A Hierarchy of Authentication Specifications

Gavin Lowe
Department of Mathematics and Computer Science
University of Leicester, University Road
Leicester, LE1 7RH, UK
E-mail: gavin.lowe@mcs.le.ac.uk

Abstract

Many security protocols have the aim of authenticating one agent to another. Yet there is no clear consensus in the academic literature about precisely what "authentication" means. In this paper we suggest that the appropriate authentication requirement will depend upon the use to which the protocol is put, and identify several possible definitions of "authentication". We formalize each definition using the process algebra CSP, use this formalism to study their relative strengths, and show how the model checker FDR can be used to test whether a system running the protocol meets such a specification.

1 Introduction

Many security protocols have appeared in the academic literature; these protocols often have the aim of achieving *authentication*, i.e., one agent should become sure of the identity of the other. The protocols are designed to succeed even in the presence of a malicious agent, called an *intruder*, who has complete control over the communications network, and so can intercept messages, and introduce new messages into the system, possibly using information from messages he has seen. However, it is rarely made clear precisely what is meant by the term "authentication". This may be dangerous, for a user may assume that the protocol satisfies a stronger condition than the one that was intended by the protocol designer, and so may place more reliance upon the protocol than is justified. In order to alleviate this problem, we study various possible meanings of the word authentication. We formalize these meanings using the process algebra CSP [8].

Suppose an agent A completes a run of an authentication protocol, apparently with B; then what can A deduce about the state of B? Can A deduce that B has recently been alive? Can A deduce that B has recently been running the same protocol as A? Maybe A can deduce that B thought he was running the protocol with A (as opposed to some third party, C). And maybe the two agents agree upon who initiated the exchange, and who responded. Further, they may agree upon the values of some or all of the data items (such as nonces and keys) used in the run. Can A assume that there was a one-one relationship between his runs and B's runs, or might it be the case that A has completed more runs than B? And can A deduce that he and B agreed upon the contents of all messages sent from one to the other?

It is my experience that different researchers will give different answers to the above questions. In order to reason and argue about authentication protocols, we must first of all define what we mean by "authentication". My own view is that an authentication protocol is designed to assure an agent A as to the identity of the other agent B with whom A is running the protocol; therefore, in most cases A should at least be assured that B thought he was running the protocol with A. However, some researchers take the view that it is enough for B to be present, and that A need not receive any further assurance as to B's current state. We should recognize that the different authentication specifications may all be valid goals: there are circumstances in which the weaker specifications are all that is needed; in other circumstances, a stronger specification may be required. However, the designer of any protocol should make it clear which form of authentication is supposed to be achieved.

In this paper we will introduce different terms corresponding to some of the possible meanings of the word "authentication" considered above. We formalize each of these meanings using the process algebra CSP [8]. The use of CSP has two advantages:

- It will allow us to compare the strengths of the different authentication specifications;
- It will open up the possibility of automatically checking whether a system running the protocol achieves the goals stated for it, using a model checker such as FDR [19, 6].

We should stress, though, that most of our results are independent of CSP: CSP just provides a convenient tool for formalizing and reasoning about the specifications.

Most of the possible meanings of authentication refer to the *recent* state of an agent B when another agent A completes a run of the protocol, apparently with B. For example, a typical authentication specification (which we will call *recent aliveness* below) is that when A completes a run of the protocol, apparently with B, then B has *recently* been running the same protocol. It turns out that the recentness is the hardest part of the specification to formalize in CSP: it normally requires modelling the passage of time. To simplify our presentation, we begin by considering authentication specifications in the case where recentness is not required; for example, we consider a specification (called *aliveness* below) that states that when A completes a run of the protocol, apparently with B, then B has previously (not necessarily recently) been running the same protocol. Later we lift our specifications to include recentness.

It is my experience that most attacks upon protocols break the weaker specifications where recentness is not required; thus from a pragmatic point of view, when using a tool such as FDR to look for attacks upon a protocol, it is sensible to begin by looking for attacks on these weaker specifications, since these tests are faster than in the cases including recentness.

Note that we will not directly consider questions of secrecy in this paper. It is not difficult to formalize secrecy within CSP, using techniques similar to those discussed here. Secrecy and authentication specifications are often broken in the same way: there are a number of attacks upon protocols that lead to an agent A thinking he has established a key with another agent B, when in fact he has been running the protocol with an intruder imitating B, and the intruder ends up knowing the key; this is a failure of authentication, because B has been incorrectly authenticated, and a failure of secrecy, because the intruder has learnt the key that was supposed to remain secret.

In the next section we give precise, although informal, definitions of authentication, first in the case where recentness is not required, and then in the case where recentness is required. In Section 3 we describe the CSP approach to modelling security protocols, and set up some of the mechanism for formalizing the authentication properties. In Section 4 we formalize the authentication properties without recentness, and prove a number of results relating them. Then in Section 5, we lift these specifications to include recentness. We sum up in Section 6. Because of limitations on space, we omit all proofs from this paper; the interested reader is referred to [12].

2 Forms of authentication

In this section we identify four different reasonable meanings of the word "authentication". But first, we need to discuss the setting in which the definitions will apply.

We will consider protocols that aim to authenticate a *responder* B to an *initiator* A, possibly with the help of a third party, a *server*. We use the word "role" to refer to the part an agent is taking in the protocol run (i.e. initiator, responder or server). It should be obvious how to generalize these ideas to include extra agents playing additional roles, or to reverse the direction of authentication.

We do not restrict ourselves to the case where a particular agent may only ever adopt a single role; on the contrary, an agent may act as an initiator in some runs, and as a responder in other runs, and possibly even as a server in others (although this latter case would be unusual).

It is worth drawing a distinction between the free variables appearing in a description of a protocol, and the actual values with which those free variables are instantiated. For example, in a protocol description, the free variable A is often used to represent the initiator; in actual runs, this variable will be instantiated with the identities of actual agents, often different identities in different runs. We will denote free variables representing agents by single letters (A, B, etc.) and will denote actual agents' identities by proper names ($Alice$, Bob, etc.); for other data items, we will use small letters for free variables (na, kab, etc.), and names beginning with a capital letter for actual values (Na, Kab, etc.).

As explained in the introduction, we begin by considering the cases without recentness, and then extend the definitions to include recentness.

2.1 Aliveness

The following is what we consider to be the weakest reasonable definition of authentication.

Definition 2.1 (Aliveness). We say that a protocol guarantees to an initiator A *aliveness* of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.

Note that B may not necessarily have believed that he was running the protocol with A. Also, B may not have been running the protocol *recently*.

Many protocols fail to achieve even this weak form of authentication. In several cases, this is due to an intruder launching a mirror attack, simply reflecting an agent's messages back at himself; examples appear in [2]. In other, more subtle attacks, an intruder attacks an agent A by using a second run of a protocol with the same agent A, so as to use the second run as an oracle; for example the attack on the BAN version of the Yahalom protocol [3] in [24]. Other attacks are due to more blatant errors; for example, the attack on the SPLICE protocol [25] in [9], which exploits the fact that key delivery messages (from a key server) do not include the identity of the agent whose key is being delivered.

Closely related to the notion of aliveness is the case where, when A completes a run of the protocol, apparently with B, then B has previously been present, but not necessarily running the protocol in question—it may be that B has been running a completely different protocol. This raises the question of interaction between protocols, where an intruder can learn information in one protocol that he can use in an attack on another protocol. In this paper we mainly restrict our attention to systems running a single protocol; we briefly discuss how to extend these techniques to cover systems running several protocols in Section 4.5. However, in general it will be very difficult to prove results about a system running several protocols: we need to be sure that no protocol acts as an oracle for any other.

2.2 Weak agreement

We strengthen the above definition to insist that B agreed he was running the protocol with A.

Definition 2.2 (Weak agreement). We say that a protocol guarantees to an initiator A *weak agreement* with another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.

Note that B may not necessarily have been acting as responder.

Several protocols achieve a guarantee of liveness, but fail to guarantee weak agreement. The normal scenario is that the intruder imitates an agent B to attack A, by using B as an oracle in a parallel run in which the intruder adopts his own identity; thus A believes he has been running the protocol with B, but B does not believe he has been running the protocol with A—B thinks he has been running the protocol with the intruder. Examples include my attack on the Needham-Schroeder Public Key protocol [16] in [10].

2.3 Non-injective agreement

The following definition adds the condition that the two agents agree as to which roles each was taking, and that they agree upon some of the data items used in the exchange.

Definition 2.3 (Non-injective agreement). We say that a protocol guarantees to an initiator A *non-injective agreement* with a responder B on a set of data items ds (where ds is a set of free variables appearing in the protocol description) if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds.

Note that this does not guarantee that there is a one-one relationship between the runs of A and the runs of B (hence the adjective "non-injective"): A may believe that he has completed two runs, when B has only been taking part in a single run.

A few protocols achieve a guarantee of weak agreement, but not non-injective agreement. For example, if the original Andrew protocol [21] is adapted to detect mirror attacks, then it achieves weak agreement, but does not achieve non-injective agreement on all the data values: an attack in [3] shows how an intruder can get A to accept a key different from the one used by B.

2.4 Agreement

We use the term, "*injective agreement*", or simply "*agreement*", when we want to insist that there is a one-one relationship between the two agents' runs. This one-one relationship may be important in, for example, financial protocols.

Definition 2.4 (Agreement). We say that a protocol guarantees to an initiator A *agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a *unique* run of B.

We will use the term *full agreement* to refer to agreement on all the atomic data items used in the protocol run. For various reasons, we consider this to be the most useful definition of authentication: it insists that the two agents agree on all the essential features of the protocol run, while avoiding specifying features that are hard to achieve and less likely to be required.

A few protocols achieve non-injective agreement, but not (injective) agreement: an agent A is tricked into thinking that B is trying to establish two sessions with him, whereas B was only trying to establish a single run. For example, in the Kerberos protocol [15], the freshness of one agent is guaranteed only by a timestamp; thus, an intruder can replay these messages (within the lifetime of the timestamp) to complete a second run; note that this attack assumes that the agents do not check that the timestamps they receive are distinct from all previous timestamps; Bellovin and Merritt report [1] that early implementations did not perform this check. Similar attacks are described in [13].

2.5 Recentness

Finally, we lift the above definitions to ensure recentness. The meaning of "recent" will depend on the circumstances: sometimes we will take it to mean within the duration of A's run; sometimes we will take it to mean at most t time units before A completed his run, for suitable t, called the *authentication time* (clearly the value of t will be implementation dependent); the designer or implementer of the protocol should make clear what degree of recentness is guaranteed. We use the terms *recent aliveness*, *recent weak agreement*, *recent non-injective agreement*, and *recent agreement* to refer to the above specifications strengthened to insist that B's run was recent, rather than just at some time in the past; we will add the phrase "within t time units" where the protocol guarantees a particular authentication time.

Some protocols meet a particular authentication specification without recentness, but fail to meet the corresponding specification with recentness. For example, consider the following one-step protocol, where kab is a key shared between A and B:

$$\text{Message 1.} \quad A \rightarrow B \ : \ \{A, k\}_{kab} \ .$$

This protocol gives B a guarantee of non-injective agreement on k, but gives no guarantee of recentness, because the message contains no information that B knows to be fresh. The protocol can be strengthened to achieve recent non-injective agreement by adding a timestamp to the message.

Also, in a setting where key compromise is possible, there is a well known attack on the Needham-Schroeder Secret Key Protocol [16], presented in [4]: once a key has been compromised, the intruder may replay messages from an earlier protocol run so as to imitate the initiator; thus (if we make the reasonable assumption that compromising keys takes quite a long time) the protocol guarantees non-injective agreement, but not recent non-injective agreement.

2.6 Discussion

All of the above definitions used a phrase of the form "B has previously been running the protocol". We take this to mean that B has progressed at least as far as the last message that B sends; clearly A can never be assured that B received any subsequent messages. Note that it is possible to define weaker specifications, where A is only assured that B has at least started the protocol.

It should be obvious that the above forms of authentication without recentness are in increasing order of strength, as are the forms of authentication with recentness. Further, each definition using recentness is stronger than the corresponding definition without recentness. We will prove these facts later, after we have formalized the authentication specifications.

2.7 Comparisons

Roscoe [20] introduces the notion of an intensional specification:

> No node can believe a protocol run has completed unless a correct series of messages has occurred (consistent as to all the various parameters) up to and including the last message the given node communicates.

That is, if an agent completes a protocol run, then the protocol must have proceeded essentially as the protocol designer intended: each agent must have seen the expected sequence of messages, all agreeing on the values of atomic data items, and with the correct relative orders of messages.

Roscoe shows how to produce a CSP representation of the above specification; this specification can be checked using the model checker FDR, in a setting very similar to our own.

This is a strong definition of authentication, stronger than our definition of full agreement. It is at least as strong as full agreement, for suppose a protocol satisfies the intensional specification, and that an agent A believes it has completed a protocol run with B. Then from the intensional specification, B's view of the protocol run must agree with A's, and in particular B must have thought he was running the protocol with A, and the two agents agree upon the roles each take, and upon the values of all atomic data items; further, from the way in which intensional specifications are formalized in CSP, there must be a one-one relationship between A's runs and B's runs. Hence full agreement is achieved.

The intensional specification is strictly stronger than full agreement for two reasons:

- In some protocols, an agent A receives an encrypted component that he is not supposed to decrypt, but merely forward to another agent B. In these cases, there is a simple attack where the intruder replaces this component with an arbitrary component, but then reverses the switch when A tries forwarding the component to B. This would not be considered an attack when using the full agreement specification, but is an attack under the intensional specification.
- Consider a protocol where a server sends two consecutive messages to A and to B, respectively. The intensional specification would insist that A and B receive these messages in the same order. However, there is a simple attack where the intruder delays A's message so that it arrives just after

B receives his message. However, this would not be considered an attack when using the full agreement specification.

It is arguable whether the above two attacks should really be considered as attacks. However, there are settings where the precise contents or the precise orderings of messages may be important. We would stress again our philosophy that protocol designers should specify precisely what their protocols are supposed to achieve.

The intensional specification does not, in general, give any guarantee of recentness. For example, consider the one step protocol from Section 2.5; this meets the intensional specification, but does not achieve recent authentication. However, in most protocols recentness is guaranteed by the way in which messages are interleaved: suppose the protocol is such that A sends a message to B and later receives a message back from B (possibly via third parties); suppose further that A is implemented to time out if the run is taking too long, so any actions that occurred since A started the run should be considered recent; then if the intensional specification is met, then it must be the case that the corresponding actions of B occurred after A started this protocol run, so B's actions must indeed have been recent. Roscoe discusses other ways in which intensional specifications can be adapted to deal with time.

Diffie, van Oorschot and Wiener [5] have a similar definition of authentication. They specify that when an agent A completes a run of the protocol, apparently with B, then B has been running the protocol, and the two agents' records of the runs *match*: that is, the messages each sees are the same in all fields relevant to authentication, and the agents agree upon which messages were incoming and which outgoing. However, the definition of matching is such that B may not necessarily think that he was running the protocol with A. Thus this definition is weaker than our weak agreement specification.

Gollmann [7] defines four goals of authentication protocols. We consider here two of these goals, which concern whether a protocol authenticates B to A. Goal G3 states that a cryptographic key associated with B has to be used during the protocol run. (The term "cryptographic key" is intended to be interpreted fairly broadly, so as to include, for example, shared secrets used for authentication; in the case where a key is shared between two agents, the word "associated" is supposed to be interpreted as referring to the agent who actually used the key in question.) The reasoning behind G3 is that B should be authenticated only if an action has occurred that must have been performed by B. Hence this is similar to our recent aliveness specification.

Goal G4 states that the origin of all messages in the protocol has to be authenticated. In other words, if A receives a message, apparently from B, then B previously tried to send this message to A. It is left vague whether B necessarily sent this message recently, and whether A may receive two messages for a single message sent by B, and so the above goal can be interpreted in different ways. However, this goal is clearly similar to Roscoe's intensional specification.

Paulson [17, 18] uses the theorem prover Isabelle to analyse security protocols. He proves properties of the form: if A receives a message of a certain form, which appears to come from B, then B indeed sent that message. Thus his notion of authentication is similar to our non-injective agreement. (The form of authentication he considers is necessarily non-injective, because of a feature of his model that allows an agent to respond several times to a single message; there is no real reason why this feature could not be changed, in which case he would be able to deal with injective agreement, although possibly at additional computational expense.)

3 Modelling protocols using CSP

In this section we briefly review the method we use for modelling security protocols using CSP. For a fuller description, the reader is referred to [11]. All the authentication specifications we are considering are safety specifications (as opposed to liveness specifications); we will therefore be working in the traces model of CSP, which is adequate for expressing safety properties.

As an example, we consider the 3 message version of the Needham-Schroeder Public Key protocol [16, 10]. The protocol can be defined by:

$$\begin{array}{lll} \text{Message 1.} & A \to B & : \; A, B, \{na, A\}_{PK(B)} \\ \text{Message 2.} & B \to A & : \; B, A, \{na, nb\}_{PK(A)} \\ \text{Message 3.} & A \to B & : \; A, B, \{nb\}_{PK(B)} \; . \end{array}$$

Each agent is represented by a CSP process that sends and receives the appropriate protocol messages, augmented with extra events that signal the beliefs of the agent. For example:

$$\begin{array}{l} INITIATOR_0(A, na, pka) \mathrel{\widehat{=}} \\ \quad \Box_{B:Agent}\, env.A.(Env0, B) \to \\ \quad comm.A.B.(Msg1, Encrypt.(PK(B), \langle na, A\rangle)) \to \\ \quad \Box_{nb:Nonce} \\ \qquad comm.B.A.(Msg2, Encrypt.(pka, \langle na, nb\rangle)) \to \\ \quad signal.Running.INIT\text{-}role.A.B.na.nb \to \\ \quad comm.A.B.(Msg3, Encrypt.(PK(B), \langle nb\rangle)) \to \\ \quad signal.Commit.INIT\text{-}role.A.B.na.nb \to \\ \quad SKIP \end{array}$$

The way in which protocol messages are represented by CSP events should be obvious; for example, the event

$$comm.A.B.(Msg1, Encrypt.(PK(B), \langle na, A\rangle))$$

represents the message:

$$\text{Message 1.} \quad A \to B \; : \; \{na, A\}_{PK(B)} \; .$$

The channel $signal$ is used to express properties of the beliefs of the agent. An event of the form $signal.Running.role.A.B.na.nb$ represents that the agent A believes that he is taking the role $role$ in a run of the protocol with B, using nonces na and nb; such an event is performed immediately before the last message that A sends. An event of the form $signal.Commit.role.A.B.na.nb$ represents that the agent A thinks he has successfully completed a run of the protocol, taking role $role$, with B, using the nonces na and nb; such an event is performed after A's last event in the protocol. These $signal$ events will be used in formalizing our authentication specifications. The commit and running signals include (in some standard order) all the data items that the agents are supposed to agree on.

In order to represent the fact that the intruder may interfere with the normal operation of the protocol, a CSP-renaming is applied to the above processes, with the effect that outputs may be intercepted by the intruder (and so fail to reach their intended destination), and inputs may be faked (that is, produced by the intruder).

The parameters of these processes are instantiated with appropriate actual values; this allows us, for example, to define several $INITIATOR$ processes with the identity $Alice$, representing that the agent $Alice$ may run the protocol several times.

The processes are then combined together in parallel with a process representing the most general intruder who can interact with the protocol. This intruder may: oversee messages passing in the system, and so learn new messages; intercept messages; decrypt messages he has seen, if he has the appropriate key, and so learn new (sub-)messages; encrypt messages he has seen with keys that he knows so as to create new messages; and use the knowledge he has accumulated to send fake messages.

FDR can be used to check whether the resulting system correctly achieves the goals of the protocol. Formalizing these goals is the subject of the rest of this paper.

4 Formalizing authentication specifications without recentness

As explained in the introduction, we begin by formalizing authentication specifications that ignore issues of recentness. In the next section we adapt these specifications to include recentness.

We introduce two pieces of notation that we will need. If P is a process, we define αP to be its alphabet. We write P^n for the interleaving of n copies of P:

$$P^n \;\widehat{=}\; \underbrace{P \,|||\, \ldots \,|||\, P}_{n}$$

Formally: $P^0 \;\widehat{=}\; STOP$ and $P^{n+1} \;\widehat{=}\; P \;|||\; P^n$. We will take $\alpha P^0 = \alpha P$.

We now formalize the different authentication specifications; we consider the specifications in the reverse order to that in the introduction. We will consider a pair of agents, A and B, taking the roles $A\text{-}role$ and $B\text{-}role$, respectively, and consider the question of whether the protocol correctly authenticates B to A.

4.1 Agreement

We begin with the *agreement* specification. Suppose the protocol uses two data items d_1 and d_2, and we want to test whether the protocol guarantees to an arbitrary agent A taking the role $A\text{-}role$, agreement with an agent B, taking the role $B\text{-}role$, on both the data items d_1 and d_2. This will be the case if, whenever A completes a run of the protocol apparently with B, then B has previously been running the protocol, apparently with A, using the same values for the data items, and there is a one-one relationship between A's runs and B's runs. Now, B running the protocol, taking the role $B\text{-}role$, apparently with A, using particular values d_1' and d_2' for the data items d_1 and d_2, is represented by the event $signal.Running.B\text{-}role.B.A.d_1'.d_2'$; similarly, A completing the protocol, taking the role $A\text{-}role$, apparently with B, using the same values for the data items, is represented by the event $signal.Commit.A\text{-}role.A.B.d_1'.d_2'$. Thus we will want to ensure that the abstraction of the system to the appropriate alphabet satisfies the following specification:

$$\begin{aligned}&Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \;\widehat{=}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2' \to\\ &\quad signal.Commit.A\text{-}role.A.B.d_1'.d_2' \to STOP\;.\end{aligned}$$

This specification says that whenever A performs an event of the form $signal.Commit.A\text{-}role.A.B.d_1'.d_2'$, then B has previously performed a corresponding event $signal.Running.B\text{-}role.B.A.d_1'.d_2'$. That is, whenever A signals that he thinks he has completed the protocol, taking the role $A\text{-}role$, with B, using the data values d_1' and d_2', then B has previously been running the protocol, taking the role $B\text{-}role$, with A, using the same data values. (Remember that we are working in the traces model of CSP, so the above specification does not insist that a $signal.Commit$ event must occur.)

Figure 1 illustrates the meanings of the arguments.

Consider the case where each agent may take part in a single run of the protocol. We may test whether the protocol correctly authenticates a particular agent B, taking role $B\text{-}role$, to an arbitrary agent A, taking role $A\text{-}role$,

Role of agent being authenticated
Role of agent to whom authentication is made
Data items agreed upon
Identity of agent being authenticated

$$Agreement(B\text{-}role, A\text{-}role, ds)(B)$$

Figure 1. The arguments of $Agreement$

with agreement on all the data items, by testing:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \sqsubseteq \\ \quad SYSTEM \setminus (\Sigma - X) \\ \text{where} \\ X = \alpha Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B)), \end{array}$$

where Σ is the set of all events. (The fact that A's identity is not included in the parameters of the process $Agreement$ has the effect of making the above refinement check test whether B is correctly authenticated to *all* agents who take the role $A\text{-}role$. The reason why we test whether a *single* agent B is correctly authenticated, rather than testing whether all agents who take the role $B\text{-}role$ are correctly authenticated, is purely pragmatic: the latter check would become very slow for large systems. The reason for the slightly odd parameterization is to keep consistency with the tool Casper, which we will discuss in the concluding section, which automatically produces CSP descriptions of security protocols, and specifications similar to the above, from a more abstract definition.)

The above form of refinement check will be very common in the rest of this paper, so we introduce a shorthand for it.

$$\begin{array}{l} SYSTEM \ \mathsf{meets} \ SPEC \ \widehat{=} \\ \quad SPEC \sqsubseteq SYSTEM \setminus (\Sigma - \alpha SPEC). \end{array}$$

So the above test is written:

$$\begin{array}{l} SYSTEM \ \mathsf{meets} \\ \quad Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B). \end{array}$$

For example, to test whether the Needham-Schroeder Public Key Protocol guarantees to an arbitrary responder B agreement with the initiator $Alice$ on the nonces na and nb, we should test:

$$\begin{array}{l} SYSTEM \ \mathsf{meets} \\ \quad Agreement \\ \qquad (INIT\text{-}role, RESP\text{-}role, \{na, nb\})(Alice). \end{array}$$

It turns out that the above specification is not met (the protocol does not correctly authenticate the initiator), as can be seen from the attack in [10, 11].

We now generalize the above refinement checks to consider the case where the agent being authenticated can perform more than one run of the protocol. In this case, we want to ensure that *every* $Commit$ event is matched by a corresponding $Running$ event. If B can perform at most n runs, then there will be at most n $Running$ events, so we want to consider the following refinement check:

$$\begin{array}{l} SYSTEM \ \mathsf{meets} \\ \quad Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B)^n. \end{array}$$

Note that this specification guarantees that only one $Commit$ event may correspond to each $Running$ event, thus ensuring that the initiator commits to only a single session for each run of the protocol performed by the responder.

In some cases it will be the case that we want to ensure that the agents involved agree on the values of some data items, but allow them to disagree on other data items. For example, if we want the agents to agree on d_1, but allow them to disagree on d_2, then we may use the specification:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{d_1\})(B) \ \widehat{=} \\ \quad signal.Running.B\text{-}role.B?A?d_1'?d_2'' \to \\ \quad signal.Commit.A\text{-}role.A.B.d_1'?d_2' \to STOP \end{array}$$

If we do not insist that they agree on any data items, then we may use the specification:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{\})(B) \ \widehat{=} \\ \quad signal.Running.B\text{-}role.B?A?d_1''?d_2'' \to \\ \quad signal.Commit.A\text{-}role.A.B?d_1'?d_2' \to STOP \end{array}$$

In general, the third argument of the $Agreement$ macro should be a subset of the data variables appearing in the $signal.Running$ and $signal.Commit$ events.

In the above, we have considered a protocol using a pair of data values; it should be obvious how to generalize to an arbitrary set of data variables ds. We will write $signal.Running.B\text{-}role.B.A.ds'$ to indicate agent B thinking that he is running the protocol using values ds' for the data variables ds, and similarly for the $Commit$ signal (strictly speaking, ds is a *set*, and ds' is an enumeration of the elements of ds, in some standard order).

It should be obvious that if a protocol guarantees agreement on some set ds_1 of data values, then it will guarantee agreement on any smaller set ds_2. The following lemma formalizes this.

Lemma 4.1. If $ds_2 \subseteq ds_1$ and

$$SYSTEM \text{ meets } Agreement(B\text{-}role, A\text{-}role, ds_1)(B)^n,$$

then

$$SYSTEM \text{ meets } Agreement(B\text{-}role, A\text{-}role, ds_2)(B)^n.$$

4.2 Non-injective agreement

In the previous subsection, we insisted that there was a one-one relationship between the runs of A and those of B. In some settings, this injectivity may not be important, and so we will allow A to commit an arbitrary number of times, for each run of B:

$$\begin{aligned}&NonInjectiveAgreement\\ &\quad(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \mathrel{\widehat{=}}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.d_1'.d_2'\}).\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role $A\text{-}role$, with B, using data values d_1' and d_2' (i.e., perform an arbitrary number of $signal.Commit.$ $A\text{-}role.A.B.d_1'.d_2'$ events) if B thinks he has been running the protocol, taking role $B\text{-}role$, with A, using the same data values, at least once (i.e., if he has performed at least one $signal.Running.B\text{-}role.B.A.d_1'.d_2'$ event).

This can be adapted to insist upon agreement of only some of the data values; for example:

$$\begin{aligned}&NonInjectiveAgreement(B\text{-}role, A\text{-}role, \{d_1\})(B) \mathrel{\widehat{=}}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2'' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.d_1'.d_2' \mid\\ &\qquad d_2' \in Data\}).\end{aligned}$$

Non-injective agreement can then be tested analogously to agreement:

$$SYSTEM \text{ meets } NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^n.$$

As with injective agreement, agreement on some set of data items implies agreement on any smaller set.

Lemma 4.2. If $ds_2 \subseteq ds_1$ and:

$$SYSTEM \text{ meets } NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds_1)(B)^n,$$

then:

$$SYSTEM \text{ meets } NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds_2)(B)^n.$$

Non-injective agreement is a weaker requirement that agreement; this is formalized by the following lemma.

Lemma 4.3. If

$$SYSTEM \text{ meets } Agreement(B\text{-}role, A\text{-}role, ds)(B)^n,$$

then

$$SYSTEM \text{ meets } NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^n.$$

4.3 Weak agreement

We can weaken the previous specification, so that A receives no guarantee as to which role B thought he was taking, and no guarantee regarding agreement on data. We define the specification:

$$\begin{aligned}&WeakAgreement(A\text{-}role)(B) \mathrel{\widehat{=}}\\ &\quad signal.Running?B\text{-}role!B?A?ds' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.ds \mid\\ &\qquad ds \in Data^*\}).\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role $A\text{-}role$, with B, if B thinks he has been running the protocol, taking some role $B\text{-}role$, with A, possibly using different data values, at least once.

Weak agreement can then be checked using the refinement test:

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^n.$$

The following lemma shows that weak agreement is indeed a weakening of non-injective agreement.

Lemma 4.4. Suppose the agent B can perform a maximum of m runs with role $B\text{-}role$, and n runs with other roles. If:

$$SYSTEM \text{ meets } NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^m,$$

then:

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^{m+n}.$$

4.4 Aliveness

Finally, we weaken the specification still further, so that A receives no guarantee that B thought he was running the protocol with A; A merely receives a guarantee that B was previously running the protocol with somebody:

$$\begin{aligned}&Aliveness(A\text{-}role)(B) \mathrel{\widehat{=}}\\ &\quad signal.Running?B\text{-}role!B?C?ds' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.ds \mid\\ &\qquad ds \in Data^* \land A \in Agent\})\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role $A\text{-}role$, with B, if B thinks he has previously been running the protocol.

Agreement can then be checked using the refinement test:

$$SYSTEM \text{ meets } Agreement(A\text{-}role)(B)^n.$$

The following lemma shows that weak agreement is indeed a strengthening of aliveness.

Lemma 4.5. If

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^n,$$

then

$$SYSTEM \text{ meets } Aliveness(A\text{-}role)(B)^n.$$

4.5 Disagreeing over the protocol being run

All of the above discussion has assumed that we are operating in a system with only a single protocol. However, the above techniques are easily extended to cover the more general case.

We may augment the $signal$ events with an extra field representing the identity of the protocol that an agent thinks he's running, and can then adapt the authentication specifications appropriately. For example, the following specification is an adaptation of the $Aliveness$ specification, which guarantees that when one agent completes a run of protocol $protId$, then the other agent was running the same protocol:

$$\begin{array}{l} AlivenessSameProtocol(protId, A\text{-}role)(B) \mathrel{\widehat{=}} \\ \quad signal.Running.protId?B\text{-}role!B?C?ds' \rightarrow \\ \quad RUN(\{signal.Commit.protId.A\text{-}role.A.B.ds \mid \\ \qquad\qquad ds \in Data^* \wedge A \in Agent\}) \end{array}$$

Similarly, the following specification allows the agents to be running different protocols:

$$\begin{array}{l} AlivenessSameProtocol(protId, A\text{-}role)(B) \mathrel{\widehat{=}} \\ \quad signal.Running?protId'?B\text{-}role!B?C?ds' \rightarrow \\ \quad RUN(\{signal.Commit.protId.A\text{-}role.A.B.ds \mid \\ \qquad\qquad ds \in Data^* \wedge A \in Agent\}). \end{array}$$

5 Formalizing authentication specifications with recentness

As described in the introduction, the specifications from the previous section do not insist that the runs are in any way contemporaneous: one agent may commit even though the other has not performed any events *recently*. In this section we strengthen the specifications to insist that the runs are contemporaneous. There are a number of ways of achieving this: we outline two possible approaches in the following two subsections; two other possible approaches are described in [12].

Throughout this section we will make an implementation assumption that no run lasts for too long: if a run lasts for longer than some allowable maximum, then the agent(s) in question should time out and abort the run. This will mean that any two events within the duration of a particular run should be considered to be recent to one another.

5.1 Recentness through freshness

In some cases, the recentness of an agent's run can be guaranteed by agreement on a fresh data value. For example, consider the example of the Needham-Schroeder Public Key Protocol, and suppose we have shown that:

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad Agreement \\ \qquad (RESP\text{-}role, INIT\text{-}role, \{na, nb\})(Bob)^n, \end{array}$$

for some agent Bob. That is: if any initiator A completes a run of the protocol, apparently with Bob, using particular values for the nonces, then A can be sure that at some time in the past, Bob believed that he was acting as responder in a run of the protocol with A, using the same values for the nonces. However, it is assumed that A invented the value of na as a fresh value for this particular run. Hence Bob's run must have occurred at some time after A invented this nonce. Thus, from the implementation assumption about A's runs not lasting too long, Bob must have performed this run *recently*.

This style of argument can be used with both the $Agreement$ and $NonInjectiveAgreement$ specifications, and will guarantee recentness whenever the agents agree on some data value that is freshly invented by the agent to whom the authentication is made. However, when there is no agreement upon a fresh variable, an alternative approach has to be found.

5.2 Timed authentication

We now consider an alternative method of ensuring recentness, namely through introducing a representation of time into the CSP model of the system. We represent the passage of one unit of time by an event $tock$. We then interpret "recently" to mean within the last $AuthTime$ time units, where $AuthTime$ is a parameter provided by the protocol tester, called the *authentication time*. Below, we will check whether the protocol correctly achieves recent authentication by performing a test of the form: whenever A

commits to a session, apparently with B, then B was running the protocol within the last $AuthTime$ *tock*s (possibly with other conditions, corresponding to the different forms of authentication).

In order for the protocol to meet such a specification, we will assume that each agent will timeout if a particular run lasts for longer than some time $MaxRunTime$. (Clearly if such an implementation assumption is not met, we have little chance of meeting a timed specification.) Below, we will define a function that takes an untimed definition of an agent, and gives a timed version, with such a timeout.

First, we define two subsidiary processes: $TOCKS(n)$ will perform at most n *tock*s before terminating; $TSKIP$ will perform an arbitrary number of *tock*s before terminating:

$$
\begin{aligned}
&TOCKS(n) \mathrel{\widehat{=}} \\
&\quad \text{if } n = 0 \text{ then } SKIP \\
&\quad \text{else } (tock \to TOCKS(n-1) \mathbin{\Box} SKIP), \\
&TSKIP \mathrel{\widehat{=}} tock \to TSKIP \mathbin{\Box} SKIP.
\end{aligned}
$$

We now define a function $AddTime$ that turns an untimed representation P of an agent into a timed one. The timed process initially allows an arbitrary amount of time to pass before a run begins. Once the run has begun, at most $MaxRunTime$ *tock*s should occur during the run itself: if an extra $tock$ occurs, then the agent should timeout, and just allow time to pass before terminating. This may be defined as follows:[1]

$$
\begin{aligned}
&AddTime(P, MaxRunTime) \mathrel{\widehat{=}} \\
&\quad tock \to AddTime(P, MaxRunTime) \\
&\quad \Box \\
&\quad ((P \mathbin{|||} TOCKS(MaxRunTime)) \\
&\qquad \triangle\ tock \to TSKIP).
\end{aligned}
$$

(In fact, the above definition allows the agent to nondeterministically abort the run after fewer than $MaxRunTime$ time units; this corresponds to the agent aborting for some reason we are not modelling, such as user intervention. However, if more than $MaxRunTime$ *tock*s occur, then one of the *tock*s must trigger the timeout. Note, though, that the trace set of the process is not affected by this possibility of early timeout.)

We apply the $AddTime$ function to all the processes representing untimed agents, so as to build timed versions; we then combine these timed agents together to build a timed system.

We are now ready to produce the timed authentication specifications. We use the $AddTime$ function to lift the untimed agreement specifications to timed versions. For example:

$$
\begin{aligned}
&TimedAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B) \\
&\mathrel{\widehat{=}} \\
&AddTime(Agreement(B\text{-}role, A\text{-}role, ds)(B), \\
&\qquad\qquad AuthTime).
\end{aligned}
$$

The above process allows at most $AuthTime$ *tock*s during the execution of $Agreement(B\text{-}role, A\text{-}role, ds)(B)$, i.e., between the $signal.Running$ and $signal.Commit$ events; thus it specifies that the $Commit$ signal must occur within $AuthTime$ *tock*s of the $Running$ signal.

We can test whether a system correctly achieves timed agreement by testing whether the appropriate abstraction of the system (built from timed agents) refines n copies of the above specification:

$$
\begin{aligned}
&SYSTEM \text{ meets} \\
&\quad TimedAgreement \\
&\qquad (B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n.
\end{aligned}
$$

But now we must redefine the $(\_)^n$ notation so as to synchronize all the individual specifications on $tock$ events:

$$P^n \mathrel{\widehat{=}} \underbrace{P \mathbin{\underset{\{tock\}}{\|}} \ldots \mathbin{\underset{\{tock\}}{\|}} P}_{n}\ .$$

Formally: $P^0 \mathrel{\widehat{=}} RUN(\{tock\})$ and $P^{n+1} \mathrel{\widehat{=}} P \mathbin{\underset{\{tock\}}{\|}} P^n$.

The other untimed authentication specifications can similarly be lifted to timed specifications by using the $AddTime$ function:

$$
\begin{aligned}
&TimedNonInjectiveAgreement \\
&\quad (B\text{-}role, A\text{-}role, ds, AuthTime)(B) \\
&\mathrel{\widehat{=}} \\
&AddTime(\\
&\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B), \\
&\quad AuthTime), \\
&TimedWeakAgreement(A\text{-}role, AuthTime)(B) \\
&\mathrel{\widehat{=}} \\
&AddTime(WeakAgreement(A\text{-}role)(B), AuthTime), \\
&TimedAliveness(A\text{-}role, AuthTime)(B) \\
&\mathrel{\widehat{=}} \\
&AddTime(Aliveness(A\text{-}role)(B), AuthTime).
\end{aligned}
$$

The following lemma is analogous to the lemmas of Section 4.

Lemma 5.1.

- If $ds' \subseteq ds$ and

$$
\begin{aligned}
&SYSTEM \text{ meets} \\
&\quad TimedAgreement \\
&\qquad (B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n
\end{aligned}
$$

[1] The process $Q \triangle R$ represents an interrupt mechanism; it initially acts like Q, but may be interrupted by the performance of any event of R, and will then continue to act like R.

then

$$SYSTEM \text{ meets } TimedAgreement(B\text{-}role, A\text{-}role, ds', AuthTime)(B)^n.$$

- If $ds' \subseteq ds$ and

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds', AuthTime)(B)^n$$

- If

$$SYSTEM \text{ meets } TimedAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

- Suppose the agent B can perform a maximum of m runs with role $B\text{-}role$, and n runs with other roles. If

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^m$$

then

$$SYSTEM \text{ meets } TimedWeakAgreement(A\text{-}role, AuthTime)(B)^{m+n}$$

- If

$$SYSTEM \text{ meets } TimedWeakAgreement(A\text{-}role, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedAliveness(A\text{-}role, AuthTime)(B)^n.$$

The timed authentication specifications are antimonotonic in the time parameter.

Lemma 5.2. Let $TimedSpec(\_)$ be one of the timed authentication specifications, with parameter representing the time (for example, $TimedSpec(t) = TimedAliveness(A\text{-}role, t)(B)$), and let $t \leq t'$. Then if

$$SYSTEM \text{ meets } TimedSpec(t)^n$$

then

$$SYSTEM \text{ meets } TimedSpec(t')^n.$$

Finally, the timed specifications imply the corresponding untimed specifications.

Lemma 5.3. Let $Spec$ be one of the untimed authentication specifications, and let $TimedSpec$ be the corresponding timed specification, i.e., $TimedSpec = AddTime(Spec, AuthTime)$. If $SYSTEM$ meets $TimedSpec$ then $SYSTEM$ meets $Spec$.

6 Conclusions

In this paper we have considered the question of the meaning of the term "entity authentication". We argued that different authentication requirements may be appropriate for different circumstances, and identified a number of possible authentication specifications.

We formalized each authentication specification using the process algebra CSP, used this formalism to study their relative strengths, and showed how to use the model checker FDR to test whether a system running a protocol achieves the various authentication requirements.

Our authentication specifications have much in common with Schneider's work [22, 23]. He uses a trace specification T authenticates R to mean that events from T authenticate events from R: events from T can happen only if there have been preceding events from R:

$$T \text{ authenticates } R \;\widehat{=}\; tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle.$$

Note that the above test does not capture our notion of injectivity: if an event from R occurs, then there may be several subsequent events from T. Nor does it capture our notion of recentness: the events from T may occur some time after the corresponding events from R. However, it is not hard to adapt the specification to as to capture both injectivity and recentness.

The $NonInjectiveAgreement$, $WeakAgreement$ and $Aliveness$ tests of this paper may be seen as testing whether a $Commit$ signal authenticates a corresponding

Running signal. This is made formal as follows:

$$
\begin{array}{l}
SYSTEM \text{ meets} \\
\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B) \\
iff \\
\forall A \in Agent \,;\, ds' \in Data^* \bullet \\
\quad SYSTEM \text{ sat } \{signal.Commit.A\text{-}role.A.B.ds'\} \\
\qquad\qquad\qquad \text{authenticates} \\
\qquad\qquad\qquad \{signal.Running.B\text{-}role.B.A.ds'\},
\end{array}
$$

$$
\begin{array}{l}
SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B) \\
iff \\
\forall A \in Agent \bullet \\
\quad SYSTEM \text{ sat} \\
\qquad \{signal.Commit.A\text{-}role.A.B.ds \mid ds \in Data^*\} \\
\qquad \text{authenticates} \\
\qquad \{signal.Running.B\text{-}role.B.A.ds' \mid \\
\qquad\quad B\text{-}role \in ROLE \wedge ds' \in Data^*\},
\end{array}
$$

$$
\begin{array}{l}
SYSTEM \text{ meets } Aliveness(A\text{-}role)(B) \\
iff \\
SYSTEM \text{ sat} \\
\quad \{signal.Commit.A\text{-}role.A.B.ds \mid \\
\qquad ds \in Data^* \wedge A \in Agent\} \\
\quad \text{authenticates} \\
\quad \{signal.Running.B\text{-}role.B.C.ds' \mid \\
\qquad B\text{-}role \in ROLE \wedge \\
\qquad ds' \in Data^* \wedge C \in Agent\}.
\end{array}
$$

Casper is a program that takes an abstract description of a security protocol, and produces a corresponding CSP description, suitable for checking using FDR. Each of the authentication specifications considered in this paper have been implemented within Casper. Thus Casper and FDR can be used together to test which, if any, authentication specifications are met by a particular protocol. A number of case studies using these techniques are available via the Casper World Wide Web page [14]; these case studies demonstrate the practicality of our methods.

When analyzing protocols using FDR, we make a number of implementation assumptions, which are reflected in the way we model the intruder and the honest agents. For example: above we made the assumption that honest agents would time out if runs are taking too long; we normally assume that the intruder is unable to decrypt messages unless he has the appropriate key; and we normally assume that the encryption method used has no "interesting" algebraic properties. However, none of these assumptions are really necessary: by altering the way we model the system, we can remove these assumptions. Note, though, that the authentication specifications discussed in this paper are independent of the implementation assumptions, and so would still be applicable under different assumptions.

Acknowledgements

I would like to thank Steve Schneider, Bill Roscoe and Dieter Gollmann for interesting discussions and useful comments on the work in this paper.

References

[1] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, 1990.

[2] R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Mulva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, 1993.

[3] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[4] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[5] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[6] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2—User Manual*, 1997. Available via URL `ftp://ftp.comlab.ox.ac.uk/pub/Packages/FDR`.

[7] D. Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Research in Security and Privacy*, 1996.

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

[9] T. Hwang and Y.-H. Chen. On the security of SPLICE/AS—the authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[10] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[11] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[12] G. Lowe. A hierarchy of authentication specifications. Technical Report 1996/33, Department of Mathematics and Computer Science, University of Leicester, 1996.

[13] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[14] G. Lowe. Casper: A compiler for the analysis of security protocols, 1997. In this volume. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`.

[15] S. P. Miller, C. Neumann, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, MIT, 1987. Available from URL `ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS`.

[16] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[17] L. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, University of Cambridge Computer Laboratory, 1996.

[18] L. Paulson. Mechanized Proofs of Security Protocols: Needham-Schroeder with Public Keys. Technical Report 413, University of Cambridge Computer Laboratory, 1997.

[19] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[20] A. W. Roscoe. Intensional specification of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.

[21] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[22] S. Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Security and Privacy*, Oakland, 1996.

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

[24] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136, 1994.

[25] S. Yamaguchi, K. Okayama, and H. Miyahara. Design and implementation of an authentication system in WIDE Internet environment. In *Proc. 10th IEEE Region Conf. on Computer and Communication Systems*, 1990.