Verifying Authentication Protocols in CSP

Steve Schneider, *Member, IEEE Computer Society*

Abstract—This paper presents a general approach for analysis and verification of authentication properties using the theory of Communicating Sequential Processes (CSP). The paper aims 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 ability to reason formally about the properties they exhibit. The theory is illustrated by an examination of the Needham-Schroeder Public-Key protocol. The protocol is first examined with respect to a single run and then more generally with respect to multiple concurrent runs.

Index Terms—Authentication, security protocols, formal methods, CSP, verification, Needham-Schroeder protocol.

1 Introduction

Authentication comes in a variety of forms, and authentication protocols are used in a number of ways depending on the precise security properties that they are believed to provide. International standard ISO 7498-2 [8] distinguishes two forms of authentication:

- *peer entity* authentication, which is intended to provide confidence in an entity's identity at the time of usage. This kind of authentication involves freshness, but does not provide any guarantees about any data transferred.
- *data origin* authentication, which provides corroboration of the source of a data unit.

Both of these forms of authentication are concerned with the identification of an entity, in one case to associate it with the current connection, and in the other case to associate it with a message. Entities are often identified with possession of a particular secret password or key by agents acting on their behalf, with protocols aiming to establish the fact of such possession. The international standard ISO/IEC 9798-1 states that "an entity to be authenticated proves its identity by showing its knowledge of a secret." Data origin authentication is commonly provided by encipherment or by digital signature, but the essential property is that the data could only have been generated by an entity in possession of particular secret information.

The question arises for any particular authentication protocol as to which kind of authentication the protocol was designed for, and which kinds it actually provides. A framework for expressing different flavors of authentication property would help to address this issue. An example of such a framework is the logic of authentication proposed in [2], which provides a language for expressing different security requirements. For example, the requirement that *A* should know she is communicating with *B* is easily distinguished from the requirement that *B* should know that *A* knows she is communicating with *B*.

Protocols are implemented in terms of messages, so the correctness of an authentication must consider the relationship between the messages of the protocol and the entities whose authentication is required. The aim of the Communicating Sequential Processes (CSP) approach presented in this paper is to describe precisely what a protocol is intended to achieve in terms of its messages, and to provide a framework for verifying claims about such properties. CSP provides a language for formal description of protocols, and a semantic theory for reasoning about their properties. 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 theoretical foundations of the CSP framework proposed by this author are described in [22]. 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 are 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 requires that the receipt of some message of the protocol guarantees that some earlier message must have been transmitted by the required agent. If *B* executes a protocol run ostensibly with *A*, then the final message *B* receives should provide some guarantees concerning *A*'s participation. We establish this by defining a suitable rank function on messages which shows that only messages above a particular rank can circulate in a restricted system in which *A* is blocked, and that this also blocks *B*'s protocol run—the rank of *B*'s final message is too low and the proof establishes that it cannot appear.

CSP is particularly suitable for describing protocols at a level close to the level we think of them. In other contexts the rank argument essentially amounts to an unreachability analysis or a proof that a particular word is not in a

- *S. Schneider is with the Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK. E-mail: s.schneider@dcs.rhbnc.ac.uk.*

Manuscript received 28 Feb. 1997; revised 18 Nov. 1997.
Recommended for acceptance by C. Landwehr.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 104040.

language. The strength of this approach is that there is a formal link between the rank arguments and a natural description of the protocol. Formalization 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 Formalization 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. Both peer entity authentication and data origin authentication properties can be specified within this approach.

One of the strengths of CSP is the ease with which specialized 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 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 specialized theory since it provides an appropriate level of abstraction for supporting the kind of reasoning we require.

The protocol to be analyzed in this paper is a slimmed down and amended version of the Needham-Schroeder public-key protocol [16], in which the public keys of A and B are already known to each other. The original form of this protocol has been the subject of an attack in [10], where Lowe suggests an amendment to the protocol intended to make it secure. The full version also involves communication between each party and a trusted server to obtain the public keys. The cut down version of the original protocol may be described as follows:

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

This protocol may be informally understood as follows: A invents a new message or *nonce* n_A, appends her identity A, encrypts the result with B's public key, and sends it to B. User B is able to decrypt the message and obtain the nonce n_A. He sends it back to A, together with his own newly invented nonce n_B, all encrypted with A's public key. A decrypts this response, and responds by sending B's nonce back. The intention is that at the end, A and B have each authenticated their identity to the other.

Lowe's attack describes a scenario in which user A's legitimate initiation of the protocol with I is used to initiate a protocol run with B where B acts as if the other partner is A:

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

Here $I(A)$ denotes agent I impersonating A, either generating a message that appears to B to have come from A, or intercepting a message addressed to A. The end result of this interchange of messages is that A believes she has established a session with I, and B believes he has established a session with A.

Lowe suggests an amendment to the protocol, requiring that the second message contains the responder's identity explicitly. The amended version, which will be used as the running example throughout this paper, is described as follows:

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

This amendment foils the attack above, but the question arises as to whether it is open to other attacks.

The structure of this paper is as follows. Section 2 introduces the CSP notation and theory which underpins this approach. Section 3 describes the way CSP is used to model and analyse security protocols and authentication properties, and presents the general approach to verification. Section 4 illustrates the approach through a verification of the amended Needham-Schroeder public-key protocol. Section 5 discusses how the approach might also be applied to confidentiality properties. Section 6 provides a general discussion and comparison with other approaches.

2 CSP Notation

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 behavior can be verified through use of the theory. Section 2 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], [21].

2.1 Events

Systems are modeled 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 messages $i.N.m = \{i.n.m \mid n \in N\}$. A channel c is said to be of type M if any event $c.m \in \Sigma$ has that $m \in M$.

2.2 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 $c!v \to P$ is able initially to perform only $c.v$, the output of v on channel c, after which it behaves as P. The input $c?x : T \to P(x)$ 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 $P \square Q$ (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 $\square_{i \in I} P_i$ 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 $P \mathbin{|[D]|} Q$ behaves as P and Q acting concurrently, with the requirement that they have to synchronize on any event in the synchronization 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 $P \mathbin{|||} Q$ which is equivalent to $P \mathbin{|[\{\}]|} Q$.

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

$$LIGHT \mathrel{\widehat{=}} on \to off \to LIGHT$$

defines a process *LIGHT* whose only possible behavior 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 $F_k(\underline{X})$ possibly involving any of those names appear on the right. For example, the set of definitions

$$COUNT_0 \mathrel{\widehat{=}} up \to COUNT_1$$

$$\begin{aligned} COUNT_{n+1} \mathrel{\widehat{=}} & (up \to COUNT_{n+2}) \\ & \square\ down \to COUNT_n \end{aligned}$$

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.

Process definitions may also contain conditions to separate different cases. The collection of *COUNT* definitions could be given as

$$\begin{aligned} & COUNT_m \mathrel{\widehat{=}} \\ & \quad up \to COUNT_{m+1} && \text{if } m = 0 \\ & \quad (up \to COUNT_{m+1}) \\ & \quad \square\ down \to COUNT_{m-1} && \text{otherwise} \end{aligned}$$

or could be given by $COUNT_m \mathrel{\widehat{=}} F_m(\underline{COUNT})$), where

$$\begin{aligned} & F_m(\underline{COUNT}) \mathrel{\widehat{=}} \\ & \quad up \to COUNT_{m+1} && \text{if } m = 0 \\ & \quad (up \to COUNT_{m+1}) \\ & \quad \square\ down \to COUNT_{m-1} && \text{otherwise} \end{aligned}$$

For readability, subscripted information may also appear bracketed on the same line as its corresponding process: for example, $COUNT_m$ may also be written as $COUNT(m)$.

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

2.3 Traces

The semantics of a process P is defined to be the set of finite sequences of events or *traces* ($traces(P)$) that it may possibly perform. This set will always be nonempty, and prefix closed: If tr is a possible trace of P, then so too is any prefix of tr. Examples of traces are $\langle\rangle$ (the empty trace, which is possible for any process) and $\langle on, off, on\rangle$ which is a possible trace of *LIGHT*. Concatenation of traces is described using the notation $tr_1 \frown tr_2$, which is the sequence of events listed in tr_1 followed by those in tr_2. Any trace is of the form $\langle\rangle$ or $\langle a\rangle \frown tr'$ where a is an event and tr' is the remainder of the trace.

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 the channels in C. These may be described inductively on sequences, and the last by a set comprehension:

$$\langle\rangle \upharpoonright D = \langle\rangle$$

$$(\langle d\rangle \frown tr) \upharpoonright D = \begin{cases} \langle d\rangle \frown (tr \upharpoonright D) & \text{if } d \in D \\ (tr \upharpoonright D) & \text{otherwise} \end{cases}$$

$$\langle\rangle \downarrow C = \langle\rangle$$

$$(\langle d\rangle \frown tr) \downarrow C = \begin{cases} \langle m\rangle \frown (tr \downarrow C) & \text{if } \exists c \in C \bullet d = c.m \\ (tr \downarrow C) & \text{otherwise} \end{cases}$$

$$tr \Downarrow C = \{m \mid (tr \downarrow C) \upharpoonright m \neq \langle\rangle\}$$

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

2.4 Analyzing Processes

Specifications are given as predicates on traces, and a process P satisfies a specification S (with tr as free variable) if all of its traces satisfy S:

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

Specifications are written with tr as the free trace variable.

For example,

$$LIGHT \textbf{ sat } \sigma(tr) \subseteq \{on, off\}$$

2.5 Proof Rules

The traces model for CSP is associated with a proof system for describing specifications on processes in terms of specifications on their components. There is a proof rule for each CSP operator, whose semantics in each case follows from the trace semantics. For example, the rule for the prefix operator is

$$\frac{P \textbf{ sat } S}{\begin{aligned} a \to P \textbf{ sat } (& tr = \langle\rangle \lor \\ & tr = \langle a\rangle \frown tr' \land S[tr'/tr]) \end{aligned}}$$

The predicate $S[tr'/tr]$ is S with all free occurrences of tr substituted by tr'.

Particular application domains are often concerned only with particular forms of specification. It is often possible to give more specialized proof rules for such specifications. This amounts to developing a specialized theory and proof system for the application area. The benefits are that the full generality of the proof rules are not required, and that there will often be lemmas and theorems, built on top of the traces model, which allow higher-level reasoning. This is the approach taken in this paper, where a key property **maintains** ρ used for verification is defined in terms of traces, and various proof rules are provided for deducing when particular process descriptions meet this property. We will see the specialized rules in Figs. 3, 4, and 5.

2.6 Process Equivalences

The traces model for CSP supports a number of algebraic equivalences on processes, whose soundness follows from the trace semantics. These are often useful in manipulating process descriptions into a form which is easier to reason about. There are many laws expressing useful identities. For the purposes of this paper, we will be interested in the effect of restricting particular events of a parallel combination $P \,|[R]|\, Stop$. This process restricts all of P's occurrences of events from R, so it has precisely those traces of P that do not contain any event from R. The equations are given in Fig. 1.

Rule `restrict.1`
If $\sigma(P) \cap R = \varnothing$ then $P \,|[\,R\,]|\, Stop = P$

Rule `restrict.2`

$$\begin{aligned}(P \,|||\, Q) \,|[\,R\,]|\, Stop = \\ (P \,|[\,R\,]|\, Stop) \,|||\, (Q \,|[\,R\,]|\, Stop)\end{aligned}$$

Rule `restrict.3`

$$\begin{aligned}(c?x : T \to P(x)) \,|[\,R\,]|\, Stop = \\ c?x : U \to (P(x) \,|[\,R\,]|\, Stop)\end{aligned}$$

where $U = T \setminus \{t \mid c.t \in R\}$.

Rule `restrict.4`

$$\begin{aligned}&(c!v \to P) \,|[\,R\,]|\, Stop = \\ &\quad c!v \to (P \,|[\,R\,]|\, Stop) \quad \text{if } c.v \notin R \\ &\quad Stop \quad\quad\quad\quad\quad\quad\quad\; \text{if } c.v \in R\end{aligned}$$

Fig. 1. Equations for restricted parallel combinations.

Rule `restrict.1` states that restricting a process on a set of events R that it cannot perform has no effect. Rule `restrict.2` states that restricting a process on a set of events distributes over interleaving.

Rules `restrict.3` and `restrict.4` are concerned with the effect of a restriction on inputs and outputs.

These equations are used throughout the paper whenever a process of the form $USER_A|[R]|\ Stop$ is expanded. They will not be referred to explicitly when used, in order to avoid cluttering proofs.

3 The General CSP Model

3.1 The CSP Network Description

The approach taken is to provide a CSP description of a generalization 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, readdress, 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 [22] 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 the approach to verification taken in 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 (finite) 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 $\sigma(USER_i) \subseteq trans.i \cup rec.i$, where *trans.i* is shorthand for *trans.i.USER.MESSAGE* and *trans.j* is shorthand for *trans.j.USER.MESSAGE*.

The resulting network is then described as follows:

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

This network is pictured in Fig. 2.

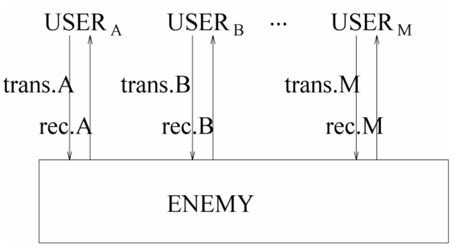


Fig. 2. CSP model of the network.

In the analysis of a protocol we might consider A and B as the two parties involved in the protocol, and the processes $USER_A$ 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 genuine user.

The channels *trans* and *rec* are of type *USER.USER. MESSAGE*. An event *trans.i.j.m* models an occurrence of node i sending a message m with destination j, and an event *rec.j.i.m* models an occurrence of node j receiving a message m with source i. Thus, i is the source, j the destination, and m the message. The message space *MESSAGE* will generally be defined by a context-free grammar, as in Section 3.1.3.

Refusals have been abstracted away, in the sense that input can never be refused. This amounts to making the assumption that the enemy can deduce nothing 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. (If this is later felt to be unrealistic, the definition can be altered accordingly, so that messages may not be input after the number of messages in the network reaches some threshold.) Attacks on protocols (apart from denial of service attacks) tend to exploit unexpected interactions between messages, and the traces model is adequate for capturing these.

3.1.1 Description of ENEMY

The capabilities of the enemy can be captured within a description that controls all *trans.i* and *rec.i* channels. Dolev and Yao first formalized this kind of attacker in [5]. It is able to accept any message output by any user, and it can also provide to any user any input that it is able to generate. In particular, this description gives the enemy the capability of redirecting, spoofing, replaying and blocking messages, as well as delivering them normally.

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

The argument S is the set of messages in the possession of the enemy. It is augmented every time some input occurs, and the messages m that may be output to agents are those that can be generated from S. This is written $S \vdash m$, where $\vdash$ will be defined in Section 3.1.4. In the output branch of the choice, an arbitrary user i is chosen, and a message is output to him along his *rec.i* channel.

The set of messages initially in the possession of the enemy will be given as a set *INIT*, and so $ENEMY = ENEMY$ $(INIT)$.

The enemy also subsumes the actions of other subverted or dishonest users by including their initial knowledge (such as their private keys) in the set *INIT*.

3.1.2 Protocol Participants

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

The agents A and B that are the participants in the protocol are modeled as $USER_A$ and $USER_B$, consisting of CSP implementations of the two halves of the protocol. More generally, if there are other participants (such as trusted third parties) then their activity will also be described as CSP processes. Obviously their description will depend entirely on the protocol being modeled.

The agents A and B which implement a run of the amended Needham-Schroeder protocol may be modeled in CSP as the processes $USER_A$ and $USER_B$ below. For the purposes of illustrating the approach, we will first simplify the analysis by considering only the case where A is the initiator and B is the receiver. This model does not allow for attacks where both parties act as initiators, or as receivers. This restriction will be relaxed in Section 4.2, where we consider the fully general case of each participant taking either role, and potentially engaging in multiple concurrent runs of the protocol.

$$\begin{aligned} USER_A = \;& \Box_{i \in USER} \\ & trans.\,A\,!\,i\,!\,p_i(n_A.\,A) \rightarrow \\ & \quad rec.\,A.\,i\,?\,p_A(n_A.\,x.\,i) \rightarrow \\ & \quad\quad trans.\,A\,!\,i\,!\,p_i(x) \rightarrow Stop \end{aligned} \tag{2}$$

$$\begin{aligned} USER_B = \;& rec.\,B\,?\,j\,?\,p_B(y.\,j) \rightarrow \\ & \quad trans.\,B\,!\,j\,!\,p_j(y.\,n_B.\,B) \rightarrow \\ & \quad\quad rec.\,B.\,j.\,p_B(n_B) \rightarrow Stop \end{aligned} \tag{3}$$

3.1.3 Message Space

The message space we use for analysis of this protocol can be considered as the language generated by the following context-free grammar:

$$\begin{aligned} RAW &::= USER \mid TEXT \\ &\quad\; \mid NONCE \mid KEY \\ MESSAGE &::= RAW \mid KEY(MESSAGE) \\ &\quad\; \mid MESSAGE.MESSAGE \end{aligned}$$

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 will adopt the following notational convention: $u \in USER$, $r \in RAW$, $t \in TEXT$, $n_i \in NONCE$, $p_i \in PUBLIC$, $s_i \in SECRET$, and $m \in MESSAGE$.

3.1.4 The 'Generates' Relation $\vdash$

We have rules which encapsulate the properties of public key cryptography, which describe the way messages may be generated from existing ones. These rules define 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 $S \vdash m$
$M1$ If $S \vdash m$ and $S \vdash k$ then $S \vdash k(m)$
$M2$ $S \vdash m_1$ and $S \vdash m_2$ if and only if $S \vdash m_1.m_2$

We are now 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 3.1.

$$ENEMY \quad \mathbf{sat} \quad (INIT \cup (tr \Downarrow trans)) \vdash tr \Downarrow rec$$

PROOF. We can prove easily by a mutual recursion induction that

$$ENEMY(S)\ \mathbf{sat}\ S \cup (tr \Downarrow trans) \vdash tr \Downarrow rec$$

The result follows from the fact that $ENEMY \mathrel{\widehat{=}} ENEMY(INIT)$. □

3.1.5 Equations

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

$$E1 \qquad p_A\, s_A(m)) = s_A(p_A(m)) = m$$

There would also be properties such as associativity of catenation:

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

Equations could also capture possible properties of encryption mechanisms. For example, commutativity of encryption is sometimes a necessary property (such as that required for Diffie-Hellman key exchange).

$$E3 \qquad k_1(k_2(m)) = k_2(k_1(m))$$

We will assume for the rest of this paper that ($E1$) and ($E2$) are the only equations on the message space unless explicitly stated otherwise. We will write $m_1 =_E m_2$ if m_1 and m_2 are equivalent under $E1$ and $E2$.

We will also assume and that $A1$—$A3$ and $M1$—$M2$ define the relation $\vdash$.

3.1.6 Modeling Issues

There are a number of issues concerning the way the protocol has been modeled. 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 accept 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 permissible input for the code fragment $rec.A.i?p_A(n_A.x) \rightarrow \ldots$ described in line 2 of $USER_A$ 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.

In line 2 the pattern matching amounts to any message of the form $p_A(n_A.x.i)$ being accepted for any nonce x, provided i matches the i chosen as the destination of the first message.

Freshness of nonces n_A and n_B is captured in the fact that neither of them appears in $INIT$, and in the fact that the definition of the generates relation $\vdash$ does not have any clause pertaining to generation of nonces. In other words, an agent (or the enemy) can generate a nonce only if it is already in his possession. The generation of a fresh nonce is *modeled* by making it known initially only to a single agent—only that agent is able to produce it. Of course, this is for analysis purposes only; in practice, nonces will be produced by mechanisms such as random number generation.

The apparent source j of B's first message (line 3) must match the source given in the message itself, since that information is the only information B has concerning the originator of the protocol.

3.2 Authentication

A message-oriented approach to authentication is discussed in [22]. Authentication will be captured in terms of events (generally transmission and receipt of particular messages) whose occurrence guarantees the prior occurrence of other messages. An authentication protocol generally achieves its aims by allowing a participant to infer from receipt of a message that the other party must have previously been involved in the protocol run. The key property is concerned with precedence between events: any element of a set of events T must have been preceded by occurrence of some element from the set of events R. This is captured as a specification as follows:

DEFINITION 3.2.

$$R \ \mathtt{precedes}\ T \ \widehat{=}\ tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle$$

A number of aspects of authentication can be captured in this style: Data origin authentication allows B to confirm that the response to his nonce challenge really was generated by A, who knows she is talking to B. This is captured by the specification '$trans.A.B.p_B(n_B)$ $\mathtt{precedes}$ $rec.B.A.p_B(n_B)$'. Receipt by B of the message $p_B(n_B)$ guarantees that A transmitted it earlier to B. This property will be the one verified in Section 4 of this paper.

This property may also be considered as describing peer entity authentication, provided freshness could also be established, since it would provide a guarantee that at the time of the protocol run it was indeed A communicating with B. Freshness could be expressed by the requirement that $trans.A.B.p_B(n_B)$ must occur after the beginning of the protocol run, and hence that it is preceded by $trans.A.B.p_B(n_A.A)$. This latter property is easily verified by an examination of $USER_A$.

A weaker version of peer entity authentication might have B simply requiring confirmation of the identity of A. This might be captured by the specification '$\{trans.A.j.pj(n_B) \mid j \in USER\}$ $\mathtt{precedes}$ $rec.B.A.p_B(n_B)$'. On receipt of the response to the nonce challenge, B can confirm that A did indeed respond to the nonce challenge, but possibly to a different agent. Again, freshness should also be established.

Data origin authentication for B's message is captured by the specification

$$\forall n : NONCE \bullet trans.B.A.p_A(n_A.n.B) \quad \mathtt{precedes}\ rec.A.B.p_A(n_A.n.B)$$

On receipt of any response to her nonce challenge, A confirms that the response was generated by B and B knows that he is talking to A.

One benefit of this approach is that it supports the expression of different authentication requirements rather than attempting to provide a definitive version of "authentication."

The following lemma is an immediate consequence of the definition. This is useful for relating the rank function approach to the authentication properties.

LEMMA 3.3.

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

The CSP trace semantics also support a number of specialized proof rules for establishing when particular processes provide specific `precedes` properties. Those used in this paper are given in Figs. 3 and 4.

The soundness of the rules follows from the trace semantics of the operators and the formal definition of R `precedes` T. We may give informal justification of their soundness by considering that occurrence of an event from T is intended to provide evidence that some event from R previously occurred. Hence a process *fails* to satisfy R `precedes` T only when it is possible for some event from T to occur before any event from R.

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

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

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

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

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

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

Finally, the rule `auth.recursion` for mutually recursive processes states that if the property R `precedes` T is preserved by recursive calls then it holds for the recursively defined process. This rule is a special case of the general rule for recursion; for further details, see the discussion in [3].

3.3 General Proof Strategy

3.3.1 Rank Functions

In order to prove that a network meets a particular authentication property expressed as R preceding T, it will be necessary to establish that no event in T can be generated anywhere in the system if all occurrences of events in R are blocked. A rank function $\rho : MESSAGES \to \mathbb{Z}$ on messages will be used to establish this. The intention is that the value associated with any message that can circulate in the network should be strictly positive; and that the values associated with messages appearing as parts of events in T should be 0 or less.

If $\rho : MESSAGE \to \mathbb{Z}$ is a function defined on messages, then we lift it to sets of messages S and sequences of messages *seq* as follows:

Rule `auto.stop`

$$\frac{}{Stop \textbf{ sat } R \texttt{ precedes } T}$$

Rule `auth.prefix.1`

$$\frac{}{a \to P \textbf{ sat } R \texttt{ precedes } T} \quad [\, a \in R \,]$$

Rule `auth.prefix.2`

$$\frac{P \textbf{ sat } R \texttt{ precedes } T}{a \to P \textbf{ sat } R \texttt{ precedes } T} \quad [\, a \notin T \,]$$

Rule `auth.choice`

$$\frac{\forall j \bullet V_j \textbf{ sat } R \texttt{ precedes } T}{\Box_j \; V_j \textbf{ sat } R \texttt{ precedes } T}$$

Fig. 3. Proof rules for authentication: prefix and choice.

Rule `auth.parallel`

$$\frac{P \textbf{ sat } R \texttt{ precedes } T}{P \,|[\, A \,]|\, Q \textbf{ sat } R \texttt{ precedes } T} \quad [\, T \subseteq A \,]$$

Rule `auth.interleaves`

$$\frac{\begin{array}{l} P \textbf{ sat } R \texttt{ precedes } T \\ Q \textbf{ sat } R \texttt{ precedes } T \end{array}}{P \,|||\, Q \textbf{ sat } R \texttt{ precedes } T}$$

Rule `auth.recursion`
If $\forall\, l \bullet N_l \mathrel{\widehat=} F_l(\underline{N})$ then

$$\frac{\begin{array}{l} \forall\, \underline{X} \bullet \\ (\forall\, l \bullet X_l \textbf{ sat } R \texttt{ precedes } T) \Rightarrow \\ (\forall\, l \bullet F_l(\underline{X}) \textbf{ sat } R \texttt{ precedes } T)) \end{array}}{\forall\, l \bullet N_l \textbf{ sat } R \texttt{ precedes } T}$$

Fig. 4. Proof rules for authentication: parallel and recursion.

$$\rho(S) = \min \{\rho(s) \mid s \in S\}$$
$$\rho(seq) = \min \{\rho(m) \mid c.m \ \mathbf{in} \ seq\}$$

The rank of an event *c.m* is the rank of the message *m*.

The definition of ρ must respect the equations on the space of messages in order to be well-defined. In the case of this paper this means that it should respect *E*1 and *E*2.

The property required by each component is that it cannot introduce any message whose rank is less than 1. In other words, if all it has ever received is messages of strictly positive rank, then so should it has ever transmitted. This property may be captured as a specification **maintains** ρ on traces:

DEFINITION 3.4.

$$\mathbf{maintains}\ \rho \widehat{=}$$
$$\rho(tr \Downarrow rec) > 0 \Rightarrow \rho(tr \upharpoonright trans) > 0$$

Observe that this is parameterized by the rank function ρ.

If $USER_i$ meets this specification then it cannot introduce a message of rank 0 or less into the system (along channel *trans.i*) if there was not one already present (which was received along channel *rec.i*).

The following theorem is at the heart of the proof strategy presented in this paper.

THEOREM 3.5. *If*

R1: $\forall m \in INIT \bullet \rho(m) \geq 1$

R2: $\forall S, m \bullet$
$((\forall s \in S \bullet \rho(s) \geq 1) \wedge S \vdash m \Rightarrow \rho(m) \geq 1)$

R3: $\forall t \in T \bullet \rho(t) \leq 0$

R4: $(USER_i \mid[R]\mid Stop$ **sat maintans** $\rho)$ for each user *i*

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

The proof of this theorem appears in Appendix A.

The requirements of the theorem combine to establish that only messages of strictly positive rank can circulate in $NET \mid[R]$ *Stop*, and hence that nothing in *T* can be generated if all events in *R* are blocked. This means that any occurrence of an event in *T* indicates that an event in *R* must have occurred previously.

The theorem describes a number of key properties required of the rank function. Properties *R*1 and *R*2 together yield the result that the enemy cannot introduce messages which fail to meet preserve strictly positive rank: if the enemy begins with such messages, and only ever sees messages of strictly positive rank, then only such messages can be generated. Property *R*4 provides the same guarantee for all of the users. Property *R*3 is used to deduce that no event from *T* can, therefore, be generated.

Observe that the construction of a suitable rank function is dependent on *R* and *T* as well as the descriptions of the users. In other words, it is dependent both on the protocol and on the particular property to be established.

3.3.2 Proof Obligations

Theorem 3.5 relies on the assumption that

$$\forall i \bullet \sigma(USER_i) \cap (trans \cup rec) \subseteq trans.i \cup rec.i$$

i.e., that no user except $USER_i$ has any interaction with *ENEMY* on the channels *trans.i* or *rec.i*. This assumption is built into the way we are modeling the network.

As discussed earlier, authentication properties expressed in CSP are in the form '*R* precedes *T*,' meaning that if some event from *T* occurs then some event from *R* must previously have occurred. Generally *T* will be a set of possible inputs which might be received at a particular node, providing evidence of the occurrence of one of a set of outputs *R* at a different node.

The proof strategy we adopt is to show that if all occurrences of events *R* are prevented in *NET*, then the events in *T* is not possible. In other words, we aim to show that the events in *T* cannot be generated by the resulting system description $NET \mid[R]\mid Stop$.

To apply Theorem 3.5 we must meet the list of requirements *R*1 to *R*4. Some of these correspond to assumptions which we must be confident can be made, and others are proof obligations.

Item *R*1 is an assumption on the enemy, and confidence in it will depend on the nature of ρ, and on the messages that have nonpositive rank.

Item *R*2 must be checked, and may be established by an induction over the definition of the generates relation $\vdash$. It therefore relies on the fact that the clauses defining $\vdash$ completely determine which messages can be generated from already known ones. Item *R*3 must be shown for the particular set *T*. Item *R*4 must be proven for the cases (*R*4*a*): $USER_A \mid[R]\mid Stop$ and (*R*4*b*): $USER_B \mid[R]\mid Stop$.

3.3.3 Specialized CSP Proof Rules

There are also a number of rules which can be given concerning the relationship between various CSP operators and the **maintains** ρ specification. These are given in Fig. 5. They are all sound with respect to the CSP traces model. Together with the equations of Fig. 1 they will be used in establishing *R*4*a* and *R*4*b* in various cases.

Informally, their soundness can be justified as follows. Rule **stop** is sound because *Stop* is unable to violate **maintains** ρ since to do so requires an output of a message of nonpositive rank, and *Stop* can perform no output. Rule **output** states that if the first output provided by a process has positive rank, then the process satisfies **maintains** provided that the behavior after this first output does not violate it.

Rule **input** is concerned with the behavior of a process subsequent to an input. The requirement to maintain positive rank is concerned that if messages coming in have positive rank, then the messages going out should also have positive rank. For a particular incoming message, there are therefore two cases to consider: if the input message *f*(*x*) has rank 0 or less, then the subsequent behavior is irrelevant since responsibility for maintaining positive rank is no longer required; and if the message *f x*) input has positive rank, then the subsequent process *P*(*j*, *x*) should maintain positive rank. Hence the rule states that the input process $rec.i?j?f(x) \rightarrow P(j, x)$ satisfies **maintains** ρ whenever *P*(*j*, *x*) does so after an input of positive rank. The form of the input *f* (*x*) describes the pattern matching implicit in the input process: *f* describes the input patterns allowed.

Rule `stop`

$$\frac{}{Stop\ \mathbf{sat}\ \mathtt{maintains}\ \rho}$$

Rule `output`

$$\frac{P\ \mathbf{sat}\ \mathtt{maintains}\ \rho}{trans.i.j.m \rightarrow P\ \mathbf{sat}\ \mathtt{maintains}\ \rho}\quad [\,\rho(m) \geqslant 1\,]$$

Rule `input`

$$\frac{\forall j, x \bullet (\ \rho(f(x)) \geqslant 1 \Rightarrow (P(j,x)\ \mathbf{sat}\ \mathtt{maintains}\ \rho))}{rec.i?j?f(x) \rightarrow P(j,x)\ \mathbf{sat}\ \mathtt{maintains}\ \rho}$$

Rule `choice`

$$\frac{\forall j \bullet V_j\ \mathbf{sat}\ \mathtt{maintains}\ \rho}{\Box_j\ V_j\ \mathbf{sat}\ \mathtt{maintains}\ \rho}$$

Fig. 5. Proof rules for *maintains* ρ.

Finally, rule `choice` states that if each branch of a choice maintains positive rank, then so does the entire choice.

4 Analysis of the Amended Needham-Schroeder Protocol

The protocol to be analyzed is as described in Section 1:

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

The property to be proved as an illustration of the approach is that receipt by B from A of the message $p_B(n_B)$ guarantees that A previously sent that message to B. This is expressed as the CSP specification

$$trans.A.B.p_B(n_B)\ \mathtt{precedes}\ rec.B.A.p_B(n_B)$$

This may be considered as an entity authentication property, establishing that B can be sure he is in communication with A. It can also be considered as a data origin authentication property, establishing for B that the source of the message $p_B(n_B)$ must be A.

4.1 Proof

In order to establish that a run of this protocol has $trans.A.B.p_B(n_B)$ guaranteed by $rec.B.A.p_B(n_B)$ we use the CSP description

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

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

The property of user B authenticating user A sending $p_B(n_B)$ requires that if B engages in the protocol as if A is the initiator, then A is indeed the initiator.

Hence, we are interested only in those executions of B which begin with some message of the form *rec.B.A.m*, where the source of message m appears to be A. The description required for the analysis is the description $USER_B$ under the restriction that B takes the first message to have A as its source.

The rank function described in Fig. 6 is suitable for an application of Theorem 3.5: it meets properties $R1$–$R4$. In particular

R4a $USER_A\ |[trans.A.B.p_B(n_B)]|\ Stop$ **sat** `maintains` ρ

R4b Since $trans.A.B.p_B(n_B) \notin \sigma(USER_B)$, it follows that $USER_B\ |[trans.A.B.p_B(n_B)]|\ Stop = USER_B$, so we have only to prove

$User_B$ sat `maintains` ρ

Each of the properties must be examined in turn.

$$\begin{aligned} \rho_0(u) &= 1 \\ \rho_0(t) &= 1 \\ \rho_0(n) &= \begin{cases} 0 & \text{if } n = n_B \\ 1 & \text{otherwise} \end{cases} \\ \rho_0(p_i) &= 1 \\ \rho_0(s_i) &= \begin{cases} 0 & \text{if } i = A \text{ or } i = B \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

$$\begin{aligned} \rho(r) &= \rho_0(r) \\ \rho(p_j(m)) &= \begin{cases} \rho(m) + 1 & \text{if } j = A \text{ and} \\ & \quad \exists\, n : NONCE. \\ & \quad m =_E n.n_B.B \\ \rho(m) & \text{otherwise} \end{cases} \\ \rho(s_j(m)) &= \begin{cases} \rho(m) - 1 & \text{if } j = A \text{ and} \\ & \quad \exists\, n : NONCE. \\ & \quad m =_E p_A(ni.n_B.B) \\ \rho(m) & \text{otherwise} \end{cases} \\ \rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\} \end{aligned}$$

Fig. 6. Rank function for verification of Lowe's amended protocol.

Well-definedness of ρ. Since ρ is defined on the structure of the messages, it is necessary to check that any equations on the message space hold: any two forms of the same message must have the same rank.

It is immediate from the definition of ρ that equations $E1$ and $E2$ both hold:

$$\rho(p_i(s_i(m))) = \rho(s_i(p_i(m))) = \rho(m)$$

and

$$\rho(m_1.(m_2.m_3)) = \rho((m_1.m_2).m_3)$$

R1. We assume that $\forall m \in INIT \bullet \rho(m) \geq 1$. This is a reasonable assumption, amounting to the claim that the enemy does not initially have access to s_A, s_B, or n_B, or messages constructed from these.

R2. We obtain the following result from a consideration of each clause in turn of the definition of $\vdash$: A1–A3, M1–M2. The result follows from the fact that $\vdash$ is defined to be the smallest relation closed under all of the clauses.

$$(\forall\, s \in S \bullet \rho(s) \geq 1) \wedge S \vdash m \Rightarrow \rho(m) \geq 1$$

R3. Observe that $\rho(p_B(n_B)) = 0$.

R4a (*USER*$_A$). We want to establish that

$$USER_A \,|[\, trans.A.B.p_B(n_B) \,]|\, Stop \quad \textbf{sat}\ \texttt{maintains}\ \rho$$

For a given $x \in NONCE$, either $x \neq n_B$ or $x = n_B$.

If $x \neq n_B$ it follows that $\rho(x) \geq 1$ and so $\rho(p_i(x)) \geq 1$ (for any i), and hence that

$$trans.A!i!p_i(x)\ Stop \rightarrow \ \textbf{sat}\ \texttt{maintains}\ \ \rho \tag{4}$$

For the second case it is immediate that *Stop* **sat** `maintains` ρ.

Hence, we have that

$$\begin{pmatrix} trans.A!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_B \\ Stop & \text{if } x = n_B \end{pmatrix} \quad \textbf{sat}\ \texttt{maintains}\ \rho$$

This provides the antecedent for Rule `input` to yield

$$\begin{aligned} & rec.A.i?p_A(n_A.x.i) \rightarrow \\ & \begin{pmatrix} trans.A!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_B \\ Stop & \text{if } x = n_B \end{pmatrix} \\ & \qquad \textbf{sat}\ \texttt{maintains}\ \rho \end{aligned}$$

And so from Rule `output` (since $\rho(p_i(n_A.A)) \geq 1$) we have for any i that

$$\begin{aligned} & trans.A!i!p_i(n_A.A) \rightarrow \\ & \quad rec.A.i?p_A(n_A.x.i) \rightarrow \\ & \quad \begin{pmatrix} trans.A!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_B \\ Stop & \text{if } x = n_B \end{pmatrix} \\ & \qquad \textbf{sat}\ \texttt{maintains}\ \rho \end{aligned} \tag{5}$$

Finally we obtain from Rule `choice` that

$$\begin{aligned} & \square_{i \in USER} \\ & \quad trans.A!i!p_i(n_A.A) \rightarrow \\ & \quad rec.A.i?p_A(n_A.x.i) \rightarrow \\ & \quad \begin{pmatrix} trans.A!i!p_i(x) \rightarrow Stop & \text{if } x \neq n_B \\ Stop & \text{if } x = n_B \end{pmatrix} \\ & \qquad \textbf{sat}\ \texttt{maintains}\ \rho \end{aligned}$$

Since this process is equivalent to $USER_A \,|[\, trans.A.B.p_B(n_B) \,]|\, Stop$ by the rules of Fig. 1, the proof is complete.

R4b (*USER*$_B$). Rule `stop` and Rule `input` (with trivial antecedent) yield that

$$rec.B.A.p_B(n_B) \rightarrow Stop \quad \textbf{sat}\ \ \texttt{maintains}\ \rho \tag{6}$$

Now for any nonce y $\rho(p_B(y.A)) \geq 1$ implies that $\rho(y) \geq 1$ which in turn implies that $\rho(p_A(y.n_B.B)) \geq 1$, we obtain from Rule `output` that

$$\begin{aligned} & \rho(p_B(y.A)) \geq 1 \Rightarrow \\ & \quad trans.B!A!p_A(y.n_B.B) \rightarrow \\ & \quad rec.B.A.p_B(n_B) \rightarrow Stop \\ & \qquad \textbf{sat}\ \texttt{maintains}\ \rho \end{aligned} \tag{7}$$

This provides the antecedent for Rule `input` to yield

$$\begin{aligned} & rec.B.A?p_B(y.A) \rightarrow \\ & \quad trans.B!A!p_A(y.n_B.B) \rightarrow \\ & \quad rec.B.A.p_B(n_B) \rightarrow Stop \\ & \qquad \textbf{sat}\ \texttt{maintains}\ \rho \end{aligned}$$

or in other words *USER*$_B$ **sat** `maintains` ρ, as required.

This completes the proof that the occurrence of $rec.B.A.p_B(n_B)$ guarantees the prior occurrence of $trans.A.B.p_B(n_B)$: B's receipt of n_B encrypted under his own public key provides evidence that A transmitted the message $trans.A.B.p_B(n_B)$.

The properties of ρ that were used in the proof, as side-conditions to the proof rules, were

1) $\rho(p_A(n_A.x.i)) \geq 1 \wedge x \neq n_B \wedge x \in NONCE \Rightarrow \rho(p_i(x)) \geq 1$ for any i (used to obtain line 4);
2) $\rho(p_i(n_A.A)) \geq 1$ for all i (used to obtain line 5);
3) $\rho(y) \geq 1 \wedge y \in NONCE \Rightarrow \rho(p_A(y.n_B.B)) \geq 1$ (used to obtain line 7);
4) $\forall\, m \in INIT \bullet \rho(m) \geq 1$ (required for $R1$);
5) $\rho(p_B(n_B)) = 0$ (required for $R3$)

Properties (1) and (3) use the fact that x and y are assumed to be nonces (rather than arbitrary messages), as discussed below.

Observe that the proof works even if $\rho(s_B) = 1$. This means that A can be authenticated to B even if B's own secret key is compromised. This is reasonable, since the fact that s_B is known only to B is really required by A rather than by B.

4.1.1 An Aside on Typing

The proof above uses the fact that messages are *typed*: the pattern matching built into the analysis of $USER_A \,|[\, trans.A.B.p_B(n_B) \,]|\, Stop$ assumes that all inputs that $USER_A$ will accept on *rec.A* will be of the form $i.p_A(NONCE.NONCE.i)$. In other words, the description of $USER_A$ assumes that the message $i.p_A(n_A.x.i)$ will be rejected unless x is a nonce.

In fact, the minimal assumption required is simply that user A is able to recognize a message of the form $p_A(n_A.x.i)$ for arbitrary messages x, and to extract the message x from this. The approach is still applicable, but at the cost of added complexity. The CSP descriptions of the processes will be more general since they will permit more inputs, and so the rank function will have to be preserved under more general conditions. For example, an untyped $USER_B$

which can accept any message y (rather than any nonce y) does not preserve the rank function of Fig. 6 since it can accept $p_B(n_A.n_C.A)$ which has rank 1 and then transmit $p_A(n_A.n_C.n_B.B)$ which has rank 0.

In fact, the protocol is still secure, but the rank function required to establish this is more complicated. In fact, two rank functions are needed, to cover the two cases where A has chosen to send to B, and where A has chosen to send to a user other than B (as explained in Section 5). Their construction is left as an exercise for the interested reader.

A verification of the general untyped case will establish that there are no attacks which can exploit type confusion between messages. A verification restricted to typed messages will be more straightforward (and sometimes possible when the general case is not), but should be accompanied by the caveat that an implementation should be able to recognize the types of the messages it is dealing with.

4.2 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 generalize 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 focused 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 other runs with any other parties, and to restrict their behavior 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 modeled 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 + 1th run can commence.

This requires a proof rule for recursive definitions, which is given in Fig. 7. Recursive definitions are of the form $N_l \mathrel{\widehat{=}} F_l(\underline{N})$, where each F_l is a function on CSP process expressions, containing instances of various N_l process names. The rule states that if it can be shown that each $F_l(\underline{X})$ satisfies *mrp* from the inductive assumption that all of the X_l satisfy `maintains` ρ, then we may conclude that each N_l in the recursive definition does in fact satisfy `maintains` ρ (The base case of the induction is that *Stop* satisfies `maintains` ρ, and this is already given by Rule `stop` of Fig. 5.).

The users are defined in terms of a mutual recursion. In this case we model each user as being prepared either to initiate or to respond to a fresh run of the protocol at any stage, independently of the number of runs it is already involved in at that time.

The protocol will be investigated as to whether an arbitrary run provides authentication—such a run will be the kth run for some k.

The individual process names N_l will be given as $USER_A(l)$ and $USER_B(l)$ for the two families of mutually recursive definitions defining users A and B, respectively.

Rule `interleaves`

$$\frac{\begin{array}{l} P_1 \ \mathbf{sat}\ \texttt{maintains}\ \rho \\ P_2 \ \mathbf{sat}\ \texttt{maintains}\ \rho \end{array}}{P_1 \mid\mid\mid P_2 \ \mathbf{sat}\ \texttt{maintains}\ \rho}$$

Rule `recursion`
If $\forall\, l \bullet N_l \mathrel{\widehat{=}} F_l(\underline{N})$ then

$$\frac{\begin{array}{l} \forall\, \underline{X} \bullet \\ ((\forall\, l \bullet X_l \ \mathbf{sat}\ \texttt{maintains}\ \rho) \\ \Rightarrow (\forall\, l \bullet F_l(\underline{X}) \ \mathbf{sat}\ \texttt{maintains}\ \rho)) \end{array}}{\forall\, l \bullet N_l \ \mathbf{sat}\ \texttt{maintains}\ \rho}$$

Fig. 7. Further proof rules for `maintains` ρ.

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

The description of $USER_B$ is entirely similar, with B replacing A throughout.

$$\begin{array}{l} User_B(l) = \\ \quad \Box_{i \in USER} \\ \qquad trans.B!i!p_i(n_B.B) \rightarrow \\ \qquad\quad \left(\begin{array}{l} rec.B.i\,?\,p_B(n_{B,l}.x.i) \rightarrow \\ \quad trans.B!i!p_i(x) \rightarrow Stop \\ \mid\mid\mid\ USER_B(l+1) \end{array}\right) \\ \quad \Box\ rec.B?j?p_B(y.j) \rightarrow \\ \qquad\quad \left(\begin{array}{l} trans.B!j\,?\,p_j(y.n_{B,l}.B) \rightarrow \\ \quad rec.B.j.p_B(n_{B,l}) \rightarrow Stop \\ \mid\mid\mid\ USER_B(l+1) \end{array}\right) \end{array}$$

We define $USER_A = USER_A(0)$ and $USER_B = USER_B(0)$.

As an example we will prove the equivalent of the first property: that $rec.B.A.p_B(n_{B,k})$ authenticates $trans.A.B.p_B(n_{B,k})$ for any given k. In other words, if B's kth run of the protocol is initiated by A and is between A and B, then B's receipt of the final message authenticates that A sent that message.

The description of $USER_B(k)$ will be restricted to reflect the fact that the analysis is with respect to this run.

$$\begin{array}{l} User_B(k) = \\ \quad rec.B.A?p_B(y.A) \rightarrow \\ \qquad \left(\begin{array}{l} trans.B!A!p_A(y.n_{B,l}.B) \rightarrow \\ \quad rec.B.A.p_B(n_{B,l}) \rightarrow Stop \\ \mid\mid\mid\ USER_B(k+1) \end{array}\right) \end{array}$$

The descriptions of the other $USER_B(l)$ and all the $USER_A(l)$ processes will remain unchanged.

An appropriate rank function is given in Fig. 8. It is very similar to the rank function in Fig. 6.

This rank function meets $R0$–$R3$. We have only to prove $R4$a for $USER_A$ and $R4$b for $USER_B$ to establish that $rec.B.A.p_B(n_B, k)$ authenticates $trans.A.B.p_B(n_B, k)$ for k.

R4a ($USER_A$). We have that

$$User_A(l) \,|[trans.A.B.p_B(n_{B,k})]|\, Stop =$$

$$\square_{i \in USER}\; trans.A!i!p_i(n_A.A) \rightarrow \left(\begin{array}{l} rec.A?j?p_A(n_{A,l}.x.i) \rightarrow \\ \quad\left\{\begin{array}{ll} Stop & \text{if } i = B \text{ and } x = n_{B,k} \\ trans.A!i!p_i(x) \rightarrow Stop & \text{otherwise} \end{array}\right. \\ ||| \\ \left(\begin{array}{l} USER_A(l+1) \\ |[trans.A.B.p_B(n_{B,k})]| \\ Stop \end{array}\right) \end{array}\right)$$

$$\square\; rec.A?j?p_A(y.j) \rightarrow \left(\begin{array}{l} trans.A?j?p_j(y.n_{A,l}.A) \rightarrow \\ \quad rec.A.j.p_A(n_{A,l}) \rightarrow Stop \;||| \\ \left(\begin{array}{l} USER_A(l+1) \\ |[trans.A.B.p_B(n_{B,k})]| \\ Stop \end{array}\right) \end{array}\right)$$

It is straightforward to prove that

$$rec.A?i?p_A(n_{A,l}.x.i) \rightarrow \left(\begin{array}{ll} Stop & \text{if } i = B \text{ and } x = n_{B,k} \\ trans.A!i!p_i(x) \rightarrow Stop & \text{otherwise} \end{array}\right)$$
$$\textbf{sat}\ \texttt{maintains}\ \rho$$

since $\rho(p_A(n_{A,l}.x.i)) \geq 1 \Rightarrow$

$$((i = B \wedge x = n_{B,k}) \vee \rho(p_i(x)) \geq 1).$$

Under the inductive assumption that $USER_A(l + 1)$ $|[trans.A.B.p_B(n_B, k)]|$ $Stop$ **sat** `maintains` ρ, Rule `interleaves` yields that

$$rec.A?j?p_A(n_{A,l}.x.i) \rightarrow \left(\begin{array}{ll} Stop & \text{if } i = B \text{ and } x = n_{B,k} \\ trans.A!i!p_i(x) & \\ \rightarrow Stop & \text{otherwise} \end{array}\right)$$
$$|||$$
$$USER_A(l+1)\,|[trans.A.B.p_B(n_{B,k})]|\,Stop$$
$$\textbf{sat}\ \texttt{maintains}\ \rho$$

$$\begin{array}{rcl} \rho_0(u) & = & 1 \\ \rho_0(t) & = & 1 \\ \rho_0(n) & = & \left\{\begin{array}{ll} 0 & \text{if } n = n_{B,k} \\ 1 & \text{otherwise} \end{array}\right. \\ \rho_0(p_i) & = & 1 \\ \rho_0(s_i) & = & \left\{\begin{array}{ll} 0 & \text{if } i = A \text{ or } i = B \\ 1 & \text{otherwise} \end{array}\right. \\ \\ \rho(r) & = & \rho_0(r) \\ \rho(p_j(m)) & = & \left\{\begin{array}{ll} 1 & \text{if } j = A \text{ and} \\ & \quad m \in NONCE.n_{B,k}.B \\ \rho(m) & \text{otherwise} \end{array}\right. \\ \rho(s_j(m)) & = & \left\{\begin{array}{ll} 0 & \text{if } j = A \text{ and} \\ & \quad m \in p_A(NONCE.n_{B,k}.B) \\ \rho(m) & \text{otherwise} \end{array}\right. \\ \rho(m_1.m_2) & = & \min\{\rho(m_1), \rho(m_2)\} \end{array}$$

Fig. 8. Rank function for verification of the *k*th run.

Rules `choice` and `output` together with the fact that $\rho(p_i(n_{A,l}.A)) \geq 1$ for any i, yield that the first branch of the choice satisfies `maintains` ρ.

Similar reasoning yields that the second branch of the choice also satisfies `maintains` ρ, and so we deduce that the process body of the equation for $USER_A(l)$ $|[trans.A.B.p_B(n_{B,k})]|$ $Stop$ satisfies `maintains` ρ. Hence, we conclude that

$$USER_A\;|[trans.A.B.p_B(n_{B,k})]|\,Stop$$
$$\textbf{sat}\ \texttt{maintains}\ \rho$$

as required.

R4b ($USER_B$). Since $trans.A.B.p_B(n_B, k)] \notin \sigma(USER_B)$ for any l, we need only to prove that $USER_B$ **sat** `maintains` ρ, which will be achieved by establishing it for each l.

We begin with the inductive hypothesis that $USER_B(l)$ **sat** `maintains` ρ for each l.

We must consider the case where $l = k$ separately from the case where $l \neq k$.

Case $l = k$:

$$USER_B(l) =$$
$$rec.B.A?p_B(y.A) \rightarrow \left(\begin{array}{l} trans.B!A!p_A(y.n_{B,k}.B) \rightarrow \\ \quad rec.B.A.p_B(n_{B,k}) \rightarrow Stop \\ |||\; USER_B(k+1) \end{array}\right)$$

An application of Rule `input` (with trivial antecedent) and an application of Rule `output` with the observation that $\rho(p_A(y.n_{B,k}.B)) \geq 1$ yields that

$$trans.B!A!p_A(y.n_{B,k}.B) \rightarrow$$
$$rec.B.A.p_B(n_{B,k}) \rightarrow Stop$$
$$\textbf{sat}\ \texttt{maintains}\ \rho$$

Rule `interleave` with the inductive hypothesis on $USER_B(k + 1)$ yields

$$\begin{pmatrix} trans.B!A!p_A(y.n_{B,k}.B) \rightarrow \\ \quad rec.B.A.p_B(n_{B,k}) \rightarrow Stop \\ ||| \; USER_B(k+1) \end{pmatrix}$$
$$\textbf{sat} \ \texttt{maintains} \ \rho$$

and so finally

$$rec.B.A?p_B(y.A)$$
$$\begin{pmatrix} trans.B!A!p_A(y.n_{B,k}.B) \rightarrow \\ \quad rec.B.A.p_B(n_{B,k}) \rightarrow Stop \\ ||| \; USER_B(k+1) \end{pmatrix}$$
$$\textbf{sat} \ \texttt{maintains} \ \rho$$

Hence, we conclude that

$$USER_B(l) \quad \textbf{sat} \quad \texttt{maintains} \ \rho \tag{8}$$

Case $l \neq k$: In this case the recursive equation is

$$\begin{aligned} & USER_B(l) = \\ & \square_{i \in USER} \\ & \quad trans.B!i!p_i(n_B.B) \rightarrow \\ & \qquad \begin{pmatrix} rec.B.i?p_B(n_{B,l}.x.i) \rightarrow \\ \quad trans.B!i!p_i(x) \rightarrow Stop \\ ||| \; USER_B(l+1) \end{pmatrix} \\ & \square rec.B?j?p_B(y.j) \rightarrow \\ & \qquad \begin{pmatrix} trans.B!j!p_j(y.n_{B,l}.B) \rightarrow \\ \quad rec.B.j.p_B(n_{B,l}) \rightarrow Stop \\ ||| \; USER_B(l+1) \end{pmatrix} \end{aligned}$$

Again the proof is almost identical to that provided for $USER_A(l) \,|[trans.A.B.p_B(n_{B,k})]|\, Stop$. The requirements on ρ required for Rules `input` and `output` to be applied are similar to those in Section 4.1:

1) $\rho(p_B(n_{B,l}.x.i)) \geq 1 \Rightarrow \rho(p_i(x)) \geq 1$;
2) $\rho(p_i(n_{B,l}.B)) \geq 1$;
3) $\rho(p_B(y.j)) \geq 1 \Rightarrow \rho(p_j(y.n_{B,l}.B)) \geq 1$

The first two arise from the first branch of the choice, and the third arises from the second branch. They are all easily checked, bearing in mind that $l \neq k$ means that $n_{B,l} \neq n_{B,k}$ and so $\rho(n_{B,l}.B) = 1$. The required result follows from an application of Rule `choice` and finally of Rule `recursion`.

Hence, we obtain for any $l \neq k$ that

$$USER_B(l) \quad \textbf{sat} \quad \texttt{maintains} \ \rho$$

Since $USER_B(k)$ **sat** `maintains` ρ has already been established, Rule `recursion` yields that

$$USER_B \quad \textbf{sat} \quad \texttt{maintains} \ \rho$$

as required.

Since the proof is valid for arbitrary k we deduce that on any protocol run receipt of the nonce challenge confirms that it was appropriately sent by the other party.

5 CONFIDENTIALITY

Authentication protocols are often concerned with key distribution, and so their authentication requirements may be bound up with other required properties of the protocol, such as the establishing of a good key, or the guarantee that at the end of the run the protocol's nonces are known only to the participants.

Authentication is distinct from confidentiality, and each might be provided by a protocol when the other is not. For example, an alternative way to provide authentication is to sign nonce challenges, but this approach does not also provide confidentiality. In general, the guarantee that each party is communicating with the other does not in itself ensure confidentiality, which must, therefore, be provided separately. However, in some situations, an attack on authentication may also provide an attack on confidentiality. For example, in the case of the protocol discussed in this paper, the nonces n_A and n_B might be expected to be secret to A and B, yet Lowe's attack allows the intruder to obtain them both because the absence of authentication allows a third party to be involved in the protocol run.

Confidentiality is also expressible in terms of CSP specifications, with only a slight adaptation to the model of the network. In order to describe cleanly in terms of messages whether or not the enemy can obtain a particular message, it is useful to add an additional channel *knows* to the enemy, which allows him to communicate anything he has deduced. The enemy's activity on this channel is similar to that on the *rec* channels. The definition of the enemy will be extended with the following branch of a choice:

$$\begin{aligned} & ENEMY(S) = \\ & \quad \ldots \\ & \quad \square \square_{S \vdash m} \ knows!m \rightarrow ENEMY(S) \end{aligned}$$

Confidentiality for a particular message m_0 will simply require that m_0 should never appear on *knows*, which is easily specified as

$$tr \restriction knows.m_0 = \langle\rangle$$

From the point of view of agent B, one of the confidentiality requirements is that his own nonce n_B remains secret. This is captured as the requirement that

$$NET \ \textbf{sat} \ tr \restriction knows.n_B = \langle\rangle$$

where $USER_A$ and $USER_B$ are as described in Section 4.1: $USER_A$ is honest but could undergo a protocol run with anyone; and $USER_B$ has a protocol run with A, since this property is concerned with confidentiality from B's point of view.

The approach to establishing this will again use rank functions, and apply a theorem similar to Theorem 3.5 adapted to address confidentiality. One simplification will be that the network NET does not need to be constrained on any events, so the unrestricted descriptions of $USER_A$ and $USER_B$ will be analyzed to ensure that they preserve the rank. Requirements $R1$ and $R2$ (with $R = \varnothing$) still need to be checked to ensure that the enemy cannot reduce the rank; finally, the equivalent of $R3$ requires the rank of the confidential message n_B itself to have rank 0. The construction of a rank function that meets these properties is left to the interested reader.

There is a second property which B requires of the protocol: that any nonce which is accepted as A's nonce challenge during the protocol run is indeed known only to A. Unlike n_B, this nonce is not known to B at the beginning of the run, and this makes the expression of the property more intricate.

This second property expressed in the form $tr \upharpoonright knows.n = \langle\rangle$ is too strong: any intruder could supply the first message $p_B(n.A)$ to B, and in this case n will not be secret. The property that is required is that if B completes a protocol run, apparently with A, then the nonce that B has accepted is known only to A and B. This is a kind of conditional confidentiality, requiring for any n that if $p_B(n.A)$ is the first message of the protocol, and the protocol runs to completion (as indicated by the occurrence of the final message $p_B(n_B)$), then $tr \upharpoonright knows.n = \langle\rangle$. This may be expressed as the following CSP specification:

$$\forall\, y{:}\, NONCE \bullet\ rec.B.A.p_B(y.A) \text{ in } tr \wedge p_B(n_B) \text{ in}$$
$$\Rightarrow tr \upharpoonright knows.y = \langle\rangle.$$

This description of the property is appropriate for the CSP description of a single run. If multiple runs are to be considered, then it would have to be more carefully crafted to ensure that the message n corresponds to the same run as the $n_{B,k}$. One way of achieving this would be to alter the CSP description by labeling each message in the run with the number of the run l (e.g., $trans.B.A.m$ will become $trans.B.A.l.m$); the label could not be tampered with by the enemy, it would be entirely under the control of the user and maintained purely to keep track of the runs for modelling purposes. The confidentiality property would then require that if y were the nonce received in the kth run, and that run were to terminate, then y could not be known to the enemy.

This property, and the previous one concerning $knows.n_B$, do not hold for the original Needham-Schroeder protocol.

There is no single rank function that can guarantee that NET meets this specification by assigning rank 0 to any nonce n which might appear on $knows$, for then n_A would have to have rank 0 (since the protocol can run to completion with this nonce). Yet A is able to send n_A to any agent, so $USER_A$ will not maintain positive ρ. However, a family of rank functions may still be used to address the different possibilities for execution: the various nonces that B can receive, and the various agents that A might choose to communicate with.

The specification above is equivalent to the following:

$$\forall\ n : NONCE \bullet rec.B.A.p_B(n.A) \text{ in tr} \Rightarrow$$
$$tr \upharpoonright p_B(n_B) = \langle\rangle$$
$$\vee\ tr \upharpoonright knows.n = \langle\rangle$$

Either the protocol does not complete, or the nonce accepted by B should be secret.

The choices for $USER_B$ can be expressed more explicitly as follows:

$$USER_B = \Box_{y \in NONCE}\ USER_B(y)$$
$$USER_B(y) = rec.B.A.p_B(y.A) \rightarrow$$
$$trans.B!A!p_A(y.n_B.B) \rightarrow$$
$$rec.B.A.p_B(n_B) \rightarrow Stop$$

Similarly, $USER_A$ can be expressed as a choice over all of the users with whom she might undertake a protocol run: $USER_A = \Box_{i \in USER}\ USER_A(i)$, where $USER_A(i)$ is given by

$$USER_A(i) = trans.A!i!p_i(n_A.A) \rightarrow$$
$$rec.A.i?p_A(n_A.x.i) \rightarrow$$
$$trans.A!i!p_i(x) \rightarrow Stop$$

In the traces model, all of the choices that are possible for the network components can be distributed over the parallel composition.

$$\begin{aligned} &USER_A \,|||\, USER_B \,||\, ENEMY(INIT) \\ &= \begin{pmatrix} \Box_{i \in USER}\ USER_A(i) \\ |||\ \Box_{y \in NONCE}\ USER_B(y) \end{pmatrix} \\ &\quad ||\,ENEMY(INIT) \\ &= \Box_{i \in USER,\, y \in NONCE} \\ &\quad \begin{pmatrix} USER_A(i) \,|||\, USER_B(y) \\ ||\, ENEMY(INIT) \end{pmatrix} \end{aligned}$$

This means that the various cases for $USER_A(i)$ and $USER_B(y)$ can be separately verified against the specification, and if all cases meet the specification, then the following general rule guarantees that the overall choice must also meet it:

$$\frac{\forall j \bullet V_j \textbf{ sat } S}{\Box_j\ V_j \textbf{ sat } S}$$

The cases to consider divide into $i = B$ and $i \neq B$, and $y = n_A$ and $y \neq n_A$. In each of these cases, a rank function can be provided which has either $\rho(p_B(n_B)) = 0$ or $\rho(y) = 0$, and which is preserved by both $USER_A(i)$ and $USER_B(y)$, guaranteeing either that the protocol cannot complete or that the nonce y accepted by B cannot appear on $knows$.

For example, when $i \neq B$, then an appropriate rank function is

$$\begin{aligned} \rho_0(u) &= 1 \\ \rho_0(t) &= 1 \\ \rho_0(n) &= \begin{cases} 0 & \text{if } n = n_B \\ 1 & \text{otherwise} \end{cases} \\ \rho_0(p_i) &= 1 \\ \rho_0(s_i) &= \begin{cases} 0 & \text{if } i = A \text{ or } i = B \\ 1 & \text{otherwise} \end{cases} \\ \rho(r) &= \rho_0(r) \\ \rho(p_j)(m)) &= \begin{cases} \rho(m) + 11 & \text{if } j = A \text{ and} \\ & m =_E n_A.n_B.B \\ \rho(m) & \text{otherwise} \end{cases} \\ \rho(s_j)(m)) &= \begin{cases} \rho(m) - 1 & \text{if } j = A \text{ and} \\ & m =_E p_A(n_A.n_B.B) \\ \rho(m) & \text{otherwise} \end{cases} \\ \rho(m_1.m_2) &= \min\{\rho(m_1), \rho(m_2)\} \end{aligned}$$

In this case $\rho(p_B(n_B)) = 0$: B's part of the protocol run will not terminate.

(Construction of the rank functions for the other cases is left as an exercise for the interested reader.) Different cases require different rank functions, but they all establish the same property for their particular case, so that property must hold for the complete description *NET*. All the rank functions allow nonces other than n_A and n_B to have rank 1 (so in the case of $y \neq n_B$ it is necessary to show that the protocol cannot complete, that is, that $\rho(p_B(n_B)) = 0$).

Hence, it appears that the rank function approach will extend to handle confidentiality properties, though the expression of such properties can be more delicate.

6 DISCUSSION

In this paper, we have shown how the theory of CSP might be specialized 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 a rank function ρ on messages, together with rules for deriving the required properties. We have also considered how the approach can be adapted to handle confidentiality properties.

We find that construction of the rank function 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 a rank function may also provide insight as to why a protocol does not provide authentication.

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 properties 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.

6.1 Comparison with Other Approaches

The CSP approach put forward by this author in [22] 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 channels *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:

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

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

Our top level requirement would be that

$NET \setminus (trans \cup rec)$ **sat**
$A.connect\_to.B$ `precedes` $B.authenticated.A$

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 of rank 0 or less 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 [19], [11] and elsewhere [24]. In the case of the analysis performed here, additional external events such as *A.nonce_challenge_OK.B* might be included to make finer distinctions between different flavors of authentication. This approach is simply a straightforward extension of the approach we have taken in this paper, introducing special events to mark particular points in the run of the protocol which we have been pinpointing directly.

Another issue of interest is that Definition 3.2 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 [12] discusses a stronger version of authentication which requires each authenticating event to correspond to a different authenticated event—he terms this requirement “injectivenes.” Such a property is important in, for example, the authorisation of financial transactions. This property can be captured by strengthening the above definition as follows:

$$R \ \mathtt{injectively\ precedes}\ T \mathrel{\widehat{=}} \#(tr \restriction R) \geq \# tr \restriction T)$$

where # *tr* denotes the length of the trace *tr*. The property *R* **injectively precedes** is strictly stronger than *R* **precedes** *T*.

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

This approach contrasts with that taken in [4], [19] 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 *G*4, which states that "the origin of all messages in the protocol has to be authenticated." Other examples are given in [20], which gives the 'canonical' intensional specification as one where the interleavings of events at different agents are in accordance with the messages in the protocol; and a slightly weaker one 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.

A different process algebraic approach is taken by Abadi and Gordon [1], where the pi-calculus [15] 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 modeledexplicitly. 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 a rank function ρ as the core of a proof is complementary to the tool-based approaches [10], [14], [9], [19] which search for attacks. The results of the latter kind of analysis provide useful information as to why a protocol is not correct, or alternatively give 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 rank function 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 [11] 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 rank function approach might also be useful in this context.

Paulson [18] 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 steps 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 events can never occur in a trace, and this requires certain lemmas establishing that particular classes of event cannot occur. This is also a feature of the approach taken in [13], 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 [17] is encouraging, but at present[1] fit 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 modeledas 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 'idealization' of protocols into the logic means that the link between a protocol and its logical treatment is informal. The formal language described in [23] 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 analyzer which is used to

1. As of April 1998.

check reachability of negated requirements, so it is closer to the tool-based approaches. However, an approach to proof reflecting that presented in this paper seems feasible.

6.2 Future Directions

This paper has presented an approach to analyzing 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.

Some form of mechanical assistance for proof has now been provided within PVS [17] by Bruno Dutertre at Royal Holloway. A CSP trace semantics has been provided for the operators used in this paper, the various proof rules have all been proven sound, and all of the theorems have been proven. Correctness of the protocols in this paper with respect to their various properties has been established. An important avenue to explore will be the extent to which construction of the rank function can be assisted. In effect, a proof can be provided with respect to particular constraints on the rank function, and the proof is completed once a function can be provided which meets all those constraints.

APPENDIX

THEOREM 3.5. *If*

$[R1:]\ \forall\, m \in INIT \bullet \rho(m) > 0$

$[R2:]\ \forall\, S, m \bullet ((\forall\, s \in S \bullet \rho(m) > 0) \wedge S \vdash m \Rightarrow \rho(m) > 0)$

$[R3:]\ \forall\, t \in T \bullet \rho(m) \leq 0$

$[R4:]\ \forall\, i \bullet (USER_i \,|[R]|\, Stop\ \mathbf{sat}\ \mathtt{maintains}\ \rho)$

then $NET\ \mathbf{sat}\ \mathtt{precedes}\ T$

Proof. By Lemma 3.3 it is sufficient to establish that $R1$–$R4$ imply $NET\ |[R]|\ Stop\ \mathbf{sat}\ tr \upharpoonright T = \langle\rangle$.

Assume for a contradiction that R1–R4 hold, and also that $\neg\,(NET\ |[R]|\ Stop\ \mathbf{sat}\ tr \upharpoonright T =]])$. Then there is some trace $tr \in traces(NET\ |[R]\ Stop)$ for which $tr \upharpoonright T \neq \langle\rangle$. Since $R3$ tells us that $\rho(t) \leq 0$ for any $t \in T$ we have that there are some messages in tr with rank less than 1. Let tr_0 be the prefix of tr whose final message is the first message of tr of rank 0 or less. In other words, tr_0 is the trace up to the point where the first message of rank 0 or less appears. By prefix closure of traces in processes, tr_0 is a trace of $NET\ |[R]|\ Stop$.

Now consider the last message of tr_0. It is either of the form $rec.i.j.x$ or $trans.i.j.x$ for some i, j, and x, where $\rho(x) \leq 0$

Case *rec.i.j.x*. We have that tr_0 is a trace of $ENEMY$. Hence by Theorem 3.1 we have that $(INIT \cup (tr_0 \Downarrow trans)) \vdash tr_0 \Downarrow rec$ and so $(INIT \cup (tr_0 \Downarrow trans)) \vdash x$. But by the definition of tr_0 we have $\forall\, m \in tr_0 \Downarrow \text{trans} \bullet \rho(m) \geq 1$, since all messages in tr_0 apart from the last have strictly positive rank, so $R1$ and $R2$ yield that $\rho(x) \geq 1$, forcing a contradiction.

Case *trans.i.x*. Let $tr_i = tr_0 \upharpoonright \{trans.i, rec.i\}$. This is the subsequence of tr_0 in which $USER_i\ |[R]|\ Stop$ participates, so tr_i is a trace of $USER_i\ |[R]|\ Stop$. Hence, by $R4$ we have $\mathtt{maintains}\ \rho(tr_i)$. Expanding the definition we find

$$(\forall\, m \in tr_i \Downarrow rec.i \bullet \rho(m) \geq 1)$$
$$\Downarrow (\forall\, m \in tr_i \Downarrow trans.i \bullet \rho(m) \geq 1)$$

The definition of tr_0 means that the antecedent of this implication is true, from which it follows that $\rho(x) \geq 1$, yielding a contradiction.

In either case we find a contradiction, which establishes the theorem. □

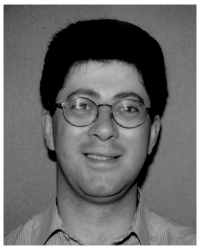
ACKNOWLEDGMENTS

The author thanks Peter Ryan, Irfan Zakiuddin, Bruno Dutertre, James Heather, Gavin Lowe, Jeremy Bryans, Andy Gordon, and Catherine Meadows for their comments on earlier forms of this work, and Chris Mitchell for some enlightening discussions on authentication. James Heather made the observation that confidentiality could be treated as a special case of authentication with the empty set for R. The author is also grateful to the anonymous referees for their careful reading of an earlier version of this paper and their suggestions for improvement. It is also a pleasure to thank Bruno Dutertre for his work on checking the proofs with PVS and suggesting improvements, underlining the necessity for mechanical assistance to make such proofs manageable. Funding for this research was received from the Defence Research Agency.

REFERENCES

[1] M. Abadi and A.D. Gordon, “A Calculus for Cryptographic Protocols: The spi Calculus,” technical report, Univ. of Cambridge, 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,* vol. 5, no. 6, 1993.
[4] W. Diffie, P.C. van Oorschot, and M.J. Wiener, “Authentication and Key Exchanges,” *Designs, Codes, and Cryptography,* vol. 2, 1992.
[5] D. Dolev and A.C. Yao, “On the Security of Public Key Protocols,” *IEEE Trans. Information Theory*, vol. 29, no. 2, 1983.
[6] D. Gollmann, “What Do We Mean by Entity Authentication,” *IEEE Computer Society Symp. Research in Security and Privacy*, 1996.
[7] C.A.R. Hoare, *Comm. Sequential Processes.* Prentice Hall, 1985.
[8] International Organisation for Standardisation, *Information Processing Systems—Open Systems Interconnection—Basic Reference Model—Part 2: Security Architecture*, ISO 7498-2. 1989.
[9] R. Kemmerer, C. Meadows,. and J. Millen, “Three Systems for Cryptographic Protocol Analysis,” *J. Cryptology*, vol. 7, no. 2, 1994.
[10] G. Lowe, “An Attack on the Needham-Schroeder Public-Key Authentication Protocol,” *Information Processing Letters*, no. 56, 1995.
[11] G. Lowe, “Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR,” *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1055. Springer-Verlag, 1996.
[12] G. Lowe, “A Hierarchy of Authentication Specifications,” *Proc. Computer Security Foundations Workshop*, no. 10, 1997.
[13] C. Meadows, “Applying Formal Methods to the Analysis of a Key Management Protocol,” *J. Computer Security*, vol. 1, no. 1, 1992.
[14] J. Millen, “The Interrogator Model,” *IEEE Computer Society Symp. Research in Security and Privacy*, 1995.

[15] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes," *Information and Computation*, no. 100, 1992.
[16] R. Needham and M.L. Schroeder, "Using Encryption for Authenticiation in Large Networks of Computers," *Comm. ACM*, vol. 21, no. 12, 1978.
[17] S. Owre, N. Shankar, and J. Rushby, "The PVS Specification Language," technical report, Computer Science Laboratory, SRI Int'l, 1993.
[18] L.C. Paulson, "Proving Properties of Security Protocols by Induction," technical report, Univ. of Cambridge, UK, 1996.
[19] A.W. Roscoe, "Modelling and Verifying Key-Excfhange Protocols Using CSP and FDR," *Proc. Computer Security Foundations Workshop*, no. 8, 1995.
[20] A.W. Roscoe, "Intensional Specifications of Security Protocols," *Computer Security Foundations Workshop*, no. 9, 1996.
[21] A.W. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1997.
[22] S.A. Schneider, "Security Properties and CSP," *IEEE Computer Society Symp. Research in Security and Privacy*, 1996.
[23] P. Syverson and C. Meadows, "A Formal Language for Cryptographic Protocol Requirements," *Designs, Code,s and Cryptography*, no. 7, 1996.
[24] T. Woo and S. Lam, "A Semantic Model for Authentication Protocols," *Proc. IEEE Computer Society Symp. Research in Security and Privacy*, 1993.



Steve Schneider received his BA degree in mathematics and philosophy, and the MSc and DPhil degrees in computer science from the University of Oxford on Timed CSP. He worked at the Oxford Programming Research Group as a research assistant on the ESPRIT project SPEC from 1989–1991, and then as an SERC postdoctoral fellow and a junior research fellow at Balliol College, Oxford. Since 1994, he has been a lecturer in computer science at Royal Holloway, University of London. His research interests include concurrency, real-time systems, formal methods, and security. Schneider is a member of the IEEE Computer Society.