Verifying authentication protocols with CSP

Steve Schneider
Department of Computer Science
Royal Holloway, University of London, Egham Surrey TW20 0EX, UK
steve@dcs.rhbnc.ac.uk

Abstract

This paper presents a general approach for analysis and verification of authentication properties in the language of Communicating Sequential Processes (CSP). It is illustrated by an examination of the Needham-Schroeder public-key protocol. The contribution of this paper is to develop a specific theory appropriate to the analysis of authentication protocols, built on top of the general CSP semantic framework. This approach aims to combine the ability to express such protocols in a natural and precise way with the facility to reason formally about the properties they exhibit.

1 Introduction

Authentication comes in a number of flavours. For example, Gollmann [6] has identified four different varieties of authentication, which raises the question for any particular authentication protocol as to which kind of authentication the protocol was designed for, and which kinds it actually provides.

The aim of the CSP approach is to reduce questions about security protocols and properties to questions concerning whether CSP processes satisfy particular CSP specifications. This approach forces the separation of properties and protocols, and allows discussion of what is meant by particular kinds of security property independently of the protocols that are intended to achieve them. The formal analysis will then be entirely within the CSP framework which allows the possibility of verification of protocols with respect to the CSP properties.

The general CSP framework proposed by this author has been described in [20]. That paper is concerned with the theoretical foundations. It offers definitions of security properties, including authentication. This means that since a CSP description of a protocol has a precisely defined semantics it is a precise mathematical question as to whether the protocol meets the property or not. However, the practicalities of how such a verification might be carried out were not addressed; that is the purpose of this paper.

The approach taken here is firstly to express the protocol in CSP. The authentication property we consider states that if some events R in the system are restricted, then other events T should not occur. We establish this by defining a suitable predicate on messages which shows that only particular messages can circulate in the restricted system, and hence that messages from T are not possible.

CSP is particularly suitable for describing protocols close to the level we think of them. In other contexts the argument for authentication essentially amounts to an unreachability analysis or a proof that a particular word is not in a language. The strength of this approach is that there is a formal link between the argument and a natural description of the protocol. Formalisation of the protocol into CSP also exposes issues and forces design decisions that may not have been explicit in the original abstract protocol description. The formalisation of the required authentication property likewise forces consideration of what, precisely, is meant by authentication. It is useful to know which precise (CSP) properties the protocol does indeed guarantee, and which it does not.

One of the strengths of CSP is the ease with which specialised theories can be constructed on top of the semantic models. This allows particular specification statements to be defined in terms of the standard semantics, and new proof rules appropriate to these specifications to be provided. This approach is taken here, where we specify and reason about authentication properties, and also about agents' inability to generate particular messages. Although standard proof rules would support the verification (since they are sound

and complete), it is preferable to develop a specialised theory since it provides an appropriate level of abstraction for supporting the kind of reasoning we require. For reasons of space, this paper will not present details of the CSP proof system and CSP algebraic identities which underpin the results. They are given in a fuller version [21] of this paper.

2 The Needham-Schroeder Public-Key Protocol

The Needham-Schroeder public-key protocol aims to achieve authentication of each of its participants to the other. It makes use of public key cryptography, where each participant has a public key which is available to everybody, and a private key known to no-one else. Any message encrypted with a public key requires the corresponding secret key in order to decrypt it.

The original version of the protocol [15] assumed that the participants did not know each other's public key, and so each engaged in a message exchange with a trusted server in order to obtain it. If the participants do begin by knowing the public keys, then the protocol may be distilled to three messages (as described in [9]) as below. We will refer to it in this paper as NS1.

$$\begin{array}{lcl} A \rightarrow B & : & p_B(n_A.A) \\ B \rightarrow A & : & p_A(n_A.n_B) \\ A \rightarrow B & : & p_B(n_B) \end{array}$$

The participants of the protocol are A and B, their public keys are p_A and p_B respectively, and each generates one nonce, n_A and n_B respectively.

An informal explanation of the design of the protocol might run as follows: A sends a nonce n_A to B, encrypted with B's public key. When A later receives a message containing the nonce n_A, she knows that B must have generated it, since only B could decrypt the first message and thus obtain the nonce n_A, so B's identity is authenticated to A. Similarly, B sent out a nonce encrypted with A's public key, so the final message received by B, containing the nonce n_B, must have been generated by A because only A could have decrypted the message that B sent out. Thus A is authenticated to B, and the two participants know that they are talking to each other.

In [9], Lowe describes an unexpected execution of the protocol, discovered using model-checking techniques on a CSP description of a network running the protocol. In this run, B receives 'authentication' that he is talking to A, but A is in fact talking to an intruder I, who she believes to be honest. The execution is described as follows, where $I(A)$ denotes the intruder I masquerading as A (in other words, generating messages which appear to have been sent from A, and intercepting messages destined for A).

$$\begin{array}{lcl} A \rightarrow I & : & p_i(n_A.A) \\ I(A) \rightarrow B & : & p_B(n_A.A) \\ B \rightarrow I(A) & : & p_A(n_A.n_B) \\ I \rightarrow A & : & p_A(n_A.n_B) \\ A \rightarrow I & : & p_i(n_B) \\ I(A) \rightarrow B & : & p_B(n_B) \end{array}$$

B's response to the nonce challenge is to A, but there is nothing in that message to indicate that the response indeed originated from B. The intruder I is therefore able to pass it off as a nonce challenge that I has issued, and so I receives the response from A, which is then passed to B.

Authentication normally requires that each agent must at least assume that the other is honest, and will correctly follow the steps of the protocol. Note that in this attack A is indeed acting honestly, though her judgement concerning I's honesty is questionable.

Lowe's fix

Lowe observed that the above sequence of messages can be prevented by including the identity of the responder in the second message to identify its originator, and ensuring that this is checked. The resulting protocol is described below. We will refer to it in this paper as NS2.

$$\begin{array}{lcl} A \rightarrow B & : & p_B(n_A.A) \\ B \rightarrow A & : & p_A(n_A.n_B.B) \\ A \rightarrow B & : & p_B(n_B) \end{array}$$

Lowe observed that the model-checking techniques used to find the earlier flaw found no flaw with this protocol. Furthermore, he offered a proof that any attack on the protocol as a whole would also be an attack on the necessarily finite CSP description that had been model-checked. Since no such attack had been found, there could be no attack on the protocol. The proof considered the nature of any possible sequence of messages embodying an attack.

This paper aims at a more direct method of verification, in which a CSP description of the protocol in a hostile environment is shown to meet particular authentication properties. The proof provides insight into why the design of the protocol is correct.

The protocol NS2 is used as an example running through the paper. For the purposes of illustrating the approach, we first consider a single run of the protocol where both participants know their roles in advance; having worked through the approach with this example, we will finally show how the approach generalises to multiple, interleaved runs of the protocol between many different agents.

Section 3 introduces the aspects of the language and theory of Communicating Sequential Processes (CSP) appropriate to this paper. Section 4 constructs a CSP model for analysing authentication protocols, and describes NS2 in CSP. Section 5 introduces the CSP approach to defining authentication properties, and considers the properties that might be required of NS2. Section 6 describes a theory for verifying authentication protocols against particular authentication requirements. Section 7 carries out the verification for NS2. Section 8 discusses the generalisation of the analysis to consider the case of multiple, concurrent runs of the protocol. Section 9 briefly discusses how the algebraic properties of cryptomechanisms might be incorporated within the analysis. We end with a discussion in Section 10.

3 CSP

CSP is an abstract language designed specifically for the description of communication patterns of concurrent system components that interact through message passing. It is underpinned by a theory which supports analysis of systems described in CSP. It is therefore well suited to the description and analysis of network protocols: protocols can be described within CSP, as can the relevant aspects of the network. Their interactions can be investigated, and certain aspects of their behaviour can be verified through use of the theory. This section introduces the notation and ideas used in this paper. In particular, only the traces model for CSP is used here. For a fuller introduction to the language the reader is referred to [7].

Events

Systems are modelled in terms of the events that they can perform. The set of all possible events (fixed at the beginning of the analysis) is denoted Σ. Events may be atomic in structure or may consist of a number of distinct components. For example, an event $put.5$ consists of two parts: a channel name put, and a data value 5. An example of events used in this paper are those of the form $c.i.j.m$ consisting of a channel c, a source i, a destination j and a message m. If M and N are sets of messages, then $M.N$ will be the set of messages $\{m.n \mid m \in M \wedge n \in N\}$. If m is a single message then we elide the set brackets and define $m.N$ to be $\{m\}.N$. Thus for example the set of events $i.N.m = \{i.n.m \mid n \in N\}$. A channel c is said to be of type M if any message $c.m \in \Sigma$ has that $m \in M$.

Processes

Processes are the components of systems. They are the entities that are described using CSP, and they are described in terms of the possible events that they may engage in. The process $Stop$ is the process that can engage in no events at all; it is equivalent to deadlock. The output $c!v \rightarrow P$ is able initially to perform only $c.v$, the output of v on channel c, after which it behaves as P. The input $c?x : T \rightarrow P(x)$ can accept any input x of type T along channel c, following which it behaves as $P(x)$. Its first event will be any event of the form $c.t$ where $t \in T$. The process $P \,\Box\, Q$ (pronounced 'P choice Q') can behave either as P or as Q: its possible communications are those of P and those of Q. An indexed form of choice $\Box_{i \in I} P_i$ is able to behave as any of its arguments P_i.

Processes may also be composed in parallel. If D is a set of events then the process $P \,|[\, D \,]|\, Q$ behaves as P and Q acting concurrently, with the requirement that they have to synchronise on any event in the synchronisation set D; events not in D may be performed by either process independently of the other. A special form of parallel operator in which the two components do not interact on any events is $P \,|||\, Q$ which is equivalent to $P \,|[\{\}]|\, Q$.

Processes may be recursively defined by means of equational definitions. Process names must appear on the left hand side of such definitions, and CSP expressions which may include those names appear on the right hand side. For example, the definition

$$LIGHT = on \rightarrow off \rightarrow LIGHT$$

defines a process $LIGHT$ whose only possible behaviour is to perform on and off alternately.

Mutually recursive processes may also be defined, where a (possibly infinite) collection of process names $X(k)$ appear on the left hand side of definitions, and CSP expressions $F(k)$ involving any of those names appear on the right. For example, the set of definitions

$$COUNT(0) \;\; \widehat{=} \;\; up \rightarrow COUNT(1)$$

$$COUNT(n+1) \;\widehat{=}\; (up \rightarrow COUNT(n+2)) \;\Box\; down \rightarrow COUNT(n)$$

define a collection of processes; $COUNT(0)$ can do any number of *up* and *down* events, but can never do more *down*s than *up*s.

For a full discussion of single and mutually recursive process definitions, see [3].

Traces

The semantics of a process P is defined to be the set of sequences of events ($traces(P)$) that it may possibly perform. Examples of traces include $\langle\rangle$ (the empty trace, which is possible for any process) and $\langle on, off, on\rangle$ which is a possible trace of $LIGHT$. The *traces model* for CSP [7] gives a formal definition of the set of traces associated with each CSP process. Every process is associated with some non-empty *prefix-closed* set of traces: if $tr_1 \frown tr_2$ is a possible trace, then so is tr_1.

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.

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.

Specification

Specifications are given as predicates on traces, and a process P satisfies a specification S if all of its traces satisfy S:

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

For example, the specification which states that event a must occur before event b might be captured with the predicate

$$\forall\, tr_1, tr_2.tr = tr_1 \frown \langle b\rangle \frown tr_2 \Rightarrow tr_1 \upharpoonright a \neq \langle\rangle$$

The fact that $traces(P)$ is always prefix-closed allows a simpler specification:

$$tr \upharpoonright b \neq \langle\rangle \Rightarrow tr \upharpoonright a \neq \langle\rangle$$

This states that if b occurs somewhere in the trace, then so does a. Since a process satisfying this specification must have *all* of its traces meeting the predicate, this means that a must occur before b, since if b occurred first in some trace tr then a prefix of tr containing b but not a would fail to meet the predicate.

4 The CSP model

The approach taken to modelling and analysing security protocols is to provide a CSP description of the Dolev-Yao model [5]. Here it is assumed that the communications medium is entirely under the control of the enemy, which can block, re-address, duplicate, and fake messages. We will define a 'generates' relation $\vdash$ which describes when new messages may be derived from existing ones: $S \vdash m$ means that knowledge of all the messages in S is sufficient to produce m. This will be used to capture the enemy's ability to fake messages. In [20] the roles of the passive medium and of the active enemy were described using distinct CSP processes which enabled the capabilities of the enemy to be separately described. For the purposes of this paper it is preferable to describe the combination of the enemy and the medium as a single CSP process $ENEMY$, since this makes for an easier analysis.

There is a set $USER$ consisting of the names of all the agents which use the network. For each $i \in USER$ we associate a process $USER_i$ which describes how user i behaves. Each process $USER_i$ communicates with $ENEMY$ by means of a channel $trans.i$ on which it transmits messages, and a channel $rec.i$ on which it receives messages. Thus we have $\sigma(USER_i) \subseteq trans.i \cup rec.i$.

The resulting network is then described as follows:

$$NET \;=\; (|||_{j \in USER}\, USER_j) \,[|\, trans, rec \,|]\, ENEMY$$

This network is pictured in Figure 1.

In the analysis of a protocol we might consider A and B as the two parties involved in the protocol, and $USER_A$ and $USER_B$ will describe the respective roles that they play.

If there are actually no other users connected to the system, then we can have $USER_i = Stop$ for each $i \neq A, B$ and we obtain

$$NET \;=\; (USER_A \;|||\; USER_B) \,[|\, trans, rec \,|]\, ENEMY$$

We retain the names of other users in the set $USER$ to retain the potential of the $ENEMY$ to masquerade as a different user.

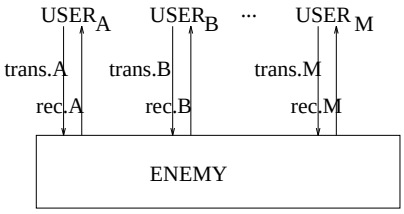

Figure 1. CSP model of the network

The channels *trans* and *rec* are of type $USER.USER.MESSAGE$. A message $trans.i.j.m$ should be thought of as node i sending a message m with destination j. Thus i is the source, j the destination, and m the message. The message space $MESSAGE$ will generally be defined as an abstract data type.

Message space

The message space we use for analysis of this protocol is as follows:

$$RAW ::= USER \mid TEXT \mid NONCE \mid KEY$$

$$MESSAGE ::= RAW \mid KEY(MESSAGE) \mid MESSAGE.MESSAGE$$

In fact for this example using public key cryptography, the space KEY will split into public keys $PUBLIC$ and secret keys $SECRET$, one of each for each user in $USER$:

$$KEY ::= PUBLIC \mid SECRET$$

We have rules concerning the way messages may be generated from existing ones. This set of rules defines the generates relation $\vdash$.

A1 If $m \in S$ then $S \vdash m$

A2 If $S \vdash m$ and $S \subseteq S'$ then $S' \vdash m$

A3 If $S \vdash m_i$ for each $m_i \in S'$ and $S' \vdash m$ then $S \vdash m$

M1 $S \vdash m \wedge S \vdash k \Rightarrow S \vdash k(m)$

M2 $S \vdash m_1 \wedge S \vdash m_2 \Leftrightarrow S \vdash m_1.m_2$

K1 $\{p_i(s_i(m))\} \vdash m$

K2 $\{s_i(p_i(m))\} \vdash m$

Description of *ENEMY*

A natural CSP description of the Dolev-Yao model separates out an essentially passive medium $MEDIUM$ from an intruder $INTRUDER$ which is able to read messages from the medium (via channel *leak*) and to manipulate it by removing and adding messages (via channels *kill* and *add*). This approach was taken in [20], and allows explicit modeling of the enemy actions. However, for verification purposes an equivalent description will be easier to use in proofs:

$$\begin{aligned} &ENEMY(S) = \qquad (1)\\ &\quad trans?i?j?m \rightarrow ENEMY(S \cup \{m\})\\ &\quad \Box \,\Box_{i,j \in USER,\, S \vdash m}\, rec.i!j!m \rightarrow ENEMY(S) \end{aligned}$$

This process has exactly the same traces as the combination of $MEDIUM$ and $INTRUDER$. It reads all messages output by any of the users on any of the channels *trans*, and it can pass on any message which it can generate to any user (which allows for honest message passing, redirecting messages, replaying messages, and inventing new messages). Furthermore, since all of these options are possibilities rather than imperatives, it is also possible that messages will be blocked simply by the enemy not exercising the option of passing them on.

The set S contains messages which the enemy knows about: every time a new message is put out by some user, the set S is augmented. There will be a set $INIT$ consisting of the enemy's initial knowledge: this may contain items such as users' names, public keys, and some nonces which the enemy may use.

$$ENEMY = ENEMY(INIT)$$

The agents implementing the protocols place restrictions on the messages that may be passed on *trans*, which in turn restricts the possibilities for messages being passed on *rec*.

We are already in a position to prove that all messages passed on *rec* must be generable from the initial set $INIT$ together with the messages input on *trans*:

Theorem 4.1

$ENEMY$ **sat** $(INIT \cup (tr \Downarrow trans)) \vdash tr \Downarrow rec$

□

Protocol participants

The agents A and B that are the participants in the protocol are modelled as $USER_A$ and $USER_B$, consisting of CSP implementations of the two halves of the

protocol. Obviously these will vary depending on the protocol being modelled. More generally, if there are other participants (such as trusted third parties) then their activity will also be described as CSP processes.

For the purposes of illustrating the CSP approach, we simplify the analysis by considering up to Section 8 only the case where A is the initiator (and knows that she is), and B is the responder (and knows that he is). This description therefore does not allow for attacks where both parties act as initiators, or as senders. This assumption will be relaxed in Section 8, where each party can independently play either role.

User A is described by the process $USER_A$:

$$\begin{aligned} USER_A \quad = \quad & \Box_{i \in USER}\ trans.A!i!p_i(n_A.A) \rightarrow \\ & rec.A.i?p_A(n_A.x.i) \rightarrow \\ & trans.A!i!p_i(x) \rightarrow Stop \end{aligned}$$

This process allows a user i to be chosen, and the first communication $trans.A!i!p_i(n_A.A)$ sends out the encrypted nonce challenge $p_i(n_A.A)$, on A's transmit channel $trans.A$, with intended destination i. Since the enemy can interfere with the source and destination fields of messages, the use of the third filed in messages is simply to track information concerning the intentions or expectations of the participants in the protocol. In this case, it tracks the fact that user A believes she is running the protocol with i.

The second communication is one of the form $rec.A.i?p_A(n_A.x.i)$. This appears on A's receive channel $rec.A$, apparently from source i. The form of the message $p_A(n_A.x.i)$ indicates that the process will only accept an input which is encrypted by p_A, whose first nonce is n_A, and whose included user identity is i; but that x can be any nonce. The use of pattern matching on message input corresponds to the assumption that any message that fails to match the pattern would be ignored, though we are modelling this as blocking receipt rather than accepting and throwing it away: the $USER_i$ processes simply do not engage in any message which does not match the pattern. In practice, this is unlikely to be achievable, especially in cases where an agent must decrypt a message before finding out if it is of a particular form. However, it does not affect the ability of the enemy to attack protocols described in this way.

Typing of messages is also implicit in pattern matching. The type of the channel determines the range of possible messages that might match the pattern. The range of permissible inputs for $rec.A.i.p_A(n_A.x.i)$ depends on the type of x. A correct run of the protocol would have x as a nonce, but without the ability to type messages the input could accept an arbitrary message for x. For the purposes of this paper, we will assume that inputs defined by pattern matching must also conform to the expected type. This amounts to assuming that it is not possible for a message of one type to be mistaken for a message of another type. This typing assumption can be relaxed within our framework, though it makes the verification more difficult since there are more general cases to consider.

The final communication $trans.A!i!p_i(x)$ consists of the response to the nonce challenge x.

The behaviour of the other protocol participant is described in CSP by the process $USER_B$:

$$\begin{aligned} USER_B \quad = \quad & rec.B?j?p_B(y.j) \rightarrow \\ & trans.B!j!p_j(y.n_B.B) \rightarrow \\ & rec.B.j.p_B(n_B) \rightarrow Stop \end{aligned}$$

The first message $rec.B?j?p_B(y.j)$ allows input of any message along B's receive channel which is encrypted with B's secret key. The apparent source j should match the source j given in the message; and y can be any nonce. Pattern matching is used again here.

The second message $trans.B!j!p_j(y.n_B.B)$ describes B's response to the first message. The required response is the nonce y together with B's nonce challenge n_B, and B's name B, all encrypted with the public key of the originator of the first message.

Finally, the third message that is expected is a response with apparent source j to the nonce challenge n_B. The only possibility is $p_B(n_B)$, so all other inputs will be blocked/ignored.

5 Authentication properties

A message-oriented approach to authentication is discussed in [20]. Authentication considers a set of messages T to authenticate another set of messages R if occurrence of some element of T must have been accompanied in P by occurrence of some element of R. The sets T and R are taken to be disjoint. The trace specification may be captured as follows

Definition 5.1

$$T \ \textbf{authenticates}\ R \quad = \quad tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle$$

□

The following lemmas are an immediate consequence of the definition:

Lemma 5.2

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

□

Lemma 5.3 if P **sat** T `authenticates` R and $R \subseteq R'$ then P **sat** T `authenticates` R' □

Flavours of origin authentication property

The treatment of authentication in terms of sets of messages allows for extremely fine-grained distinctions to be made between different flavours of authentication property.

Even if we restrict the authenticating message to be $USER_B$'s last one $rec.B.A.p_B(n_B)$, then there are a number of possibilities as to what this message might authenticate:

1. Authenticating "$trans.A.B.p_B(n_A.A)$" corresponds to the requirement that occurrence of B's final message guarantees that A initiated the protocol run with B, with the nonce n_A.
2. Authenticating "$trans.A.B.p_B(NONCE.A)$" corresponds to the requirement that on occurrence of B's last message it is guaranteed that A initiated the protocol run with B, but possibly with a different nonce. Lemma 5.3 yields that this is implied by property (1).
3. "$\{trans.A.j.p_j(NONCE.A) \mid j \in USER\}$" being authenticated corresponds to the requirement that on occurrence of B's last message it is guaranteed that A initiated a protocol run, but possibly with a different user and nonce. Lemma 5.3 yields that this is implied by property (2), and hence by property (1).
4. Authenticating "$rec.A.USER.p_A(NONCE.n_B.B)$" confirms that A received B's nonce challenge, and that A initiated the run with B (as shown by the name contained in the second message).
5. "$rec.A.USER.p_A(NONCE.n_B.USER)$" being authenticated confirms only that A received B's nonce challenge. It does not confirm that A initiated the run with B.
6. Authenticating "$trans.A.B.p_B(n_B)$" confirms that A responded to B's nonce challenge.
7. Authenticating "$trans.A.B.p_B(NONCE)$" authenticates that A responded to some challenge believed to come from B, but not necessarily with the nonce that B issued. It follows from Lemma 5.3 that this is implied by property (6).
8. Authenticating "$\{trans.A.j.p_j(n_B) \mid j \in USER\}$" authenticates that A responded to B's nonce challenge, but does not necessarily associate it with B. It follows from Lemma 5.3 that this is also implied by property (6).
9. "$\{trans.A.j.p_j(NONCE) \mid j \in USER\}$" being authenticated confirms only that A responded to some nonce challenge—in other words, it verifies that A is live. It follows from Lemma 5.3 that this is implied by property (8) and hence by property (6).

Property 6 is one of the strongest properties, and is the one that is most appropriate to prove for user B: that if he reaches the end of the protocol, then A also reached the end of the protocol, and responded to the nonce challenge n_B, and believed she was talking to B. The original protocol NS1 fails to meet this property, though it does meet the weaker Property 8 (a proof is provided in [21]).

6 Verification

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 6.1 If I is a predicate on messages such that

C1: $\forall\, m \in INIT.I(m)$

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

C3: $\forall\, m \in T.\neg I(m)$

C4: $\forall\, i.(USER_i \,|[\, R \,]|\, Stop \textbf{ sat } \texttt{maintains } I)$

then $NET \,|[\, R \,]|\, Stop \textbf{ sat } tr \upharpoonright T = \langle\rangle$ □

The predicate I is intended to hold of 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. Conditions $C1$ and $C2$ together mean that if the enemy only ever sees messages which satisfy I, then he can only ever generate messages which satisfy I.

Condition $C4$ states that the same is true for the users of the network (when restricted on R): they never output a message that does not satisfy I unless they previously received such a message. The specification `maintains` I is defined as:

$$\begin{aligned}&\texttt{maintains } I(tr) \mathrel{\widehat{=}} \\ &\quad (\forall\, m \in (tr \Downarrow rec) : I(m)) \\ &\qquad \Rightarrow (\forall\, m \in (tr \Downarrow trans) : I(m))\end{aligned}$$

If every message received on *rec* meets I, then every message sent out on *trans* also meets I.

The predicate I can therefore be seen as describing an invariant: at every stage of the protocol's execution when R is restricted, I must hold of the next message. Since $C3$ states that I does not hold for any message in T, this means that no message in T can ever be generated if R is restricted, which in turn means that T authenticates R by Lemma 5.1.

The problem for any particular protocol, and a particular authentication property expressed in terms of R and T, is to find an appropriate invariant I which makes $C1$ to $C4$ all true.

Rank functions

A previous approach [21] introduced a *rank function* ρ on messages which captured their level of encryption. Only messages of positive rank were to circulate in the system. In this setting, the invariant $I(m)$ was taken to be $\rho(m) > 0$. The rank function was used to define the invariant I below, but Theorem 6.1 is here presented in terms of I rather than ρ in order to provide a sharper description of the role played by the predicate. In many cases it may well be easier to define I in terms of a rank function, though this requires further investigation.

7 Verification of Property 6

The authentication property 6 states that "$rec.B.A.p_B(n_B)$ **authenticates** $trans.A.B.p_B(n_B)$", which confirms that A responded to B's nonce challenge.

An appropriate predicate for verification of NS2 with respect to this property is the following. It is dependent on the descriptions $USER_A$ and $USER_B$, and on the property to be verified.

It has been defined by structural induction on the definition of the message space.

$$\begin{aligned}&I(m) \Leftrightarrow \\ &\quad m \in USER \\ &\quad \vee\ m \in TEXT \\ &\quad \vee\ m \in NONCE \wedge m \neq n_B \\ &\quad \vee\ m \in SECRET \wedge m \neq s_A \wedge m \neq s_B \\ &\quad \vee\ m \in PUBLIC \\ &\quad \vee\ \exists\, m' : MESSAGE, k : PUBLIC. \\ &\qquad m = k(m') \wedge \\ &\qquad ((k = p_A \wedge m' \in NONCE.n_B.B) \vee I(m')) \\ &\quad \vee\ \exists\, m' : MESSAGE, k : SECRET. \\ &\qquad m = k(m') \wedge \\ &\qquad \neg(k = s_A \wedge m' \in p_A(NONCE.n_B.B)) \wedge I(m') \\ &\quad \vee\ \exists\, m_1, m_2 : MESSAGE. \\ &\qquad m = m_1.m_2 \wedge I(m_1) \wedge I(m_2)\end{aligned}$$

We must show that $C1$–$C4$ of Theorem 6.1 are satisfied by this invariant, in order for the theorem to be applicable.

C1

The condition $C1$ really amounts to an assumption on the initial information available to the enemy. The requirement that $\forall\, m \in INIT.I(m)$ states that the enemy is unable to generate B's nonce, and that he is not in possession of either A's or B's secret key. This does not require a proof, but rather formalises the assumption we are making.

C2

Condition $C2 : (\forall\, m' \in S.I(m')) \wedge S \vdash m \Rightarrow I(m)$ is a requirement on the interaction between the relation $\vdash$ and the predicate. It may be proven inductively by considering each of the clauses defining $\vdash$ in turn.

For example, consider the clause $M1 : S \vdash m \wedge S \vdash k \Rightarrow S \vdash k(m)$. If $\forall\, m' \in S.I(m')$ then by the inductive assumption $C2$ we have $I(m)$ and $I(k)$. There are two possibilities for k: a public key, or a secret key. If k is

a public key p_j then the fact that $I(m)$ holds means that $I(k(m))$ holds as required. If k is a secret key s_j then $j \neq A$ (since $I(s_j)$ but $\neg I(s_A)$), and so again $I(m)$ implies $I(s_j(m))$. This establishes the inductive step for $M1$.

C3

Condition $C3 : \forall\, m \in T.\neg I(t)$ must be checked in this case for the single message that constitutes T—$p_B(n_B)$. Since $\neg I(n_B)$, it follows from the definition of I that $\neg I(p_B(n_B))$.

C4

Finally, condition $C4$ must be checked for $USER_A$ and $USER_B$. This means that neither $USER_A$ nor $USER_B$, when restricted on $trans.A.B.p_B(n_B)$, can introduce any messages which do not satisfy I. We use $RUSER_A$ and $RUSER_B$ to refer to $USER_A$ and $USER_B$ respectively restricted on this event.

We first consider $USER_A$. The rules of CSP allow the restricted process $RUSER_A$ to be rewritten to a sequential process as follows:

$$
\begin{array}{l}
RUSER_A = \\
\quad USER_A \,|[\, trans.A.B.p_B(n_B) \,]|\, Stop = \\
\quad \Box_{i \in USER} \\
\qquad trans.A!i!p_i(n_A.A) \rightarrow \\
\qquad rec.A.i?p_A(n_A.x.i) \rightarrow \\
\qquad \left\{ \begin{array}{ll} Stop & \text{if } i = b \wedge x = n_B \\ trans.A!i!p_i(x) \rightarrow Stop & \text{if } i \neq b \vee x \neq n_B \end{array} \right.
\end{array}
$$

There are proof rules for CSP (presented in [21]) which allow a formal proof that this process satisfies **maintains** I. Here we will present an informal argument, by considering the messages that $RUSER_A$ sends and receives. We must prove that if every message received by the process meets the predicate I, then every message sent out by it also does so. We proceed by establishing that no message sent out by $RUSER_A$ violates this requirement.

The first message that is sent out is $p_i(n_A.A)$ for some i. The definition of I yields that $I(p_i(n_A.A))$, so the first message sent out by $RUSER_A$ does not violate our requirement.

A message $p_A(n_A.x.i)$ is then received. If this does not satisfy I, then $RUSER_A$ cannot violate the requirement, since the requirement is appropriate only when $USER_A$ receives messages that all meet I. Hence we need only consider the case where $I(p_A(n_A.x.i))$. From the definition of I in this case, either $n_A.x.i \in NONCE.n_B.B$, or $I(n_A.x.i)$. This disjunction implies that either $x = n_B$ and $i = b$, or else $I(x)$. In the first case, the restricted $USER_A$ sends out no further messages, so cannot violate the requirement. In the second case, the restricted user sends out $p_i(x)$, and since $I(x)$, it follows that $I(p_i(x))$, so again the requirement is not violated. This completes the proof.

We now turn our attention to $USER_B$.

The restricting event $trans.A.B.p_B(n_B)$ does not intersect with the events that $USER_B$ performs, so $USER_B$ itself is the process to consider to establish $C4$ for B. We are concerned with runs of $USER_B$ which lead to the authenticating event $rec.B.A.p_B(n_B)$, so we are only concerned with those runs which have A as the apparent origin of the first message—any other agent identity in the first message cannot lead to the occurrence of $rec.B.A.p_B(n_B)$.

The definition we are thus concerned with is

$$
\begin{array}{lll}
USER_{ba} & = & rec.B.A?p_B(y.A) \rightarrow \\
 & & \quad trans.B!a!p_A(y.n_B.B) \rightarrow \\
 & & \quad\quad rec.B.A.p_B(n_B) \rightarrow Stop
\end{array}
$$

The argument we make is similar to the case for $RUSER_A$. The first message is receipt of $p_B(y.j)$. If $\neg I(p_B(y.j))$ then the requirement is vacuously met, so we have only to consider the case where $I(p_B(y.j))$. The definition of I yields that $I(y)$.

The second message is a transmission of $p_A(y.n_B.B)$ where y is the nonce received in the first message. The definition of I ensures that this message meets I. The final message is receipt of another message, so cannot violate the requirement **maintains** I. Hence $USER_{ba}$ **sat** **maintains** I as required.

A final step

Technically, the restriction of the analysis of $USER_B$ to the case where the initiating agent is A means that the theorem has proven that

$$
\begin{array}{l}
(USER_A \;|||\; USER_{ba}) \,|[\, trans, rec \,]|\, ENEMY \textbf{ sat} \\
\quad rec.B.A.p_B(n_B) \textbf{ authenticates } trans.A.B.p_B(n_B)
\end{array}
$$

Here we give the technical justification that this restriction is justified.

Any system which is unable ever to perform the event $rec.B.A.p_B(n_B)$ also meets the specification

$rec.B.A.p_B(n_B)$ **authenticates** $trans.A.B.p_B(n_B)$, vacuously. The run of $USER_B$ with a particular agent j may be described as

$$\begin{aligned} USER_{bj} \quad = \quad & rec.B.j?p_B(y.A) \rightarrow \\ & trans.B!j!p_j(y.n_B.B) \rightarrow \\ & rec.B.j.p_B(n_B) \rightarrow Stop \end{aligned}$$

and then $USER_B = \Box_{j \in USER} \; USER_{bj}$ describes $USER_B$ as the choice of runs over all users. This description is equivalent to the original one given for $USER_B$, since an input of j is semantically identical to a choice over all j.

If $j \neq a$, then the fact that $USER_{bj}$ is unable to perform $rec.B.A.p_B(n_B)$ means that its composition with $USER_A$ and $ENEMY$ is also unable to do so, and hence that the authentication property holds vacuously:

$$(USER_A \;|||\; USER_{bj}) \,|[\, trans, rec \,]|\, ENEMY \textbf{ sat}$$
$$rec.B.A.p_B(n_B) \textbf{ authenticates } trans.A.B.p_B(n_B)$$

The reasoning above for $USER_{ba}$ has already established that this holds for the case where $j = a$.

The choice operator preserves specifications: if all processes meet a specification, then so does the process consisting of the choice between them. This means that

$$\Box_{j \in USER} (USER_A \;|||\; USER_{bj}) \,|[\, trans, rec \,]|\, ENEMY$$
sat
$$rec.B.A.p_B(n_B) \textbf{ authenticates } trans.A.B.p_B(n_B)$$

Finally, we make use of the fact that both forms of parallel composition distribute over choice to obtain

$$(USER_A \;|||\; (\Box_{j \in USER} \; USER_{bj})) \,|[\, trans, rec \,]|\, ENEMY$$
sat
$$rec.B.A.p_B(n_B) \textbf{ authenticates } trans.A.B.p_B(n_B)$$

or in other words

$$NET \textbf{ sat}$$
$$rec.B.A.p_B(n_B) \textbf{ authenticates } trans.A.B.p_B(n_B)$$

as required.

8 Multiple runs

All the analysis performed above has been on a system where there is but a single run of the protocol between A and B, and where A and B take the roles of initiator and responder respectively. While it is possible informally to generalise the verifications to systems with repeated runs of the protocol, it is also possible to describe in CSP the situation where agents perform multiple runs of the protocol and hence provide a formal verification. Analysis of a single run allows attention to be focussed on the two participants of the run. In the case where we have multiple runs, it is appropriate to model the two parties as able to engage in their other runs with any other parties, and restrict their behaviour only for the protocol run under analysis. This is the approach that we shall take.

One issue to be addressed concerns the requirement to use a fresh nonce on every protocol run. This may be modelled in CSP by using an infinite sequence of nonces where $n_{A,k}$ and $n_{B,k}$ are used on the kth run of A and B respectively. The kth run of the protocol will be defined in terms of what occurs during that run, and when the $k+1$th run can commence.

The most general situation allows an agent to take the role of initiator or of responder independently in each run. These possibilities are captured in the description of a user by allowing the $l+1$th run to begin at any point after the start of the lth run. In particular, $USER_i(l)$ describes user i at the point where the lth run of the protocol is ready to be executed. As soon as it begins, the description enables the $l+1$th run to execute concurrently with run l (and any earlier runs which are still proceeding).

The processes $USER_A(k)$ are described as follows:

$$\begin{aligned} & USER_A(k) = \\ & \quad \Box_{i \in USER} \\ & \qquad trans.A!i!p_i(n_{A,k}.A) \rightarrow \\ & \qquad\quad \left(\begin{array}{l} rec.A.i?p_A(n_{A,k}.x.i) \rightarrow \\ \quad trans.A!i!p_i(x) \rightarrow Stop \\ ||| \; USER_A(k+1) \end{array} \right) \\ & \quad \Box \; rec.A?j?p_A(y.j) \rightarrow \\ & \qquad \left(\begin{array}{l} trans.A!j!p_j(y.n_{A,k}.A) \rightarrow \\ \quad rec.A.j.p_A(n_{A,k}) \rightarrow Stop \\ ||| \; USER_A(k+1) \end{array} \right) \end{aligned}$$

The possibilities for $USER_A(k)$ are as follows:

1. to transmit a message of the form $trans.A.i.p_i(n_{A,k}.A)$ corresponding to $USER_A$ initiating a run of the protocol. The behaviour subsequent to this message is the rest of the messages appropriate to the protocol run, together

with the possibilities of $USER_A(k+1)$ that the next protocol run will begin.

2. to receive a message of the form $rec.A.j.p_A(y.j)$, purporting to be a message from j initiating a protocol run with A. The subsequent behaviour is to continue through the protocol run, while also offering the possibility to begin the next protocol run.

We define $USER_A = USER_A(0)$: the user begins in a state where no runs have been performed.

The description of user $USER_B$ is exactly that of $USER_A$ with occurrences of the name A replaced by occurrences of the name B.

The appropriate version of Property 6 for this description is the requirement that $rec.B.A.p_B(n_{B,k})$ authenticates $trans.A.B.p_B(n_{B,k})$ for any given k.

Theorem 6.1 is applicable for any descriptions of $USER_A$ and $USER_B$, including recursive ones. If we use the same predicate I as was previously given (with $n_{B,k}$ replacing n_B in the definition), then we have already established that conditions $C1$–$C3$ are met; and we have only to establish $C4$: that $USER_i \,|[\, trans.A.B.p_B(n_{B,k}) \,]|\, Stop$ **sat** `maintains` I for $i = a$ and $i = b$.

The CSP proof rule for recursive definitions allows a proof to proceed which has a structure similar to the proof for the single run given above. The proof rule states that if a recursive definition can be shown to *preserve* a specification, then the recursively defined process *meets* that specification. In the case we are considering, we wish to show that $RUSER_A(l)$ **sat** `maintains` I, where $RUSER_A(l)$ is $USER_A(l)$ restricted on $trans.A.B.p_B(n_{B,k})$; each process $RUSER_A(l)$ is defined in terms of $RUSER_A(l+1)$. The rule states that if, for each l, under the *assumption* that $RUSER_A(l+1)$ **sat** `maintains` I it is possible to show that $RUSER_A(l)$ **sat** `maintains` I, then we may deduce that $RUSER_A(l)$ **sat** `maintains` I for each l.

The assumption that $RUSER_A(l+1)$ **sat** `maintains` I means that when $RUSER_A(l)$ is analysed, it is only the messages that the lth protocol run itself can introduce which need to be considered; the assumption on $RUSER_A(l+1)$ means that we already assume that no subsequent runs will violate the requirement.

The analysis of the messages in the lth protocol run is entirely similar to the analysis already carried out on the single run of the protocol presented above, and does not need to be reproduced here. The main difference is the requirement to incorporate an extra case analysis of $k = l$ and $k \neq l$.

9 Equations

We might also wish to allow for equations on the message space. Some natural ones would be those describing the relationship between encryption and decryption:

$$E1 \qquad p_i(s_i(m)) = s_i(p_i(m)) = m$$

This would remove the need for $K1$ and $K2$ in the definition of the generates relation, since they reduce under the equality to $\{m\} \vdash m$ which is already covered by $A1$.

There would also be properties such as associativity of concatenation:

$$E2 \qquad m_1.(m_2.m_3) = (m_1.m_2).m_3$$

Equations could also capture possible properties of encryption mechanisms. For example, encryption distributing over catenation would be quite a significant weakness:

$$E3 \qquad k(m_1.m_2) = k(m_1).k(m_2)$$

Such equations can be imposed on the message space, but in general their presence might allow additional attacks on protocols. It is useful to understand conditions under which a proof of a protocol's correctness remains valid in the face of such equations.

In fact, the only impact the introduction of an equation has on a proof is that it may make the predicate I ill-defined. Since I is often defined structurally over the message space, it is possible that two different ways of constructing the same message m may correspond to contradictory values for $I(m)$. It is necessary to check that this possibility has not arisen in order for the proof to remain valid.

To check that the invariant I is well-defined, it is necessary to check that if $m_1 = m_2$ then $I(m_1) \Leftrightarrow I(m_2)$ for any equations given. This states that I respects the relevant equations.

Of the three that have been given here, I is easily shown to respect $E1$ and $E2$. On the other hand it does not respect $E3$.

In fact, there can be no invariant in the presence of $E3$: an attempt to construct one led to the discovery

of an attack, where n_C is a nonce which the intruder can generate.

$$\begin{array}{lcl} A \to B & : & p_B(n_A.A) \\ B \to I(A) & : & p_A(n_A.n_B) \\ I(B) \to A & : & p_A(n_A.n_B.n_C) \\ A \to I(B) & : & p_B(n_B.n_C) \\ I(A) \to B & : & p_B(n_B) \end{array}$$

This run establishes the possibility that A takes $n_B.n_C$ to be B's nonce challenge, and so B's completion of the protocol does not provide authentication that A received B's nonce challenge B.

10 Discussion

In this paper we have shown how the theory of CSP might be specialised to provide a theory for reasoning about authentication protocols. The process of verification requires the assumptions about encryption mechanisms and about the capability of a hostile agent to be made explicit. The theory includes a CSP model of the framework containing the protocol; rules for establishing authentication of messages within a single agent; and a general theorem for deducing authentication between agents from their properties with respect to an invariant on messages, together with rules for deriving the required properties.

We find that construction of the invariant predicate forces consideration of the precise reasons why a protocol is expected to work. In this respect, it should reflect the understanding of the protocol designer and make this understanding precise and explicit. Failed attempts to construct an invariant may also provide insight as to why a protocol does not provide authentication. Use of a weak encryption mechanism which allows $k(m_1.m_2) = k(m_1).k(m_2)$ would not provide authentication, as stated in Section 9, and the difficulty in finding the invariant may lead to the discovery of the attack, as indeed happened here.

The CSP language, in common with other process algebras such as CCS, provides a language suitable for the description of protocols in a natural way. Abadi and Gordon [1] observe that this kind of approach combines a precise and solid foundation for reasoning about protocols together with a clear relationship to implementations.

Another benefit of the process algebra approach is to identify precisely the authentication property required of a protocol. This may be left vague in the original formulation of the protocol, and performing the verification often gives information as to which properties actually hold, as well as pinning down precisely the properties which are provided by the protocol. The vital question as to whether the properties obtained are indeed those required are beyond the scope of the formal analysis itself, and must really be assessed according to the intended use of the protocol. Security properties may be captured as CSP specifications, or alternatively in terms of equivalences between processes, as is done in [1], where a network built using the protocol is required to be equivalent to a network which describes the effect of a correct operation of the protocol: equivalence means that the protocol must operate correctly: that no context written in the process algebra can distinguish between the actual protocol and its specification.

Comparison with other approaches

The CSP approach put forward by this author in [20] advocated the encapsulation of required properties in terms of the interactions between the protocol agents and their users. The intention is to separate out the required properties from the way of implementing them. Since a property of a system should be described in terms of its possible interactions with its environment, the internal events $trans.i$ and $rec.i$ should not appear in the property description. This approach requires extra events such as $A.connect\_to.B$ and $B.authenticated.A$ to appear in the protocol description as captured in $USER_A$ and $USER_B$. The descriptions might then be as follows:

$$\begin{array}{lcl} USER_A & = & a.connect\_to?i \to \\ & & \quad trans.A!i!p_i(n_A.A) \to \\ & & \quad\quad rec.A.i?p_A(n_A.x.i) \to \\ & & \quad\quad\quad trans.A!i!p_i(x) \to Stop \end{array}$$

$$\begin{array}{lcl} USER_B & = & rec.B.A?p_B(y.A) \to \\ & & \quad trans.B!a!p_A(y.n_B) \to \\ & & \quad\quad rec.B?i.p_B(n_B) \to \\ & & \quad\quad\quad B.authenticated.A \to Stop \end{array}$$

Our top level requirement would be that

$$NET \setminus (trans \cup rec) \textbf{ sat}$$
$$B.authenticated.A \textbf{ authenticates } A.connect\_to.B$$

The essential proof strategy will remain that proposed in this paper, but the high-level description of the property will be purely in terms of the interactions between the network and its environment. The proof rules will remain unchanged, though there will be an additional

assumption required that messages not meeting the invariant I are not introduced to protocol agents by the environment; this was guaranteed when such agents had no channels apart from *trans* and *rec*.

The approach of including additional 'control' events into a protocol description is appropriate both for abstracting away the details of the protocol description, and also for providing a clearer understanding of what the protocol designer is attempting to achieve. This separation of concerns allows authentication specifications to be formulated independently of the details of any particular protocol. External events are introduced in other CSP approaches to protocol analysis [18, 10] and elsewhere [23]. In the case of the analysis performed here, additional external events such as $A.nonce\_challenge\_OK.B$ and $A.last\_message.B$ might be included to make finer distinctions between different flavours of authentication. This approach is simply an straightforward extension of the approach we have in fact taken in this paper, introducing special events are introduced to mark particular points in the run of the protocol which we have been pinpointing directly.

Another issue of interest is that Definition 5.1 is not concerned with matching up occurrences of events from T and R: once some event from R has occurred then this definition allows arbitrarily many events from T to occur without breaking authentication. Lowe [11] discusses a stronger version of authentication which requires each authenticating event to correspond to a different authenticated event—he terms this requirement 'injectiveness'. Such a property is important for example in the authorisation of financial transactions. This property can be captured by strengthening the above definition as follows:

Definition 10.1

$$T \texttt{ injectively authenticates } R \mathrel{\widehat{=}} \#(tr \upharpoonright R) \geqslant \#(tr \upharpoonright T)$$

□

If T `injectively authenticates` R then it follows immediately that T `authenticates` R, since if $tr \upharpoonright R = \langle\rangle$ then $\#(tr \upharpoonright R) = 0$ and so $\#(tr \upharpoonright T) = 0$, and thus $tr \upharpoonright T = \langle\rangle$.

The techniques developed in this paper are concerned only with non-injective authentication. Their extension to deal with injectiveness is an area of ongoing research.

This approach contrasts with that taken in [4, 18] where specifications are what Roscoe calls *intensional*, requiring that the protocol works 'as expected' in some sense. Such properties cannot be expressed as CSP specifications independently of the protocol itself, and they really correspond to a recipe for providing the specification appropriate to the particular protocol. For example, Gollmann identifies a number of authentication properties in [6]; the one closest to those we have considered here is $G4$, which states that "the origin of all messages in the protocol has to be authenticated." We would treat this as a conjunction of authentication properties: $r3$ authenticating $s1$, $r3$ authenticating $s3$ and $r2$ authenticating $s2$ (stipulating that authentication is required only when the protocol run is complete). Other examples are given in [19], which gives the 'canonical' intensional specification as one where the interleavings of messages at different agents are in accordance with the messages in the protocol; and a slightly weaker on in [4], which requires that whenever a participant completes its part of a protocol run then the other participant must have engaged in the sequence of events described by the protocol. These intensional properties can be formulated for any particular protocol, but not in CSP terms independently of any protocol. They correspond to recipes for producing a specification appropriate for a given protocol.

A different process algebraic approach is taken by Abadi and Gordon [1], where the pi-calculus [14] is extended (to the spi calculus) to model encryption. The resulting language allows protocols to be described in a straightforward way; the treatment of freshness of nonces and keys is more explicitly provided by the process description language itself: encapsulation with the ν operator provides a natural and pleasing model of nonce and key generation. Correctness of a protocol is established by showing that it is testing equivalent to a specification process that describes explicitly what the protocol is intended to achieve. In other words, they are indistinguishable in any context. As a result, the enemy does not have to be modelled explicitly. The capabilities of the enemy are precisely those that can be described within the spi calculus language: a context distinguishing between protocol and its specification would describe an attack on the protocol, and conversely if no context distinguishes them, then the protocol implements the specification in the context of any enemy which may be described in the spi calculus. This contrasts with the approach taken in this paper, where the specification is separated to some extent from the protocol, and which allows finer distinctions to be drawn between different notions of authentication. The explicit description of the enemy

allows analysis with regard to different enemy capabilities within the same framework. More comparative examples are required to explore the relative merits of each approach.

The approach of providing an invariant I as the core of a proof is complementary to the tools based approaches [9, 13, 8, 18] which search for attacks. The results of the latter kind of analysis provides useful information as to why a protocol is not correct, or alternatively gives a bald statement that no attack can be found. This is appropriate for debugging, but does not provide understanding as to why a protocol is correct. It is a claim of this paper that the invariant provides the basis for such an understanding, and it might be profitable to explore the interplay between the state exploration approaches and proofs. One problem concerns the relationship between the finite nature of the state space and the infinite possibilities for attacks (such as arise from such aspects as the possibility of arbitrary depths of encryption and combining of messages), and Lowe [10] has begun to consider a proof strategy based on the general form of a protocol run for establishing when the absence of an attack on a finite state space really does imply that no attack is possible on the infinite state space. It seems that the invariant approach might also be useful in this context.

Paulson [17] has investigated the application of the proof tool Isabelle/HOL to proving security properties of protocols. He specifies security protocols in terms of traces of the system as a whole. The rounds of a protocol are translated into rules about how system traces may be augmented. Possible enemy activities also become rules. Once all of the rules have been identified, the aim is to prove that any system trace that can be generated using the rules must meet the required property; this is established by induction. He does not use an explicit invariant, but he also aims to prove that particular messages can never occur in a trace, and this requires certain lemmas establishing that particular classes of message cannot occur. This is also a feature of the approach taken in [12], which applies language theory to establish that particular terms cannot be generated using given production rules. Mechanical assistance for proofs is invaluable, and Paulson has some useful results concerning reusability of proof strategies. It appears that a number of protocols have proofs of a similar shape, which allows efficient analysis of new protocols. Recent work on providing mechanical assistance to the CSP proofs using PVS [16] is encouraging, but at present[1] it is still necessary to provide the invariant in order for the proof to proceed, so analysis of fresh protocols will not be as automatic as Paulson reports for Isabelle/HOL.

Other approaches to direct proof of protocols, rather than the indirect route by establishing absence of attacks, tend to be based on formal languages for describing security properties, together with rules which support reasoning about statements in the language. Protocols are modelled as rules which allow the derivation of new statements from existing ones. The best known example of such a language is the BAN logic [2], though the need for 'idealisation' of protocols into the logic means that the link between a protocol and its logical treatment is informal. The formal language described in [22] contains lower-level primitives which relate more directly to the steps taken by a protocol, and supports reasoning concerning the knowledge of the intruder at particular stages. This language is used in conjunction with the NRL protocol analyser which is used to check reachability of negated requirements, so it is closer to the tools based approaches. However, an approach to proof reflecting that presented in this paper seems feasible.

Future directions

This paper has presented an approach to analysing and verifying authentication protocols, driven in part by consideration of the Needham-Schroeder protocol. The verification was done by hand, and the cryptographic mechanisms considered are straightforward: nonces, and public-key encryption. There is an obvious need to investigate the CSP handling of other security mechanisms such as timestamps, and to investigate more complicated protocols. Other security properties such as confidentiality should also be investigated. It seems likely that the same approach will apply to confidentiality properties, since such properties may be expressed in terms of unreachability.

Some form of mechanical assistance for proof appears feasible, and this would bring benefits in managing the details of the proofs which can become cumbersome to track. A theory based around the rules given in the paper has now been described in the theorem prover PVS [16], and investigations are continuing into the assistance that can be provided into finding the invariant.

Acknowledgements

Thanks to Peter Ryan, Irfan Zakiuddin, Jeremy Bryans, Bruno Dutertre, Andy Gordon, Gavin Lowe,

[1] January 1997

Cathy Meadows, Abraham Sidiropoulos, and the anonymous referees for comments on earlier forms of this work. It is also a pleasure to thank Bruno Dutertre for his work on checking the proofs with PVS and providing a number of corrections, underlining the necessity for mechanical assistance to make such proofs manageable.

I am grateful to the Defence Research Agency for the provision of funding.

References

[1] M. Abadi and A.D. Gordon, *A calculus for cryptographic protocols: the spi calculus,* University of Cambridge, draft, 1996.

[2] M. Burrows, M. Abadi, and R. Needham, *A logic of authentication,* Technical Report 39, SRC, DEC, 1989.

[3] J. Davies and S. Schneider, *Recursion induction for real-time processes,* Formal Aspects of Computing 5(6), 1993.

[4] W. Diffie, P.C. van Oorschot, and M.J. Wiener, *Authentication and key exchanges,* Designs, Codes and Cryptography 2, 1992.

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

[6] D. Gollmann, *What do we mean by Entity Authentication?* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1996.

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

[8] R Kemmerer, C. Meadows, and J. Millen, *Three systems for cryptographic protocol analysis,* Journal of Cryptology, 7(2), 1994.

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

[10] G. Lowe, *Breaking and fixing the Needham-Schroeder public-key protocol using FDR,* TACAS, LNCS 1055, 1996.

[11] G. Lowe, *A hierarchy of authentication specifications,* in preparation, 1996.

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

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

[14] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes,* parts I and II, Information and Computation 100, 1992.

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

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

[17] L.C. Paulson, *Proving Properties of Security Protocols by Induction,* Computer Laboratory, University of Cambridge, 1996.

[18] A.W. Roscoe, *Modelling and verifying key-exchange protocols using CSP and FDR,* Proceedings of CSFW8, IEEE press 1995.

[19] A.W. Roscoe, *Intensional specifications of security protocols,* Proceedings of CSFW9, IEEE press 1996.

[20] S.A. Schneider, *Security Properties and CSP,* IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, 1996.

[21] 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, 1996.

[22] P. Syverson and C. Meadows, *A formal language for cryptographic protocol requirements,* Designs, Codes and Cryptography 7, 1996.

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