Modelling security properties with CSP

Steve Schneider
Royal Holloway, University of London

Technical Report

CSD-TR-96-04

February 29, 1996

Department of Computer Science
Egham, Surrey TW20 0EX, England

Abstract

Security properties such as confidentiality and authenticity may be considered in terms of the flow of messages within a network. To the extent that this characterisation is justified, the use of a process algebra such as Communicating Sequential Processes (CSP) seems appropriate to describe and analyse them. This paper explores ways in which security properties may be described as CSP specifications, how security mechanisms may be captured, and how particular protocols designed to provide these properties may be analysed within the CSP framework. The paper is concerned with the theoretical basis for such analysis. A formal verification of a simple example is carried out as an illustration.

1 Introduction

Security protocols are designed to provide properties such as authentication, key exchanges, key distribution, non-repudiation, proof of origin, integrity, confidentiality and anonymity, for users who wish to exchange messages over a medium over which they have little control. These properties are often difficult to characterise formally (or even informally). The protocols themselves often contain a great deal of combinatorial complexity, making their verification extremely difficult and prone to error. This paper promotes the view that process algebra can provide a single framework both for modelling protocols and for capturing security properties, facilitating verification and debugging. It is a discussion paper, proposing possible approaches rather than providing definitive answers.

It has been argued that security properties should be considered as properties concerning the flow of messages within a network. To the extent that this characterisation is justified, the use of a process algebra such as CSP [Hoa85] seems appropriate to describe and analyse them. This paper considers ways in which security properties may be described using the notation of CSP, how security mechanisms may be captured, and how particular protocols designed to provide these properties may be analysed within the CSP framework. Some familiarity with CSP is assumed.

The approach presented is rather general, and it is clear that the modelling of particular properties and analysis of particular protocols will require tailoring of the model presented here. But this paper aims at exploring a general approach rather than trying to construct a universal model suitable for handling all possible security issues, which is probably an unrealistic goal.

Security properties are generally properties requiring that something bad should not occur (though they are not exclusively of this form). Of course, particular communication protocols will also aim to be live (something good should occur), in that they will be designed to achieve goals such as delivery of messages. But there is a distinction to be drawn between the security requirements implemented by such a protocol, and its liveness requirements which are important for communication but which are generally independent of security. It is possible that there are some security properties which can be expressed only as liveness properties; these are outside the scope of this paper. Hence the traces model for CSP will be adequate for our present needs: to analyse properties of the form 'something bad should not happen' it is sufficient to focus on what systems may do, rather than what they must do. All equivalences and refinements expressed in this paper are therefore grounded in the traces model.

2 Security properties

A network provides a means for users such as people or applications programs to communicate by sending and receiving messages. This situation may be modelled at a high level of abstraction in CSP as a process NET which provides to each user two ways of interacting with it: sending messages to other parties, and receiving messages from other parties.

We will assume a universal set $MESSAGE$ of all messages that might be sent by any party, and we will consider the users to be numbered up to n:

$$USER = \{0, 1, \ldots, n\}$$

The channel employed by user i to input messages to the network will be the input channel $in.i$, of type $USER.MESSAGE$. An input of the form $in.i.j.m$ is considered an instruction from user i to transmit message m to user j.

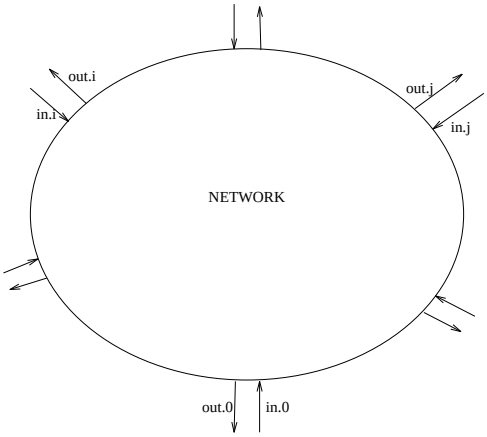

Figure 1 High level view of the network

The channel employed by user i to receive messages output from the network will be the output channel $out.i$, of type $USER.MESSAGE$. An output of the form $out.i.j.m$ is considered to be receipt by user i of message m sent by user j.

Users' requirements on the network are expressed in terms of the behaviour of the network as a whole, and CSP has been used successfully for some years in the description and analysis of communications protocols. Common safety and liveness properties are readily expressed in terms of the possible behaviour of the network with respect to the users. For example, the property that no spurious messages are generated is captured as a safety property that requires any output message to have previously been input: if $out.i.j.m$ appears in a trace, then $in.j.i.m$ must have already occurred. The liveness requirement that no message is lost can be formalised as follows: for any input message $in.i.j.m$ the corresponding output $out.j.i.m$ must eventually become available. Generally these properties are expressed precisely and formally in terms of the semantic models of CSP.

Although it is necessary to know the internal structure of the network in order to demonstrate that it provides particular services, the services or properties themselves should be expressible simply in terms of the interactions the network offers its users. This is the case for common communications protocols, and in this paper we take the view that security properties can be captured in the same way. We therefore examine and offer definitions of these properties before considering the network at any finer level of detail.

There are two views from which security properties can be considered:

- from the viewpoint of the users of the network, who do not know which other parties are to be trusted. Properties expressed from this viewpoint will generally include assumptions (implicitly or explicitly) that a user's communication partner will not act contrary to the aims of the protocol. For example, that any shared secrets should not be disclosed to third parties.

- from a high-level 'God's eye view' which identifies those nodes which follow their protocols faithfully, and also identifies those which are engaging in more general activity, perhaps in attempting to find a flaw in a protocol. If this view is taken then care should be taken to ensure that this privileged information is not accidentally used in protocol description: the responses of a node should not be dependent on information which is available only at the high-level view. In some circumstances, a node may not have knowledge concerning its communication partner; in other cases, a protocol may be invoked only when communicating with particular known and trusted users (how this knowledge and trust is obtained is outside the scope of this report).

We will follow the high-level view in this report. This means we can postulate the existence of an enemy whose identity is known and can be used in the formulation of security properties. We will use $0 \in USER$ as the name of this enemy process. Later we will justify the decision to use only a single enemy, by arguing that further enemies do not increase the vulnerability of protocols: the single enemy in a sense encapsulates the behaviour of all enemies.

Confidentiality

Confidentiality is achieved when users may communicate particular messages (drawn from the set M) without the possibility of any user other than the intended recipient receiving them. In other words, if an input $in.i.j.m$ occurs, then any (subsequent) output $out.h.l.m$ must be for user j: i.e. $h = j$. Thus given user j and message m, if an output $out.j.i.m$ occurs (for some i) then some user l (not necessarily i) must have sent that message to j: there must be some previous input of the form $in.l.j.m$. Thus j cannot obtain messages that were intended for some other user.

From the God's-eye view, the only user which might obtain messages intended for some other user will be user 0. Hence confidentiality will be captured as a specification requiring that any message output by user 0 must have actually been sent to user 0. We restrict attention to the message set M as being those messages which are intended to remain confidential. We also assume they cannot be generated by user 0 (which would be true for example for signed messages), though this is a simplifying assumption that is not justified in all circumstances. This assumption is implicit in the definition, since otherwise 0 could simply guess any confidential message. Other messages (such as encrypted messages or control messages) will in general be available to eavesdroppers, but confidentiality is not concerned with protecting these.

These considerations may be captured as a trace specification

Definition 2.1 NET provides confidentiality for the set of messages M if and only if

$$NET \textbf{ sat } \forall m : M \bullet tr \upharpoonright out.0.USER.m \neq \langle\rangle \Rightarrow tr \upharpoonright in.USER.0.m \neq \langle\rangle$$

□

This may also be expressed within the CSP process algebra:

$$\forall m : M \bullet NET \underset{in.USER.0.m}{\parallel} STOP = NET \underset{\substack{out.0.USER.m \\ in.USER.0.m}}{\parallel} STOP$$

Observe that this is not equivalent to

$$NET \underset{in.USER.0.M}{\|} STOP = NET \underset{\substack{out.0.USER.M \\ in.USER.0.M}}{\|} STOP$$

For example, $NET = in.1.0.m1 \longrightarrow out.0.1.m1 \longrightarrow in.1.2.m2 \longrightarrow out.0.1.m2 \longrightarrow STOP$ meets the latter equivalence but not the former: a message $m2$ from user 1 to user 2 has been output by user 0, and this breaches confidentiality (assuming $m1, m2 \in M$).

The quantification over all messages in M is necessary.

This property may also be captured in the traces model, as the property $CONF$

$$CONF(tr) \mathrel{\widehat{=}} messages(tr \upharpoonright out.0.USER.M) \subseteq messages(tr \downarrow in.USER.0.M)$$

This states that the messages (from message set M) output from user 0 must be a subset of those that were sent to it. In other words, user 0 cannot obtain any messages from M that are not sent to it.

The fact that this property is a **sat** specification means that it is preserved by refinement.

It is an immediate consequence of the process algebra characterisation that a system providing confidentiality for two sets $M1$ and $M2$ separately it provides it confidentiality for both sets simultaneously: if

$$\forall m : M1 \bullet NET \underset{in.0.j.m}{\|} STOP = NET \underset{\substack{out.j.0.m \\ in.0.j.m}}{\|} STOP$$

and

$$\forall m : M2 \bullet NET \underset{in.0.j.m}{\|} STOP = NET \underset{\substack{out.j.0.m \\ in.0.j.m}}{\|} STOP$$

then

$$\forall m : M1 \cup M2 \bullet NET \underset{in.0.j.m}{\|} STOP = NET \underset{\substack{out.j.0.m \\ in.0.j.m}}{\|} STOP$$

A simplification

Observe that if no messages are ever sent to 0 (perhaps if users are communicating with users they know and trust to be honest) then the characterisation of confidentiality may be simplified, since no messages will ever be sent to user 0.

The definition simplifies to

$$NET \textbf{ sat } \forall m : M \bullet tr \upharpoonright out.0.USER.m = \langle\rangle$$

which is equivalent to the simpler form

$$NET \textbf{ sat } tr \upharpoonright out.0.USER.M = \langle\rangle$$

This property may be expressed entirely within the process algebra in a number of different ways. The first way captures the idea that if attention is focussed entirely upon events from $out.0.USER.M$, then nothing should be observed:

$$NET \setminus (\Sigma \setminus out.0.USER.M) \;=\; STOP$$

The process $STOP$ is a refinement of $NET \setminus (\Sigma \setminus out.0.USER.M)$ (since in the traces model is is a refinement of every process), so achieving equality is equivalent to obtaining refinement in the other direction:

$$STOP \sqsubseteq NET \setminus (\Sigma \setminus out.0.USER.M)$$

An alternative characterisation is obtained by considering the effects of preventing NET from performing any events in $out.0.USER.M$. A system providing confidentiality should not be affected by this restriction:

$$NET \;=\; NET \underset{out.0.USER.M}{\|} STOP$$

Since restricting the behaviour of NET can only reduce its behaviours, it follows automatically that the restriction is a refinement of NET. Hence the processes are equivalent precisely when there is a refinement in the other direction:

$$NET \underset{out.0.USER.M}{\|} STOP \sqsubseteq NET$$

A final characterisation regards the system as acceptable if every event it can perform is in the set $\Sigma \setminus out.0.USER.M$. In other words, everything it can perform is also possible for a process which can always perform any of those events:

$$RUN_{\Sigma \setminus out.0.USER.M} \sqsubseteq NET$$

All of these characterisations are provably equivalent to the assertion NET **sat** $tr \upharpoonright out.0.USER.M = \langle\rangle$

It is straightforward using the process algebra to show that if a system provides confidentiality for two sets $M1$ and $M2$ separately, then it provides confidentiality for both sets simultaneously: if

$$NET \underset{out.0.USER.M1}{\|} STOP \;=\; NET$$

and

$$NET \underset{out.0.USER.M2}{\|} STOP \;=\; NET$$

then

$$\begin{aligned} NET \underset{out.0.USER.(M1 \cup M2)}{\|} STOP \;&=\; NET \underset{out.0.USER.M1}{\|} STOP \underset{out.0.USER.M2}{\|} STOP \\ &=\; NET \underset{out.0.USER.M2}{\|} STOP \\ &=\; NET \end{aligned}$$

Message Authentication

This property requires that messages can be guaranteed to be 'authentic', in the sense that a particular message purporting to have come from a particular source really did come from that source. Authentication requires that messages cannot be forged.

In abstract terms, event b 'authenticates' event a if the observation of b is possible only if a occurred previously: the observation of b provides 'evidence' of a's previous occurrence.

Definition 2.2 Event b authenticates event a in process P if and only if P **sat** $AUTH(tr)$, where

$$AUTH(tr) \;=\; tr \upharpoonright b \neq \langle\rangle \Rightarrow tr \upharpoonright a \neq \langle\rangle$$

□

Observe that this specification does not restrict the number of occurrences of event b to each occurrence of event a.

The expression of this property in terms of a **sat** specification demonstrates that it is preserved by refinement.

This specification can also be captured as a process algebraic equation.

$$P \underset{a,b}{\|} STOP \;=\; P \underset{a}{\|} STOP$$

And since it is always the case that

$$P \underset{a}{\|} STOP \sqsubseteq P \underset{a,b}{\|} STOP$$

the condition is equivalent to

$$P \underset{a,b}{\|} STOP \sqsubseteq P \underset{a}{\|} STOP$$

For example, the process

$$\begin{aligned} P \;=\;& a \longrightarrow b \longrightarrow STOP \\ & \Box \\ & b \longrightarrow c \longrightarrow STOP \end{aligned}$$

has c authenticating b:

$$\begin{aligned} P \underset{b,c}{\|} STOP \;&=\; a \longrightarrow STOP \\ &=\; P \underset{b}{\|} STOP \end{aligned}$$

but it does not have b authenticating a:

$$\begin{aligned} P \underset{a,b}{\|} STOP \;&=\; STOP \\ &\neq\; b \longrightarrow c \longrightarrow STOP \\ &=\; P \underset{a}{\|} STOP \end{aligned}$$

In other words, a b event can occur even if a did not occur previously.

In the context of sending and receiving messages, we would require a received message $out.j.i.m$ to authenticate a sent message $in.i.j.m$. In other words, receipt of the message $i.m$ ('m from i') by node j is possible only if that message was sent by node i. Thus on a system NET consisting of the medium, enemy and nodes, the property to check would be

$$NET \underset{\substack{in.i.j.m \\ out.j.i.m}}{\|} STOP = NET \underset{in.i.j.m}{\|} STOP$$

For example, a buffer process

$$COPY = in?x \longrightarrow out!x \longrightarrow COPY$$

has $out.x$ authenticating $in.x$ for any x: no message can be output unless it has previously been input.

This characterisation of authentication can be promoted to sets of events The set B authenticates the set A in P if any of the messages in B authenticates any of the messages in A. In other words, if any of the messages in B is seen, then one of the messages in A must previously have occurred. This is captured as follows

Definition 2.3 B authenticates A in P if and only if

$$P \underset{A \cup B}{\|} STOP = P \underset{A}{\|} STOP$$

□

This form might be useful when we wish to check that a message is genuine even when its originator is unknown. This could be captured as the set $out.j.USER.m$ authenticating the set $in.USER.j.m$. The authenticating message indicates that some honest node generated the original message.

The buffer process $COPY$ has the weaker property of $out.M$ authenticating $in.M$: no output can occur before input. This property is strictly weaker than the previous property, in which every output authenticates a corresponding input. For example, a random message generator

$$RAND = in?x \longrightarrow \prod_{y \in M} out!y \longrightarrow RAND$$

also has $out.M$ authenticating $in.M$—no message can be output if there wasn't previously an input—but does not have $out.x$ authenticating $in.x$ for any particular x.

The definition provides a straightforward proof of transitivity of authentication: if C authenticates B, and B authenticates A, then C authenticates A:

$$\begin{aligned} P \underset{A \cup C}{\|} STOP &= P \underset{A}{\|} STOP \underset{C}{\|} STOP \\ &= P \underset{A \cup B}{\|} STOP \underset{C}{\|} STOP \\ &= P \underset{A \cup B \cup C}{\|} STOP \\ &= P \underset{B \cup C}{\|} STOP \underset{A}{\|} STOP \end{aligned}$$

$$\begin{aligned} &= P \underset{B}{\|} STOP \underset{A}{\|} STOP \\ &= P \underset{A \cup B}{\|} STOP \\ &= P \underset{A}{\|} STOP \end{aligned}$$

Furthermore, it follows from this definition that if sets A and B authenticate each other, then $A \cup B$ is authenticated by $A \cap B$. In other words, the first event performed from the union of A and B must in fact be contained in their intersection.

$$\begin{aligned} P \underset{A \cup B}{\|} STOP &= P \underset{A \cup B}{\|} STOP \underset{A \cup B}{\|} STOP \\ &= P \underset{A}{\|} STOP \underset{A \cup B}{\|} STOP \\ &= P \underset{A \cup B}{\|} STOP \underset{A}{\|} STOP \\ &= P \underset{B}{\|} STOP \underset{A}{\|} STOP \\ &= P \underset{A \cap B}{\|} STOP \end{aligned}$$

Thus if A and B are mutually authenticating and disjoint, then no events in either set can occur.

Note also that authentication is also reflexive (A authenticates A), though this is not of much use.

Anonymity

A one-way anonymity property (where the receiver of the anonymous message has no idea who sent it) amounts to the requirement that the anonymous message could have been sent by any of the other users: when a message could have been sent by a particular user, it must also have been possible for any other user.

This may be captured in a number of different ways, depending on precisely which flavour of anonymity is intended.

Strong anonymity

The first approach uses alphabet renaming as an abstraction operator. In providing anonymity, we aim to achieve a situation in which observations that are possible for a particular observer do not allow the observer to deduce which of a set of events, say A, actually occurred. In other words, whenever any event from A occurs, then any of the others should also have been possible. We might approach this in the following way. Consider an alphabet renaming operator f_A which maps all events in A to a new event e[1]. Whenever some event from A is performed, it is replaced with the new event e. We will have anonymity if in fact any event from A could have given rise to the event e—in other words, whenever e is possible in $f_A(P)$, in fact all events from A were possible in the original process P. This can be captured within the process algebra, since the inverse alphabet renaming f_A^{-1} makes all events in A available whenever e is available. Thus the anonymity condition amounts to requiring that $f_A^{-1}(f_A(P)) = P$: whenever any event from A is possible, then all events from A are possible.

[1] The new event is intended to be an event not in the universal set of events Σ. A theoretically cleaner alternative might be to use an arbitrary event a from A, in place of e. I have avoided this alternative here since it may lead to confusion.

Definition 2.4 A process P is strongly anonymous with respect to a set A if and only if $f_A^{-1}(f_A(P)) = P$, where

$$\begin{array}{lcll} f_A(a) & = & e & \text{if } a \in A \\ f_A(a) & = & a & \text{otherwise} \end{array}$$

for some new event e □

As an example, consider the process

$$\begin{array}{lcl} P & = & a_1 \longrightarrow b \longrightarrow STOP \\ & & \Box \\ & & a_2 \longrightarrow b \longrightarrow STOP \end{array}$$

The intention is that the occurrence of the event b should provide no indication of whether it was a_1 or a_2 that occurred. Thus we use the alphabet renaming function f_{a_1,a_2} and observe that

$$f_{a_1,a_2}^{-1}(f_{a_1,a_2}(P)) = P$$

It follows that P provides strong anonymity with respect to the set $\{a_1, a_2\}$, which is exactly as we would expect.

On the other hand, if we consider the process

$$\begin{array}{lcl} Q & = & a_1 \longrightarrow b \longrightarrow STOP \\ & & \Box \\ & & a_2 \longrightarrow b \longrightarrow b \longrightarrow STOP \end{array}$$

where the number of copies of event b that are possible depends on the initial event chosen, then we find that

$$\begin{array}{lcl} f_a^{-1}(f_a(Q)) & = & a_1 \longrightarrow b \longrightarrow b \longrightarrow STOP \\ & & \Box \\ & & a_2 \longrightarrow b \longrightarrow b \longrightarrow STOP \end{array}$$

This is not the same as the original process Q. For example, it allows the trace $\langle a_1, b, b\rangle$, which is not possible for the original process Q.

Theorem 2.5 P is strongly anonymous with respect to A if and only if $P \sqsubseteq f_A^{-1}(f_A(P))$ □

This follows from the fact that $f_A^{-1}(f_A(P)) \sqsubseteq P$ is always true (in the traces model): any trace of P will be a trace of $f_A^{-1}(f_A(P))$.

Theorem 2.6 If P is strongly anonymous with respect to A and $B \subseteq A$ then P is strongly anonymous with respect to B □

However, a converse result does not hold: if P is strongly anonymous with respect to A and B, this does not mean that it is strongly anonymous with respect to $A \cup B$. This is trivial to see when it is observed that a process is always anonymous with

respect to any singleton set, but not with respect to all sets of size two. As a more concrete example, consider the process

$$\begin{aligned} P \;=\; & a_1 \longrightarrow c \longrightarrow STOP \\ & \Box \\ & a_2 \longrightarrow c \longrightarrow STOP \\ & \Box \\ & b_1 \longrightarrow d \longrightarrow STOP \\ & \Box \\ & b_2 \longrightarrow d \longrightarrow STOP \end{aligned}$$

This process is strongly anonymous for $\{a_1, a_2\}$ and $\{b_1, b_2\}$, but not for their union.

On the other hand, we do obtain the following result:

Theorem 2.7 If P is strongly anonymous with respect to A and B, and $A \cap B \neq \{\}$, then P is strongly anonymous with respect to $A \cup B$. □

Observe also that this property of anonymity is not preserved by refinement, since it is true for the least refined process RUN but not true for all systems. [2]

In the context of sending messages over a network, we will often wish to obtain partial anonymity: we aim to obscure the origin of the message, but we do not wish to obscure its contents. For any particular message $in.i.j.m$, we require the possibility that any node could have sent the message. Thus the property required is that NET is strongly anonymous with respect to the set $S_{j,m} = \{in.k.j.m \mid k \in USER\}$ for any fixed j and m.

In fact, we require a family of anonymity results, one for each $S_{j,m}$. These may be collected together within a single algebraic equation, using a single alphabet renaming that encapsulates all of the separate alphabet renamings at once.

Theorem 2.8 P is strongly anonymous with respect to each of a family $\{A_i\}$ of pairwise disjoint sets of events if and only if $f^{-1}(f(P)) = P$, where

$$\begin{aligned} f(a) &= a_i \quad \text{if } a \in A_i \\ f(a) &= a \quad \text{otherwise} \end{aligned}$$

where the events a_i are all new events. □

From the point of view of the network property mentioned above, the property we must check for NET is that $f^{-1}(f(NET)) = NET$ where a suitable f is given by $f(in.k.j.m) = in.j.m$. The function that removes the originator of the message provides anonymity for messages sent over the network.

[2] It is also important to note that when communication is taking place in the presence of an enemy who is able to monitor all traffic as it goes on the network, then no protocol could achieve anonymity, since the enemy can see the source of any message as it enters the network. Anonymity is normally required when the 'enemy' is considered to be the recipient, who has access only to his own incoming messages.

Weak anonymity

This form of anonymity may be considered too strong in some circumstances, since it requires that *every* instance of any event intended to be anonymous could be replaced by any alternative event. However, in the case of a process such as

$$P = (a \longrightarrow b \longrightarrow c \longrightarrow STOP) \,\Box\, (b \longrightarrow a \longrightarrow c \longrightarrow STOP)$$

We might hope that anonymity is present for the set $\{a, b\}$, since the occurrence of c does not provide sufficient evidence as to which order a and b occurred. Yet the renaming function $f(a) = f(b) = d$ yields

$$\begin{array}{lll} P & = & a \longrightarrow (a \longrightarrow c \longrightarrow STOP \\ & & \qquad \Box\ b \longrightarrow c \longrightarrow STOP) \\ & & \Box \\ & & b \longrightarrow (a \longrightarrow c \longrightarrow STOP \\ & & \qquad \Box\ b \longrightarrow c \longrightarrow STOP) \end{array}$$

which is not the same as P. The point is that once the first event has occurred, the second is constrained to be different to the first and is therefore not independently anonymous; once the first event has occurred, there is only one possibility for the second event in any particular instance. This situation might arise in an anonymous voting algorithm for instance, where each party is allowed to vote only once: anonymity should be present even though subsequent votes must be different from the first.

This form of anonymity is characterised by the property that the same behaviour would result for any permutation of the events for which anonymity is required. If the intention is that no observer should be able to distinguish a from b, then the behaviour of the system should be the same as its behaviour when a and b are exchanged. Indeed, for the function $f(a) = b, f(b) = a$ we do obtain $f(P) = P$ for the process P above.

This suggests an alternative, weaker, characterisation of anonymity.

Definition 2.9 A process P is weakly anonymous with respect to a set of events A if and only if for any permutation p of the set A we have $p(P) = P$ □

This is weaker than the previous definition, since it requires all instances of any particular event to be replaced in the same way, rather than allowing each instance to be possible replaced by any other event.

However, it requires consideration of a large number of equivalences in order to establish anonymity, which grows exponentially in the size of A. This problem can be alleviated to a certain extent by considering only transpositions (permutations which swap precisely two elements): since every permutation is a product of transpositions, we need check that $t(P) = P$ only for those transpositions t on A. And since all transpositions are generated by those involving one particular element a of A, we may restrict attention further, to these transpositions. Define $t_{a,b}$ to be the alphabet transformation that maps a to b, maps b to a, and leaves all other events unchanged.

Theorem 2.10 Given a set A and element $a \in A$, $t_{a,b}(P) = P$ for all $b \in A$ if and only if P is weakly anonymous with respect to A. □

Proof The 'if' direction is trivial, since for every $b \in A$ the function $t_{a,b}$ is a permutation on A.

To prove the 'only if' direction, recall that it is a result in CSP that for functions f and g we have $f(g(P)) = f \fatsemi g(P)$. Since the transpositions $T_a = \{t_{a,b} \mid b \in A\}$ generate the entire group of permutations, any permutation p is equivalent to $t_1 \fatsemi t_2 \fatsemi \ldots \fatsemi t_n$ where each $t_i \in T_a$. In each case $t_i(P) = P$, so $P = t_1(t_2(\ldots t_n(P)\ldots)) = p(P)$. □

Corollary 2.11 If P is weakly anonymous on A and $B \subseteq A$ then P is weakly anonymous on B □

Theorem 2.12 If P is strongly anonymous with respect to A, then it is weakly anonymous with respect to A □

Consideration of the situation with regard to the network again illustrates the need for a specialisation of the notion of anonymity. When messages of the form $in.i.j.m$ are being communicated on the network, the requirement is not so much that every instance of a particular $in.i.j.m$ could be replaced by another message $in.k.j.m$, but rather that every instance of i and j within any message could be replaced by the same k and h. This would correspond better to anonymity in for example business transactions, where a session between two parties might consist of a number of messages generated by each. Rather than consider the permutations of all messages of the form $in.k.j.m$ we wish to consider the subgroup of permutations consisting of permutations of the agent component of the message. This is the subgroup generated by the mappings $f_{1,j}$ defined by

$$\begin{aligned} f_{1,i}(in.1.j.m) &= in.i.j.m \\ f_{1,i}(in.i.j.m) &= in.1.j.m \\ f_{1,i}(a) &= a \qquad \text{if } a \neq in.1.j.m \text{ and } a \neq in.i.j.m \end{aligned}$$

The network will provide the form of anonymity required if $NET = f_{1,j}(NET)$ for every user $j \in USER$.

Integrity

Integrity is sometimes seen as the dual or converse of confidentiality—that there is no leakage of information from low level users to high level users, or in our case from the enemy (user *0*) to the honest parties.

But integrity is also seen as assurance that messages have not been tampered with, an assurance that might be given by a hash value or a checksum. This will be the case even for messages that have been generated by the enemy.

With this viewpoint, it seems that integrity is a form of authenticity, in the sense that any message that is output must follow a particular input message; in this case, the output authenticates the input. If the originator of the message need not be known, then we have simply that output of a message authenticates input of that message by some node. It is a form of authenticity, with the difference that the users are abstracted away.

$$NET \underset{in.USER.USER.m}{\|} STOP = NET \underset{\substack{out.j.USER.m \\ in.USER.USER.m}}{\|} STOP$$

The output message m must have been input by some user.

Non-repudiation

Non-repudiation is concerned with both parties in a transaction obtaining *evidence* that the transaction occurred, in case of subsequent denial by the other party. The evidence should be sufficient to convince an arbitrator or *judge* that the transaction occurred. In this sense it is different from authenticity; a party may know more than can be proven to a judge. It also differs from authenticity in that users are not primarily concerned with malicious interference by another party. For example, a party may not have generated a particular message, and will presumably have knowledge of this. In itself this knowledge does not provide any evidence that would convince a judge. Evidence of a transaction should be such that it could not have been obtained if the transaction did not occur.

A two-way non-repudiation protocol requires that each party obtains evidence of the transaction.

The receiver of a message should obtain some message (or evidence) to pass to a judge which establishes, or authenticates, that the sender did send a message.

The sender of a message should obtain some message (or evidence) to pass to a judge which establishes, or authenticates, that the receiver did receive the message.

A one way non-repudiation protocol need provide only one particular party with the required evidence.

At the highest level of description, despite the differences with authentication, it turns out that non-repudiation is modelled as an authentication property. We see later that the difference comes at a more detailed level of description.

We consider first the case where evidence e is provided as proof by the sender (user i, say) that the other party (user j) received message m. User i should be in a position to input evidence e onto the network only if the message m was previously received by user j. This may be captured as follows:

Definition 2.13 In a transaction between sender i and receiver j, message e provides evidence for i that m was received if $in.i.USER.e$ authenticates $out.j.USER.m$. □

In a similar fashion, the receiver may provide evidence that a message was sent.

Definition 2.14 In a transaction between sender i and receiver j, message e' provides evidence for j that m' was sent if $in.j.USER.e'$ authenticates $in.i.USER.m'$. □

The task in setting up a network in which non-repudiation is ensured is in finding appropriate evidence for messages, in other words for any given message m finding a suitable e. In general, a non-repudiation requirement will not be concerned only with one particular message m and its evidence, but with providing evidence for any of a set M of possible messages. In this case, what is required is a function $e : M \longrightarrow MESSAGE$ such that $e(m)$ provides evidence for m for every $m \in M$.

3 The network

Architecture

A common architecture for which security protocols are designed consists of a network of *nodes* (typically workstations) which are able to communicate asyn-

chronously by sending messages to each other over a *medium*, which acts as a postal service. The need for security arises from the fact that the users of this service (such as people, and applications programs) do not have control over the medium, and so it is possible for malicious agents to intercept or interfere with network traffic. The need for confidentiality in the face of an insecure medium creates the need for some form of encryption, and the need for authenticity when message forgery is possible also raises the need for some form of security mechanism.

A common approach to modelling this situation is to consider a set of nodes as connected to the medium, which is modelled as a single process. Although the medium will in general consist of a network of processes, this network may be considered at a higher level of abstraction as a single process. The only interactions the nodes may have with other nodes must be through this medium. As discussed earlier, we will find it convenient to model malicious interference by means of a separate enemy process node *0* which manipulates the essentially passive medium.

Thus the service provided to the users is modelled as

$$(\ |||_{i \in USER \setminus 0}\ NODE_i) \underset{trans, rec}{\|} MEDIUM$$

The nodes are unable to interact directly, so their operation is entirely interleaved. They all communicate with the medium by means of channels *trans* and *rec*, used by the nodes to transmit and receive messages respectively.

Quite often the distinction between a user and the node by which the user communicates with the network is blurred when addressing security properties. An authentication check that a particular server remains up requires a response directly from that server rather than from the network operator responsible for it. Hence in some cases it is appropriate to think of the user and the node as being the same entity. However, for the purposes of this paper we find it convenient to treat them as distinct.

All forms of interference will be modelled by an intruder process $ENEMY = NODE_0$ that is able to alter the condition of the medium via certain channels not available to the nodes. The entire system will be described by

$$NET \mathrel{\widehat{=}} (\ |||_{i \in USER \setminus 0}\ NODE_i) \underset{trans, rec}{\|} MEDIUM \underset{leak, kill, add}{\|} ENEMY$$

The process NET will always refer to this configuration, though it will generally be parameterised by particular descriptions of the node processes $NODE_i$, and of the processes $MEDIUM$ and $ENEMY$.

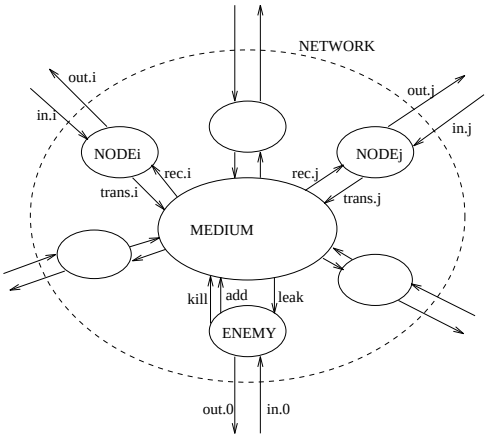

Figure 2 Architecture of the network

Messages

The kind of messages that are transmitted and received will depend upon the particular protocol being modelled, so it is probably best, at least initially, to defer definition of the type of these channels until we come to model a protocol. We can note that each node $NODE_i$ will use channels $trans.i$ and $rec.i$ to interact with the medium, so *trans* and *rec* may be thought of as denoting families of channels rather than single channels. It is also likely that a destination field will be required as part of the message, as well as the message itself and possibly an encryption. It is not clear at this stage how best to handle encrypted messages: in order to maintain the possibility that the number of encryption levels may be arbitrarily large, a recursive data structure will be required, perhaps along the lines of

$$\begin{aligned} MESSAGE ::= \; & PLAINTEXT \\ | \; & KEY \\ | \; & KEY(MESSAGE) \\ | \; & MESSAGE.MESSAGE \end{aligned}$$

and even plaintext messages might have some non-trivial structure:

$$\begin{aligned} PLAINTEXT ::= \; & USER \\ | \; & TEXT \\ | \; & PLAINTEXT.PLAINTEXT \end{aligned}$$

This is not the only structure appropriate for messages. For example, in a key-exchange protocol, keys themselves take on a dual role, being used to encrypt messages, but also comprising the messages to be encrypted. Thus for key-exchange mechanisms, the set KEY should also be included as possible $PLAINTEXT$. Other cryptographic mechanisms such as hash functions may be included as possible messages, in which case the definition of $MESSAGE$ might be extended with two extra lines $HASH$ and $HASH(MESSAGE)$. For the purposes of this paper, we will use the definition of $MESSAGE$ as given above, while remembering that this can be varied according to modelling needs.

It will also prove useful, when considering what an enemy may deduce about messages it has received, to be able to extract the information in messages. An extraction function $kernel$ may be defined by structural induction on $MESSAGE$; and $kernel_0$ defined for $PLAINTEXT$. In the case we have given above, these functions will be defined as follows:

$$\begin{aligned} kernel(p) &= kernel_0(p) \\ kernel(k) &= \{k\} \\ kernel(k(m)) &= kernel(m) \\ kernel(m_1.m_2) &= kernel(m_1) \cup kernel(m_2) \\ kernel_0(u) &= \{u\} \\ kernel_0(t) &= \{t\} \\ kernel_0(p_1.p_2) &= kernel_0(p_1) \cup kernel_0(p_2) \end{aligned}$$

The function $kernel$ lifts to sets in the obvious way.

Message properties

An intruder is able to manipulate the medium in particular ways. The approach taken here of using events to signal particular modes of interference (in preference to having them occur nondeterministically) was originally taken in [Ros94]. The advantage of this approach is that it allows greater control over the level and type of interference that may occur.

However, the enemy is not capable of producing all messages: for example, it cannot generate a message encrypted with a key it does not have (though of course it could reproduce such a message if it had previously received it).

In fact, the messages the intruder is able to generate will depend on the messages it has already seen pass as network traffic, the messages it is already able to generate, and the keys it has seen or which it owns.

We will use an information system [Ros86] to define which messages can be generated by the enemy. It will have a trivial consistency relation: any set of messages is consistent. The definition of the relation $\vdash$ of the information system will be dependent on, and should encapsulate the encryption mechanism. An information system defines a relation $\vdash$ between finite sets of *tokens* and single tokens, indicating when the token can be generated from the set. In this case, we will use the relationship to indicate when the enemy, or indeed any other agent, can generate a particular message given the messages it has already seen.

Consider an example in which messages may be encrypted by means of either secret keys or public keys.

There will be the set $PUBLIC$ of all the nodes' public keys—for simplicity we assume one for each node

$$PUBLIC = \{p_i \mid i \in USER\} \subseteq KEY$$

There will also be the set S of all the nodes' secret keys—one for each node

$$SECRET = \{s_i \mid i \in USER\} \subseteq KEY$$

This set is distinct from the set of public keys:

$$SECRET \cap PUBLIC = \{\}$$

Finally there will be a set of shared keys $SHARED$, distinct both from public and secret keys:

$$SECRET \cap SHARED = \{\}$$

$$PUBLIC \cap SHARED = \{\}$$

The entailment relation $\vdash$: $\mathbb{P}_{fin}(MESSAGE) \times MESSAGE$ will be a relation between a finite set of messages (that we think of as the enemy having seen) and messages that the enemy can generate. The relation is closed under the axioms for an information system:

A1. If $m \in B$ then $B \vdash m$
A2. If $B \vdash m$ and $B \subseteq B'$ then $B' \vdash m$
A3. If $B \vdash m_i$ for each $m_i \in B'$ and $B' \vdash m$ then $B \vdash m$

We will abuse notation and allow the relation between possibly infinite sets and messages:

$$S \vdash m \Leftrightarrow \exists\, T \subseteq^{fin} S \bullet T \vdash m$$

We encapsulate the way in which messages can be generated by considering the possible structures for a message:

M1. $B \vdash m \wedge B \vdash k \;\Rightarrow\; B \vdash k(m)$
M2. $B \vdash m_1 \wedge B \vdash m_2 \;\Leftrightarrow\; B \vdash m_1.m_2$

where m, m_1 and m_2 are messages, and k is a key.

Certain properties of particular encoding mechanisms may also be captured by providing additional inference rules. For example, the relationship between secret and public keys may be captured by the following pair of rules:

K1. $\{p_i(s_i(m))\} \vdash m$
K2. $\{s_i(p_i(m))\} \vdash m$

where $p_i \in P$ and $s_i \in S$.

For example, these rules allow us to deduce the obvious result that possession of a message encrypted with a secret key ($s_i(m)$, say), together with possession of the public key, allows the original message to be retrieved:

1.	$\{p_i, s_i(m)\}$	$\vdash$	$s_i(m)$	A1
2.	$\{p_i, s_i(m)\}$	$\vdash$	p_i	A1
3.	$\{p_i, s_i(m)\}$	$\vdash$	$p_i(s_i(m))$	M1, 1, 2
4.	$\{p_i, s_i(m)\}$	$\vdash$	m	3, K1, A3

The appropriate rule for shared keys is that possession of a shared key together with a message encrypted with that key allows generation of the original message:

K3. $\{k, k(m)\} \vdash m$ if $k \in SHARED$

It is also possible to encode various other deductions we might wish to include in the capability of the enemy, for example deducing a key from observing both an encoded message and that message in plaintext:

K4. $\{m, k(m)\} \vdash k$

The rules that we give model different encryption and decryption capabilities of the enemy.

The rules can also be used to encapsulate properties of encryption. For example, if encryption were commutative, then we could include the rule

K5. $\{k_1, k_2(m)\} \vdash k_2(k_1(m))$

Medium

The description of *MEDIUM* involves a number of decisions about the best way to model the network medium.

We must allow the possibility of an intruder, who is able to manipulate the medium in particular ways. This could be done by building the intruder into the medium (so the medium itself has the capability of interfering with message traffic in particular ways); but we prefer to follow Roscoe's approach [Ros94] of including a separate model of the intruder. This second approach gives greater separation between the medium itself, which would then be considered as essentially a passive service provided to the various nodes, and a malicious agent who has particular capabilities to manipulate the medium in particular ways. The capabilities of this agent will be made more explicit, and manipulation of the medium will be associated with particular events, which will make attacks on protocols easier to follow and understand.

The medium (containing the set of messages B) may be described initially as $MEDIUM(\{\})$, where:

$$\begin{aligned} MEDIUM(B) \quad = \quad & INPUT(B) \\ & \Box \\ & OUTPUT(B) \\ & \Box \\ & IA(B) \end{aligned}$$

The process $INPUT(B)$ permits input to the medium. We must decide on the type of messages that the medium will accept and offer. For the purposes of this paper, we will separate out the destination and source from the body of the message. Again there are other possibilities, for example, if the message is to be broadcast to all users then no explicit destination field is required.

$$INPUT(B) \quad = \quad \Box_i \, trans.i?j?x \longrightarrow MEDIUM(B \cup \{i.j.x\})$$

Here the channel *trans* is of type $USER.USER.MESSAGE$. A message $trans.i.j.m$ should be thought of as node i sending an input $j.m$ to the medium, indicating the wish that message m be delivered to node j. Thus i is the source, j the destination, and m the message.

We have abstracted away from refusals, in the sense that input can never be refused, which amounts to making the assumption that nothing can be deduced from how or when the messages are accepted. This is a reasonable assumption, since there are protocols currently in use to perform tasks such as masking network traffic. Hence at this level of abstraction we can assume that messages are always accepted by the network[3].

The process $OUTPUT$ allows output from the medium:

$$OUTPUT(B) \quad = \quad \Box_{i.j.x \in B} \, rec.j!i!x \longrightarrow MEDIUM(B \setminus \{i.j.x\})$$

Here the channel rec is of type $USER.USER.MESSAGE$. A message $rec.j.i.m$ corresponds to the receipt of a message m by node j which is labelled as coming from source node i.

Note that an empty external choice is simply equivalent to $STOP$, so when the set B is empty (i.e. the medium contains no messages) there is no possibility of output.

Finally, the process $IA(B)$ describes the possible interactions with the medium due to Intruder Actions. A perfectly secure medium would treat this part of the process description as $STOP$. In cases we are considering, we model the ways in which the medium is susceptible to interference. Here, the medium is vulnerable to having messages removed, added, or leaked.

$$\begin{array}{lcll} IA(B) & = & kill \longrightarrow \Box_{b \in B} MEDIUM(B \setminus \{b\}) & \text{if } B \neq \{\} \\ & & \quad MEDIUM(\{\}) & \text{if } B = \{\} \\ & & \Box & \\ & & add?i?j?x \longrightarrow MEDIUM(B \cup \{i.j.x\}) & \\ & & \Box & \\ & & \Box_{i.j.x \in B} \, leak!i!j!x \longrightarrow MEDIUM(B) & \end{array}$$

Enemy action

In modelling the enemy we are concerned with messages that the enemy is able to generate. These may be used to disrupt a protocol, or may correspond to information about what the enemy has discovered concerning supposedly confidential messages.

Certain assumptions may be made concerning the enemy, depending on the property that is under analysis. When checking confidentiality, it is often assumed that the enemy is unable to generate those messages M (which can be generated by the users) that should be kept confidential. On the other hand, when checking authentication it should be assumed that the enemy (as well as the honest users) is capable of generating those messages whose authenticity is ensured by the protocol, since if the enemy is unable to generate them then there is no need for an authentication protocol. For integrity, it is assumed that the enemy is capable of generating messages from M.

These assumptions may be incorporated into the description of the enemy, which may then be parameterised by a set of messages S that it has seen, and a set of messages $INIT$ that it is initially able to generate. The assumptions can be expressed as conditions on $INIT$ and on the set of messages M which the particular security property is concerned with.

[3] If this is later felt to be unrealistic, the definition of $INPUT$ can be altered accordingly (so that messages may not be input after the number of messages in the network reaches some capacity threshold)

The question also arises as to whether it is sufficient to model enemy action using a single $ENEMY$ process. In principle it is possible that a number of malicious agents acting together might effect an attack where a single agent would be unable to do so. Whether or not this is possible in the model being developed here depends on the actual description of the process $ENEMY$. In fact, the description to be presented enjoys the property that

$$ENEMY = ENEMY \parallel ENEMY$$

Hence for all analysis done at the level of traces we see that any number of enemies acting together are encapsulated within the description of a single enemy.

In addition to the messages that can be generated from those messages already seen, the enemy is able to generate particular plaintext messages. Furthermore, the enemy should be considered to be in possession of all of the nodes' public keys, and all of the users' names. We therefore use a set $INIT$ to model all of the information initially in the possession of the enemy. Thus we have $PUBLIC \subseteq INIT$ and $USER \subseteq INIT$. The relation $\vdash$ gives the capability of the enemy to generate messages from messages already in its possession. The CSP description of the enemy will use this relation. The set S records those messages that have been read from the medium. This is initialised to $\{\}$, so $ENEMY$ is defined to be $ENEMY(\{\})$, where

$$\begin{aligned} ENEMY(S) \quad = \quad & KILL(S) \\ & \Box \\ & ADD(S) \\ & \Box \\ & LEAK(S) \\ & \Box \\ & KNOWS(S) \end{aligned}$$

The first option allows the enemy to kill a message—to remove it from the medium. It is described simply as

$$KILL(S) \quad = \quad kill \longrightarrow ENEMY(S)$$

In fact, when dealing with trace properties of communication protocols, the ability of the enemy to kill messages is entirely irrelevant. Although the possible removal of messages from the medium can interfere with liveness properties of communications protocols, it cannot compromise properties expressed only in terms of traces. This is because the medium allows the reordering of messages, so any particular message could always be ignored and remain in the medium without being killed. Any protocol which guarantees a security property if the enemy is unable to kill messages will therefore guarantee it in any case. An equally useful definition of $KILL(S)$ would be $STOP$ (which would be equivalent to omitting this option entirely).

The second course of action available to the enemy is to insert any message that it can generate onto the medium. These are any messages that can be generated from its initial set $INIT$ together with the messages S that have since come into its possession.

$$ADD(S) \quad = \quad \Box_{INIT \cup S \vdash x} \; add!i!j!x \longrightarrow ENEMY(S)$$

Observe that this description incorporates the ability of the enemy to manipulate message address fields, thus giving the impression that a message comes from a source other than the genuine source.

If the enemy is considered to be simply an eavesdropper with no power to add messages to the medium, then the ADD component would simply be modelled as $STOP$ (or omitted entirely).

The third option allows it to observe any message currently on the medium:

$$LEAK(S) \quad = \quad leak?i?j?x \longrightarrow ENEMY(S \cup \{x\})$$

The final option is included to model the enemy's knowledge of particular messages. This is accomplished by allowing the enemy to output any message that can in fact be generated.

$$KNOWS(S) \quad = \quad \Box_{INIT \cup S \vdash x} \, out.0.0!x \longrightarrow ENEMY(S)$$

The channel $out.0$ is used to indicate those messages that the enemy can deduce from what has already been seen together with what was known initially.

The argument S represents the set of messages that the enemy has already seen. Normally this will be the empty set at the beginning of a protocol run, but it is possible to model the effect that possession of a particular key might have on the vulnerability of a protocol, by including such a key, or some other message, in the set S.

Observe that we have allowed the insertion of any message into the medium, so in particular false sources can be attached to messages. Rerouting of a message can also be modelled, by having the enemy read it via leak, kill it (this is cleaner though not essential, as already discussed), and then add the same message with a different destination field back to the medium.

Now the assumption that is made in the case of confidentiality can be formalised. We are assuming that none of the messages that we wish to keep confidential are in fact in the kernel of the messages that can initially be generated by the enemy:

$$M \cap kernel(INIT) = \{\}$$

On the other hand, for integrity and authenticity, we are (implicitly) assuming that

$$M \cap INIT \neq \{\}$$

in the sense that protocols designed to provide these services are intended to deal with messages that can be generated by an enemy.

When checking a confidentiality protocol, strong use is made of this assumption, since if the enemy can output a message that is supposed to be confidential then the protocol is considered to be insecure. However, there are situations such as key-exchange where a protocol is designed to provide both confidentiality and authenticity, in which case it is reasonable to begin analysis with $M \cap INIT \neq \{\}$. In such situations, the above modelling of what the enemy knows is not adequate, and it would be necessary to construct a more sophisticated, complex model of the enemy which keeps track of incoming and outgoing messages and outputs on $out.0$ only those messages it deduces have been generated by the legitimate users, in particular ignoring those messages in M that it puts onto the medium and then reads back via $leak$. We will not pursue this further in this paper, but will observe that it is a situation to bear in mind.

Nodes

We must consider the nodes—which are the link between the user and the medium—to be under the control of the user. It is the nodes that will provide the security facilities required by the users, such as encrypting and deciphering messages.

The (finite) set of all nodes will be labelled using the set $USER = \{0, 1, \ldots, n\}$.

The nodes provide the means by which users send messages over the network. A user communicates with the network is in fact communicating with the corresponding node. Nodes interact with users by inputting plaintext messages with intended destinations, and outputting such messages together with their source. The process $NODE_i$ thus communicates with its user via channels $in.i$ and $out.i$ of type $USER.PLAINTEXT$. An input $in.i.j.m$ to node $NODE_i$ is interpreted as a request from user i to send message m to user j. Similarly, an output communication $out.i.j.m$ is interpreted as delivery to user i of message m purporting to come from j.

Nodes (with the exception of node 0) interact with the medium by transmitting (possibly enciphered) messages together with other control messages, intended recipients, and any other messages employed by the protocol being used. The channels used are $trans.i$ for transmission, and rec.i for receipt of messages. These channels are of type $USER.MESSAGE$, where the set $MESSAGE$ contains both plain and encrypted messages (as discussed later). A communication $trans.i.j.m$ corresponds to $NODE_i$ placing message m with destination j onto the medium. A communication $rec.i.j.m$ corresponds to $NODE_i$ receiving message m from the medium, with source purporting to be j.

The description of a $NODE_i$ process will depend on the security property we are aiming to verify of the network. For confidentiality, authenticity, anonymity and integrity its description will consist of a CSP implementation of the particular protocol under analysis. For example, an extremely simple protocol to provide confidentiality of messages sent from user 1 to user 2 will be implemented using $NODE_2$'s public key p_2 and secret key s_2 as follows:

$$NODE_1 = in.1.2?x \longrightarrow trans.1!2!p_2(x) \longrightarrow NODE_1$$

$$NODE_2 = rec.2?j?y \longrightarrow \begin{array}{ll} STOP & \text{if } s_2(y) \notin PLAINTEXT \\ out.2!j!s_2(y) \longrightarrow NODE_2 & \text{otherwise} \end{array}$$

where $s_2(p_2(x)) = x$ for any message x. Observe that this protocol does not ensure authenticity.

The situation is different in the case of non-repudiation. In this case, verification is from the judge's viewpoint, and the judge does not have control over the nodes used in a non-repudiation protocol. In fact, from the judge's viewpoint, the parties could each be dishonest. Indeed, it is this possibility that generates the need for a non-repudiation protocol in the first place.

The judge has to allow for the possibility that each node has the capabilities of node 0. Thus non-repudiation has to be established in the context of nodes which can *kill*, *add*, and *leak* messages as well as interact with the medium in the usual ways. The nodes for which non-repudiation should be established are therefore

$$NODE_i(M) = in.i?j?x \longrightarrow NODE_i(M \cup \{x\}) \;\Box\; \Box_{INIT \cup M \vdash x}\; out.i!j!x \longrightarrow NODE_i(M) \;\Box$$

$$\Box_{INIT \cup M \vdash x} trans.i!j!x \longrightarrow NODE_i(M)$$
$$\Box$$
$$rec.i?j?x \longrightarrow NODE_i(M \cup \{x\})$$
$$\Box$$
$$kill \longrightarrow NODE_i(M)$$
$$\Box$$
$$\Box_{INIT \cup I \vdash x} add!i!j!x \longrightarrow NODE_i(M))$$
$$\Box$$
$$leak?l?j?x \longrightarrow NODE_i(M \cup \{x\})$$

And from a modelling point of view the interfaces of these processes with the network must be expanded to include the channels *add*, *leak* and *kill*.

Since the node is able to generate its own plaintext messages, the $in.i$ channel is perhaps redundant, but is retained as a source of messages so that particular non-repudiation protocols will refine this node.

The set of messages M corresponding to the initial state of the node will contain all of the keys which the node may use to encrypt and decrypt messages.

Meadows' example

In order to illustrate the above material, we will present a simple example used in [Mea92] and [GrM95]. It is not even a protocol, but is instead a simple example designed purely for illustrative purposes. In fact, it is not the kind of example that best illustrates the benefits of the process algebra approach, since process algebra would be of more use in exploring subtle patterns of interactions between different parties; here the interactions are fairly simple. However, it illustrates the approach to proof. Although the proof of such an obvious property seems unduly long, it is also lengthier than might be expected in [Mea92] and [GrM95]. This is because a significant amount of formalisation needs to be done before the proof can actually proceed,

The example consists of a legitimate user who encrypts received messages with a particular key, and returns them to the medium. This could be described as a legitimate node (number *1* for definiteness) which receives messages on *rec.1*, encrypts them, and returns them on *trans.1*. The process algebra is as follows:

$$NODE_1 \quad = \quad rec.1?j?x \longrightarrow trans.1!j!k(x) \longrightarrow NODE_1$$

where k is a key possessed by $NODE_1$.

The aim is to establish that the enemy cannot obtain a particular message a that it does not already possess. This is expressed as confidentiality with respect to a:

$$NET = NET \underset{out.0.0.a}{\|} STOP$$

or alternatively as

$$NET \textbf{ sat } (tr \upharpoonright out.0.0.a = \langle\rangle)$$

We make the standard assumption for confidentiality, that the enemy is not in possession of any messages containing a:

$$KERNEL(a) \not\subseteq kernel(INIT)$$

We may take the description of NET to consist of the node $NODE_1$, the initially empty medium $MEDIUM(\{\})$, and the enemy who has initially learned nothing: $ENEMY(\{\})$.

It will prove useful to extract certain sets of messages from traces of the system:

$$\begin{aligned}
LEAK(tr) &= \{m \mid \exists\, i,j \bullet tr \upharpoonright leak.i.j.m \neq \langle\rangle\} \\
ADD(tr) &= \{m \mid \exists\, i,j \bullet tr \upharpoonright add.i.j.m \neq \langle\rangle\} \\
TRANS_k(tr) &= \{m \mid \exists\, j \bullet tr \upharpoonright trans.k.j.m \neq \langle\rangle\} \\
REC_k(tr) &= \{m \mid \exists\, j \bullet tr \upharpoonright rec.k.j.m \neq \langle\rangle\} \\
OUT_0(tr) &= \{m \mid tr \upharpoonright out.0.0.m \neq \langle\rangle\} \\
MESS(tr) &= LEAK(tr) \cup ADD(tr) \cup TRANS_1(tr) \cup REC_1(tr) \cup OUT_0(tr)
\end{aligned}$$

Lemma 3.1 The kernel function is closed under the generates relation, i.e.

$$B \vdash m \Rightarrow kernel(m) \subseteq kernel(B)$$

□

Proof By considering all of the clauses that define the relation $\vdash$: A1, A2, A3, M1, M2, K1, K2, K3, and K4. The result follows for each clause, so it is true for the relation. □

In order to prove confidentiality of NET with respect to a we will use certain properties of its components. The required properties are described in the following lemma. They combine information about the state of the components (as maintained in S and B) and events that have occurred (extracted from the trace). A combination of information from both these sources is often required in establishing this kind of result. State-based approaches commonly include a 'history' variable as a component of the state in order to record trace information. The approach taken here is closer to event-based approaches which provide some way of extracting the state of the system from its trace.

Lemma 3.2 The component processes meet the following specifications:

$$\begin{aligned}
ENEMY(S) \quad &\textbf{sat} \quad E1_S(tr) = kernel(ADD(tr)) \subseteq kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr)) \\
ENEMY(S) \quad &\textbf{sat} \quad E2_S(tr) = kernel(OUT_0(tr)) \subseteq kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr)) \\
MEDIUM(B) \quad &\textbf{sat} \quad M1_S(tr) = kernel(LEAK(tr)) \subseteq kernel(B) \cup kernel(TRANS_1(tr) \cup kernel(ADD(tr)) \\
MEDIUM(B) \quad &\textbf{sat} \quad M2_S(tr) = kernel(REC(tr)) \subseteq kernel(B) \cup kernel(TRANS_1(tr) \cup kernel(ADD(tr)) \\
NODE_1 \quad &\textbf{sat} \quad N1_S(tr) = kernel(TRANS_1(tr)) \subseteq kernel(REC_1(tr))
\end{aligned}$$

□

Proof In each case by a recursion induction on the definition of the process. We will provide a proof of $E1$ as an illustration.

The definition of $ENEMY$ is given as the unique fixed point of a guarded function on a vector of processes indexed by all sets of messages S. The function may be given as

$$\begin{aligned}
F(\vec{X})_S &= kill \longrightarrow X_S \\
&\quad \square
\end{aligned}$$

$$\Box_{INIT \cup S \vdash x}\ add!i!j!x \longrightarrow X_S$$
$$\Box$$
$$leak?i?j?x \longrightarrow X_{S \cup \{x\}}$$
$$\Box$$
$$\Box_{INIT \cup S \vdash x}\ out.0.0!x \longrightarrow X_S$$

We wish to prove that each component of the fixed point of F meets the corresponding component of the vector of specifications $\vec{SPEC}$ given by

$$SPEC_S(tr) = kernel(ADD(tr)) \subseteq kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr))$$

We have to show (1) that this vector of specifications is satisfiable, and (2) that it is preserved by the function F.

(1)It is clearly satisfiable, since $\vec{STOP}$ satisfies it.

(2) To prove that $\vec{SPEC}$ is preserved by F, we begin by assuming that $\vec{X}$ **sat** $\vec{SPEC}$. We have to prove that each possible course of action for $F(\vec{X})_S$ satisfies $SPEC_S$.

The rule for external choice is

$$\frac{P_a \textbf{ sat } T_a(tr)}{\Box_{a \in A}\ a \longrightarrow P_a \textbf{ sat } \begin{array}{l} tr = \langle\rangle \\ \vee \\ tr = a^\frown tr' \wedge a \in A \wedge T_a(tr') \end{array}}$$

We will use it for each branch in turn.

We first deduce that

$$kill \longrightarrow X_S \quad \textbf{sat} \quad \begin{array}{l} tr = \langle\rangle \\ \vee \\ tr = kill^\frown tr' \wedge SPEC_S(tr') \end{array}$$

A trace tr meeting the second disjunct of this specification has $ADD(tr) = ADD(tr')$ and $LEAK(tr) = LEAK(tr')$; it follows from $SPEC_S(tr')$ that $SPEC_S(tr)$. Furthermore, a trace meeting the first disjunct (i.e. $tr = \langle\rangle$) has $SPEC_S(tr)$. Hence

$$kill \longrightarrow X_S \quad \textbf{sat} \quad SPEC_S(tr)$$

The second choice yields

$$\Box_{INIT \cup S \vdash x}\ add!i!j!x \longrightarrow X_S \quad \textbf{sat} \quad \begin{array}{l} tr = \langle\rangle \\ \vee \\ tr = add.i.j.x^\frown tr' \wedge INIT \cup S \vdash x \wedge SPEC_S(tr') \end{array}$$

Observe that for traces tr meeting the second disjunct of this specification $kernel(ADD(tr)) = kernel(ADD(tr')) \cup kernel(x)$. Now $kernel(ADD(tr')) \subseteq kernel(S) \cup kernel(INIT) \cup LEAK(tr')$ since $SPEC_S(tr')$. Also $LEAK(tr') = leak(tr)$, and $kernel(x) \subseteq kernel(S) \cup kernel(INIT)$ (since $INIT \cup S \vdash x$). Hence it follows that

$$kernel(ADD(tr)) \subseteq kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr))$$

Since the other possibility (that $tr = \langle\rangle$) also yields $SPEC_S(tr)$, we therefore conclude

$$\Box_{INIT \cup S \vdash x}\ add!i!j!x \longrightarrow X_S \quad \textbf{sat} \quad SPEC_S(tr)$$

The third choice yields

$$leak?i?j?x \longrightarrow X_{S\cup\{x\}} \quad \textbf{sat} \quad \begin{array}{l} tr = \langle\rangle \\ \vee \\ tr = leak.i.j.x^\frown tr' \wedge SPEC_{S\cup\{x\}}(tr') \end{array}$$

A trace tr meeting the second disjunct of this specification has

$$\begin{array}{rcl} ADD(tr) & = & ADD(tr') \\ & \subseteq & kernel(S \cup \{i.j.x\}) \cup kernel(INIT) \cup kernel(LEAK(tr')) \\ & = & kernel(S) \cup kernel(\{i.j.x\})) \cup kernel(INIT) \cup kernel(LEAK(tr')) \\ & = & kernel(S) \cup kernel(INIT) \cup kernel(LEAK(\langle i.j.x\rangle^\frown tr')) \\ & = & kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr)) \end{array}$$

In other words, $SPEC_S(tr)$ holds. Since $SPEC_S(tr)$ also holds in the case where the first disjunct is true, it follows that

$$leak?i?j?x \longrightarrow X_{S\cup\{i.j.x\}} \quad \textbf{sat} \quad SPEC_S(tr)$$

Finally, the fourth choice falls to the same sort of analysis as the second, and yields

$$\Box_{INIT\cup S\vdash x}\ out.0.0!x \longrightarrow X_S \quad \textbf{sat} \quad SPEC_S(tr)$$

We now finish by applying the rule for choice three times:

$$\frac{\begin{array}{l} P \textbf{ sat } T(tr) \\ Q \textbf{ sat } T(tr) \end{array}}{P \,\Box\, Q \textbf{ sat } T(tr)}$$

to obtain

$$\begin{array}{l} kill \longrightarrow X_S \\ \Box \\ \Box_{INIT\cup S\vdash x}\ add!i!j!x \longrightarrow X_S \\ \Box \\ leak?i?j?x \longrightarrow X_{S\cup\{x\}} \\ \Box \\ \Box_{INIT\cup S\vdash x}\ out.0.0!x \longrightarrow X_S \end{array} \quad \textbf{sat} \quad kernel(ADD(tr)) \subseteq kernel(S) \cup kernel(INIT) \cup kernel(LEAK(tr))$$

Since this is true for arbitrary S, we conclude that $F(\vec{X})$ **sat** $\vec{SPEC}$. □

Define $E1(tr) = E1_{\{\}}(tr)$, and similarly for $E2$, $M1$, $M2$ and $N1$. Each component of NET behaves according to the corresponding specifications of these five. The components therefore constrain the behaviour of the entire process NET, since they will only allow particular events to occur at certain times.

The following rule may be used to establish that the specifications met by the components are also met by the system as a whole.

$$\frac{\begin{array}{l} S(tr) \Leftrightarrow S(tr \restriction A) \\ P \textbf{ sat } S(tr) \\ Q \textbf{ sat } tr \restriction A \setminus B = \langle\rangle \end{array}}{P \underset{B}{\parallel} Q \textbf{ sat } S(tr)}$$

The antecedents state that (1) the truth of S depends only on those events from A; (2) the process P meets S; (3) process Q is unable to perform any events from A that are not also in B. These together mean that when P is placed in parallel with Q interacting on events from B, then P is involved with every occurrence of every event from A (since any event from A that Q can perform is also in B, and hence P also engages in it), and so P ensures that the combination meets $S(tr)$.

In the case of NET, it is easiest first to consider the combination $NODE_1 \underset{\{\}}{\|} ENEMY$.

We firstly consider the case of $E1(tr)$. In applying the rule, we have $A = \{add, leak\}$ and $B = \{\}$. Then we obtain the three antecedents to the rule: $E1(tr) = E1(tr \upharpoonright \{add, leak\})$, $ENEMY$ **sat** $E1(tr)$, and $NODE_1$ **sat** $tr \upharpoonright A \setminus B = \langle\rangle$, and hence we conclude

$$NODE_1 \underset{\{\}}{\|} ENEMY \textbf{ sat } E1(tr)$$

Using similar reasoning, we obtain

$$NODE_1 \underset{\{\}}{\|} ENEMY \textbf{ sat } E2(tr)$$

and

$$NODE_1 \underset{\{\}}{\|} ENEMY \textbf{ sat } N1(tr)$$

Now we consider the effect of placing this combination in parallel on all events with $MEDIUM$. In fact, this combination is equivalent to NET:

$$NET = (NODE_1 \underset{\{\}}{\|} ENEMY) \underset{\Sigma}{\|} MEDIUM$$

In this case, the first and third antecedents to the rule are trivially satisfied. Hence any specification met by either component (providing an instance of the second antecedent) will be met by the combination. Thus we deduce that each of the specifications $E1$, $E2$, $M1$, $M2$ and $N1$ hold for NET.

We are now in a position to establish the theorem which asserts correctness of the example.

Theorem 3.3 The network is secure:

$$NODE_1 \underset{trans.1, rec.1}{\|} MEDIUM \underset{leak, add, kill}{\|} ENEMY \textbf{ sat } tr \upharpoonright out.0.0.a = \langle\rangle$$

□

Proof The strategy of the proof is as follows: we will actually prove that the kernels of all messages passed around the system must be contained in the kernel of $INIT$. Since it is given that this is not true for a, it follows that a can never be passed along channel $out.0$.

The proof begins by assuming for a contradiction that $out.0.0.a$ is possible for some trace of NET. It follows that there must have been a point in the trace where the kernel of a was first contained in the kernel of the messages passed around.

We focus on the trace tr which is the trace of the system up to that point, and establish that there is no possible last message for this trace: every possibility leads to a contradiction.

We assume for a contradiction that NET does not meet the specification above. Then there is some trace tr_0 of NET for which $tr_0 \upharpoonright out.0.0.a \neq \langle\rangle$. Let tr be the longest prefix of tr_0 such that $kernel(MESS(init(tr))) \subseteq kernel(INIT)$, (where $init(tr)$ is trace tr with its last message removed) but $kernel(MESS(tr)) \not\subseteq kernel(INIT)$. The trace tr is well defined, since if this property is not true for any prefix of tr_0 then it is still true for tr_0: $kernel(a) \subseteq kernel(MESS(tr_0))$ but $kernel(a) \not\subseteq kernel(INIT)$, and so $kernel(MESS(tr_0)) \not\subseteq kernel(INIT)$.

Now there are a number of possibilities for the last message in tr: it could be a message along one of six channels: $leak$, add, $trans$, rec, out, $kill$. We consider each of these in turn.

If the last message in tr is $leak.i.j.x$, then we have from specification 1 (with $B = \{\}$) that

$$kernel(LEAK(tr)) \subseteq kernel(TRANS_1(tr) \cup kernel(ADD(tr))$$

and since $TRANS_1(init(tr)) = TRANS_1(tr)$ and $ADD(init(tr)) = ADD(tr)$ we obtain

$$\begin{aligned}
kernel(MESS(tr)) &= kernel(MESS(init(tr))) \cup kernel(MESS(\langle leak.i.j.x\rangle)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(LEAK(tr)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(TRANS_1(tr)) \cup kernel(ADD(tr)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(TRANS_1(init(tr))) \cup kernel(ADD(init(tr))) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(MESS(init(tr))) \\
&\subseteq kernel(INIT)
\end{aligned}$$

but this contradicts the definition of tr. Hence the last message in tr cannot be a $leak$ event. Entirely similar reasoning applies to rec (specification 1) and $trans$ (specification 1).

The next two cases differ slightly from the preceding ones. If the last event in tr is a communication of the form $add.i.j.x$, then we have from specification 1 (with $S = \{\}$) that

$$kernel(ADD(tr)) \subseteq kernel(INIT) \cup kernel(LEAK(tr))$$

Now $kernel(LEAK(tr)) = kernel(LEAK(init(tr)))$, so

$$\begin{aligned}
kernel(MESS(tr)) &= kernel(MESS(init(tr))) \cup kernel(MESS(\langle add.i.j.x\rangle)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(ADD(tr)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(INIT) \cup kernel(LEAK(tr)) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(INIT) \cup kernel(LEAK(init(tr))) \\
&\subseteq kernel(MESS(init(tr))) \cup kernel(INIT) \\
&\subseteq kernel(MESS(init(tr)))
\end{aligned}$$

and this again contradicts the definition of tr. Hence the last message in tr cannot be an add event. Entirely similar reasoning applies to $out.0$ (specification 1).

The final possibility is that the last event in tr is a $kill$ event. But in this case $MESS(tr) = MESS(init(tr))$, which contradicts the definition of tr, again yielding a contradiction.

Hence the initial assumption that there is some trace tr_0 with $tr_0 \upharpoonright out.0.0.a$ cannot be true, yielding the required result. □

The proof could easily be adapted to take other nodes into account. In fact, descriptions of the other nodes are not even necessary, all that is required of them is that they meet some particular specification: for example, that they do not transmit any messages whose kernel intersects with that of a:

$$NODE_j \quad \textbf{sat} \quad kernel(TRANS_j(tr)) \cap kernel(a) = \{\}$$

Of course, more complex specifications might be more appropriate, for example that the messages added to the network by the nodes do not intersect with a (though those that were passed to the node may be passed back):

$$NODE_j \quad \textbf{sat} \quad kernel(TRANS_j(tr)) \cap (kernel(a) \setminus kernel(REC_j(tr))) = \{\}$$

This latter specification is in fact met by $NODE_1$.

4 Discussion

This paper has been concerned with the expression of particular security properties and protocols within the framework of CSP, in order to provide a foundation for analysis and verification. This approach is motivated in part by the availability of model-checking tools such as FDR[FSEL94], and the work has always proceeded with an eye on applicability of these tools. However, it is inevitable that there will be some difficulties, and it may be necessary to adapt some of the properties. In such cases, we will have to establish that the properties we are checking do indeed correspond to the properties presented here, or at least that results obtained by application of the tools allow us make the inferences we require.

For example, the sets $PLAINTEXT$ and $MESSAGES$ will be infinite, even when the base sets are very small, which makes them unsuitable for direct analysis by means of model-checking by means of current[4] state-of-the-art technology, though the situation will improve as value-passing is introduced. Techniques such as Skolemisation (deducing results concerning all messages from verifications on place-holders) might also be appropriate here. In any case some simplifications will have to be made (perhaps concerning the maximum number of encryptions) in order to regain finitude of the message space; and some additional justification will then be required to derive general correctness from correctness under these assumptions. This should not present any problems, since the protocols themselves will only perform encryptions to a certain level (generally no more than two) and so any interference involving deeper levels will be detected in any case. But nevertheless it will be necessary to prove that the imposition of a bound does not rule out any attacks on a protocol, in order to have confidence in the results of the analysis.

The modelling of the enemy as a separate process allows for the possibility of introducing tactics in the state space exploration when model-checking, for example by restricting the number of messages that the enemy will place on the medium. By accompanying enemy interference with the performance of events, we may introduce tactics by introducing further constraints in parallel, refining the system. This may prove useful when attempting to detect flaws, since a flaw in a refinement will be a flaw in the original system, but correctness of refined protocols does not imply that

[4] June 1995

the original one is correct unless it can be demonstrated that the introduction of the tactic does not rule out any possible attacks.

Non-repudiation appears to be a completely different kind of property. Each party in a non-repudiating exchange of messages is concerned that the other might not be honest. Furthermore, it is not enough for each party to be satisfied that the other party received the required messages; each party aims to obtain evidence sufficient to convince an outside party that the exchange took place.

Meadows' example appears to be particularly straightforward (which is what makes it a good example for comparing different approaches) because the proof rests on the fact that at no stage is the information required to generate the message a ever introduced into the system; the invariant for the system is therefore fairly straightforward, and does not rely particularly on encryption and decryption properties, but simply on the property that no generation of messages by the $\vdash$ relation can introduce new information. It will be harder to find suitable invariants for scenarios where information is present in encrypted form (such as communication between two users via a shared key), where it will be necessary to prove that at no stage could it ever be decrypted. More subtle properties of encryption and decryption will be required.

It seems disappointing that such a simple example as Meadows' still requires a lengthy proof. However, part of the point of doing such a proof is to explore the relationship between the language-theoretic ideas underpinning it and the invariant of the CSP recursive description. It seems likely that this relationship will be similar in many proof of this type, and we would hope to obtain theorems which allow results concerning the language of messages that can be generated to be translated immediately to the CSP setting without the need for a laborious manual translation. A close relationship between CSP protocol descriptions and rules for generating messages would allow more natural proofs. Once this is achieved we would expect the result that a particular set of rules cannot generate any message containing a to translate immediately into the result that the corresponding CSP description has the required confidentiality property.

Acknowledgements

Thanks are particularly due to Peter Ryan, and also to Richard Moore, Paul Gardiner, Bill Roscoe, Gavin Lowe, Michael Goldsmith and Andreas Fuchsberger for much lively discussion and perceptive comments on earlier versions of this work.

Thanks also to DRA Malvern for providing funding for this research.

References

[FSEL94] Formal Systems (Europe) Ltd, *Failures Divergences Refinement User Manual and Tutorial,* 1994.

[GrM95] J.W. Gray and J. McLean, *Using temporal logic to specify and verify cryptographic protocols (progress report),* Proceedings of the eighth IEEE Computer Security Foundations Workshop, 1995.

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

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

[Ros86] A.W. Roscoe, *Lecture Notes on Domain Theory,* Oxford University, 1986.

[Ros94] A.W. Roscoe, *Prospects for describing, specifying and verifying key-exchange protocols in CSP and FDR*, Formal Systems (Europe) Ltd, 1994.