maintains}\ \rho)$

then $NET \,|[\, R \,]|\, Stop$ **sat** $tr \upharpoonright T = \langle\rangle$

The rank function ρ is intended to have positive value on all messages which can be generated by some agent (including the enemy) during a run of the protocol, when all messages in the set R are prevented from occurring. The intention is to show that this restriction on R means that no event from T can occur, and hence by Lemma 1 that T $\mathsf{authenticates}$ R. Conditions $C1$ and $C2$ together mean that if the enemy only ever sees messages of positive rank, then he can only ever generate messages of positive rank.

Condition $C4$ states that the same is true for the users of the network (when restricted on R): they never output a message of non-positive rank unless they previously received such a message. The specification $\mathsf{maintains}\ \rho$ is defined as:

$$\begin{aligned}&\mathsf{maintains}\ \rho(tr) \mathrel{\widehat{=}} \\ &\qquad (\forall\, m \in (tr \Downarrow rec) : \rho(m) > 0) \Rightarrow (\forall\, m \in (tr \Downarrow trans) : \rho(m) > 0)\end{aligned}$$

If every message received on *rec* has positive rank, then so does every message sent out on *trans*.

The predicate "$\rho(m) > 0$" can therefore be seen as describing an invariant: at every stage of the protocol's execution when R is suppressed, it must hold of the next message. Since $C3$ states that it does not hold for any message in T, this means that no message in T can ever be generated.

The problem for any particular protocol, and a particular authentication property expressed in terms of R and T, is to find an appropriate rank function ρ which makes $C1$ to $C4$ all true, and to verify this fact.

5 Translating to PVS notation

In [DS97a], an embedding of CSP in PVS is presented, precisely for mechanising the proofs necessary with this approach. CSP traces are represented as lists, a pre-defined notion in PVS. Processes are described as sets of traces. The CSP operators are then defined as trace combinators. For example, the choice operator '$\Box$' returns the union of its two arguments.

Since a process P satisfies a predicate E iff all its traces satisfy E, a *satisfaction* operator '`|>`' can be defined, so that P `|>` E provided P is a subset of E.

The Dolev-Yao framework has already been translated into PVS [DS97a, BS97], so all that was required was to define the message space and the protocol agents.

The message space was defined as

```
message : DATATYPE WITH SUBTYPES nonkey, key
   BEGIN
    text     (x_text     : Text)               : text?  : nonkey
    nonce    (x_nonce    : Nonce)              : nonce? : nonkey
    user     (x_user     : Identity)           : user?  : nonkey
    conc     (x_conc, y_conc : message)        : conc?  : nonkey
    session  (x_session  : SessionKey)         : session?  : key
    longterm (x_longterm : LongTerm)           : longterm? : key
    code     (x_code : key, y_code : message) : code?  : nonkey
    hash     (x_hash : key, y_hash : message) : hash?  : nonkey
   END message
```

The message authentication code is defined by

```
 mac(k, m) : message =  conc(hash(k, m), m)
```

The enemy may deduce certain information from the messages it sees. This deductive ability, given by $\vdash$ in the CSP model, is transcribed into PVS by the `Gen` relation:

```
  Gen(S)(m) : INDUCTIVE bool =
0)       S(m)
1) OR (EXISTS m1, m2: Gen(S)(m1) AND Gen(S)(m2) AND m = conc(m1, m2))
2) OR (EXISTS m1: Gen(S)(conc(m1, m)))
3) OR (EXISTS m2: Gen(S)(conc(m, m2)))
4) OR (EXISTS m1, k: Gen(S)(m1) AND Gen(S)(k) AND m = crypto(k, m1))
5) OR (EXISTS k: Gen(S)(k) AND Gen(S)(crypto(k, m)))
6) OR (EXISTS m1, k: Gen(S)(m1) AND Gen(S)(k) AND m = hash(k, m1));
```

Any message already known to the enemy is considered to be part of the generated set (line 0). The enemy may concatenate messages, or split concatenated messages (lines 1-3). If it is in possession of a key and an arbitrary message, it may encrypt the message with the key. (line 4). Since all keys are symmetric, if it owns a key and a message encrypted with that key, it may decrypt the message (line 5). Finally, if it owns a key and an arbitrary message, it may form the hash of the message with respect to the key (line 6). The transitivity requirement is implicit, because `m1`, `m2` and `k` are quantified over `Gen(S)`.

With these in place, it now remains to prove that each of the protocol participants maintain rank. This means that if the CSP description of an individual participant is restricted, so it cannot transmit any messages from the set R, then it is unable to transmit the message T. The contrapositive of this says that if the message T is observed on the medium, it must have been preceded by an event from the set R.

We need to define an operator `crypto`, which will encrypt and decrypt messages.

```
crypto(k, m) : message =
 CASES m OF
  code(x, y) :
   CASES k OF
    longterm(i): IF x = longterm(i)  THEN y ELSE code(k, m) ENDIF,
    session(i): IF x = session(i) THEN y ELSE code(k, m) ENDIF
   ENDCASES
  ELSE code(k, m)
 ENDCASES
```

It applies the function `code`, returning the original message if the key has been applied twice to the same message.

In the following definitions, `lt` is an abbreviation for the function which returns the longterm key of a user `i`, and `sk(i,j)` returns the session key for `i` and `j`.

The first user is defined as:

```
userA : process[event] =
  Choice! skey :
    ( trans(a, b, mac(lt(a), conc4(Ia, Ib, Na, null))) >>
    (   rec(a, b, crypto(lt(a), conc3(skey, Ib, Na))) >>
           Stop[event]   )
```

The `Choice! skey` means that in the second part of the definition, agent A is prepared to accept an arbitrary session key, provided that it forms part of a certificate encrypted with its longterm key, and contains his original nonce.

The definition of `userB` is similar, except that it first waits for a message from `userA`, then uses that message instead of the placeholder . It also expects two messages, each containing a single key certificate, and does not pass anything on to the lower members of the chain. These changes were necessitated by the definition of the server.

```
userB : process[event] =
 Choice! ltk, Nx, m, skup, skdown, i, k :
  (rec(b, i, mac(ltk, conc4(user(i), Ib, Nx, m))) >>
   ( trans(b, k, mac(lt(b), conc4(Ib, user(k), Nb,
                  mac(ltk, conc4(user(i), Ib, Nx, m))))) >>
    ( rec(b, k, crypto(lt(b), conc3(skup, user(k), Nb))) >>
     ( rec(b, k, crypto(lt(b), conc3(skdown, user(i), Nb))) >>
            Stop[event] ))))
```

The definition of the server differs in some ways from its CSP definition. In the CSP definition, the server receives one message, and sends out one message. However, a definition of this form would require the definition of '*response*' to be incorporated into the definition of the server.. This would require significant extra complexity in the PVS coding. It proved easier to make use of the assumptions of the Dolev-Yao model.

Since the medium is entirely in the control of the enemy, who may re-order, redirect or kill messages arbitrarily, we do not need to define the server to recurse on a single message to produce all the certificates. The inner layers of the message have already been circulating in the medium, and we may therefore rely on the medium to pass these on to the server as appropriate.

This is not as radical an assumption as it may seem, and it introduces no further attacks on the protocol. The medium may already destroy any run of the protocol by refusing to pass on messages. But what we are interested in are *safety* properties: if a protocol run completes successfully, then we want to be sure that the session keys are uncompromised. Provided it is possible for a single run to complete successfully, we are not interested in any incomplete runs.

In this description, the server receives a message, which either has at least two levels of message authentication codes, or contains the placeholder null. The two parts of the definition result from the pattern matching that occurs on the first message. It the message contains at least two levels of message authentication codes, then the server prepares and sends the appropriate two key certificates. These are addressed direct to the intended recipient: since our enemy may arbitrarily redirect messages, nothing is gained by insisting that they pass through all members of the chain.

If the incoming message contains the placeholder null, then only one certificate is necessary.

```
Fs(X) : process[event] =
 (Choice! m2, Nix, Njx, i, j, k, l :
 (rec(s,l,mac(lt(j), conc4(user(j), user(k), Njx,
             mac(lt(i), conc4(user(i), user(j), Nix, m2))))) >>
 ( trans(s, j, crypto(lt(j), conc3(sk(j,k), user(k), Njx))) >>
 (  trans(s, j, crypto(lt(j), conc3(sk(j,i), user(i), Njx))) >>
         X))))
\/
 (Choice! Nix, i, j, l :
 ( rec(s, l, mac(lt(i), conc4(user(i), user(j), Nix, null))) >>
 (  trans(s, l, mac(lt(i), conc4(user(i), user(j), Nix, null))) >>
 (   Stop[event]))))

  server : process[event] = mu(Fs)
```

5.1 The authentication property

Recall that we wish to prove that, for any i and S, the message T

$$rec(b, i, encrypt(longterm(lt(b)), conc3(S, Ia, Nb)))$$

authenticates R

$$\{trans(s, j, encrypt(longterm(lt(b)), conc3(S, Ia, Nb)))\}$$

We now have to define a rank function, which must assign a rank of 1 or above to allow messages which may possibly circulate in the network, and a rank of 0 or below to all messages which may not circulate in the network.

The rank function we used is given in Figure 3. The rank of all text, nonces and user identities is one. All session keys apart from the one between A and B have rank one, and all longterm keys have rank one, apart from the ones belonging to A, B and the server. All hashed messages have a rank of one. Encrypted messages have the same rank as the message itself, unless it is encrypted with either A or B's longterm key. All messages encrypted with either A or B's longterm key have rank one greater than the message itself, except for the authenticating message.

Proving that each of the processes maintains rank is very straightforward. The proof consists mainly of PVS macro steps developed specifically for authentication protocols, and presented in [DS97a]. The run times (on a Sparc 5) to check the proofs once they were developed were: `userA` took 25 seconds, `userB` took 91 seconds and `server` took 223 seconds.

6 Incorrect Implementation

In [RS97], an attack on an implementation of the recursive authentication protocol is described. The implementation decision which leads to the attack is straightforward. The server computes the certificates as $K_{ab} \bigoplus \mathsf{Hash}_{K_a}\{Na\}$, where '$\bigoplus$' represents the bitwise XOR of two bit strings.

To see that this is insecure, note that (with three agents in the chain) the server returns certificates of the form

$$K_{ab} \bigoplus \mathsf{Hash}_{K_a}\{Na\}, K_{ab} \bigoplus \mathsf{Hash}_{K_b}\{Nb\},$$
$$K_{bc} \bigoplus \mathsf{Hash}_{K_b}\{Nb\}, K_{cs} \bigoplus \mathsf{Hash}_{K_c}\{Nc\},$$

Anyone in possession of these certificates (and they are all broadcast across the network) can compute xor'd pairs of session keys, as

$$K_{ab} \bigoplus \mathsf{Hash}_{K_b}\{Nb\} \bigoplus K_{bc} \bigoplus \mathsf{Hash}_{K_b}\{Nb\} = K_{ab} \bigoplus K_{bc}$$

Thus if the enemy knows one session key, he may compute all others.

```
  rho(m) : RECURSIVE int =
     CASES m OF
       text(z)          : 1,
       nonce(z)         : 1,
       user(z)          : 1,
       session(z)       : IF session(z) = session(sk(a, b)) THEN 0
                             ELSE 1 ENDIF,
       longterm(z)      : IF z = lt(a) OR z = lt(b) OR z = lt(s) THEN 0
                             ELSE 1 ENDIF,
       conc(z1, z2)     : min(rho(z1), rho(z2)),
       hash(q, z)       : 1,
       code(q, z)       : rank_code(q, z, rho(z))
     ENDCASES
   MEASURE size(m)

rank_code(q, m, n) : int =
     CASES q OF
        session(z1): n,
        longterm(j)  :
             IF    j=lt(a) THEN rank_lt_a(m, n)
             ELSIF j=lt(b) THEN rank_lt_b(m, n)
             ELSE n
             ENDIF
     ENDCASES

rank_lt_a(m, n) : int  =  n+1

rank_lt_b(m, n) : int  =
     IF m = conc3(session(sk(a,b)), Ia, Nb) THEN 0 ELSE n+1 ENDIF
```

Figure 3: The Rank Function

6.1 PVS analysis

Although we knew the flaw in this protocol before beginning the analysis, we proceeded with a mechanical analysis, to see where it broke down, and whether we could make any deductions from that.

In fact, the flaw revealed itself very quickly. The new generates function, which includes XOR, is given by:

```
Gen(S)(m) : INDUCTIVE bool =
   S(m)
 OR  (EXISTS m1, m2 : Gen(S)(m1) AND Gen(S)(m2) AND m = conc(m1, m2))
 OR  (EXISTS m1 : Gen(S)(conc(m1, m)))
 OR  (EXISTS m2 : Gen(S)(conc(m, m2)))
 OR  (EXISTS m1 : Gen(S)(m1) AND m = hash(m1))
 OR  (EXISTS m1, m2 : Gen(S)(m1) AND Gen(S)(m2) AND m = xor(m1, m2))
 OR  (EXISTS m1 : Gen(S)(m1) AND Gen(S)(xor(m1, m)))
 OR  (EXISTS m2 : Gen(S)(m2) AND Gen(S)(xor(m, m2)));
```

It is impossible to prove that the "blank" rank function

```
rho(m) : RECURSIVE int =
 CASES m OF
   text(z)      : 1,
   nonce(z)     : 1,
   user(z)      : 1,
   session(z)   : IF session(z) = session(sk(a, b)) THEN 0 ELSE 1 ENDIF,
   longterm(z) : IF z = lt(a) OR z = lt(b) THEN 0 ELSE 1 ENDIF,
   conc(z1, z2): min(rho(z1), rho(z2)),
   hash(z1)     : 1,
   xor(z1, z2) : 1
 ENDCASES
   MEASURE size(m)
```

is valid, in other words an attempted proof of

$$\forall S, m : positive(\rho, S) \wedge (S \mid -m) \Rightarrow \rho(m) > 0$$

fails. It requires a sublemma:

$$\forall m1, m2 : \rho(m1) > 0 \wedge \rho((m1 \bigoplus m2)) > 0 \Rightarrow \rho(m2) > 0)$$

and the counter-example is that $\rho(s_{ab}) \leq 0$, since it is secret, but the enemy may know s_{bc}, since he may be masquerading as agent C, and $s_{ab} \bigoplus s_{bc}$ is also known, since it circulates in the network, so the proof of the sublemma fails.

Other rank functions could be tried, in which case the proof would fail at some other stage.

6.2 Corrected Implementation

The corrected implementation proposed in [RS97] is a very simple extension of the incorrect version. They suggest that the server return certificates of the form

$$K_{ab} \bigoplus \mathsf{Hash}_{Kb}\{Nb, A\}, K_{bc} \bigoplus \mathsf{Hash}_{Kb}\{Nb, C\}$$

which does indeed provide secure session keys between pairs of honest agents. This has now been proven for the most general case, when A, B and the server are honest.

7 Dealing with failed proofs

One of the less intuitive parts of the proof method outlined above is the rank function. It is not easy to tell at a glance whether a particular rank function will work or not, and if a proof fails it is not necessarily obvious whether this is because the protocol is flawed, or because the rank function was inappropriate. To some extent, improvements on a flawed rank function may be deduced by considering the PVS output. After applying the macro steps, we can reduce nontrivial sequents to their component parts (using `grind`), which gives us a list of consequents and antecedents. The antecedents originate essentially from the information that the rank function provides about messages which have already been observed in the network.

If none of the consequents follow from the antecedents [1], then it is sometimes possible to deduce a strengthening of the rank function by observing the consequents. Further protocol verification attempts are required in order to develop heuristics for this.

[1] PVS requires only that one consequent be proved, in order to prove the sequent

References

[BO97] J. Bull and D. J. Otway. The Authentication Protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, DRA, Feb 1997.

[BS97] J. W. Bryans and S. A. Schneider. Mechanical Verification of the full Needham-Schroeder Public-Key Protocol. Technical Report CSD-TR-97-11, Royal Holloway, University of London, April 1997.

[DS97a] B. Dutertre and S. A. Schneider. Embedding CSP in PVS. An application to Authentication Protocols. Technical Report CSD-TR-97-12, Royal Holloway, University of London, April 1997.

[DS97b] B. Dutertre and S. A. Schneider. Using a PVS Embedding of CSP to verify Authentication Protocols. In *Proceedings of TPHOLS*, 1997. To appear.

[DY83] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

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

[Pau97] L. Paulson. Mechanized proofs for a recursive authentication protocol. unpublished, 1997.

[RS97] P. Y. A. Ryan and S. A. Schneider. An Attack on a Recursive Authentication Protocol: A cautionary tale. DRA report, May 1997.

[Sch96] S. A. Schneider. Using CSP for protocol analysis: the Needham-Schroeder Public-Key Protocol. Technical Report CSD-TR-96-14, Royal Holloway, University of London, October 1996.

[SOR93] N. Shankar, S. Owre, and J. M. Rushby. *The PVS Proof Checker: A Reference Manual (Draft).* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993.