A Hierarchy of Authentication Specifications

Gavin Lowe

Department of Mathematics and Computer Science,
University of Leicester

e-mail: `gavin.lowe@mcs.le.ac.uk`
URL: `http://www.mcs.le.ac.uk/~gl7/`

November 1996

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 study several possible definitions of the word. We formalize each definition using the process algebra CSP, and use this formalism to study their relative strengths. We also show how the model checker FDR can be used to test whether a system running the protocol meets these authentication specifications.

1 Introduction

Many 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 [Hoa85].

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". 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.

Since there are so many different meanings of the word "authentication", we will avoid using this word, other than in a very general way; we will introduce different terms corresponding to some of the possible meanings considered above. We formalize each of these meanings using the process algebra CSP [Hoa85]. 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 [Ros94, For96].

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: here are a number of attacks upon protocols that lead to an agent A thinking he has established a key with another agent B, but in fact he has been running the protocol with an intruder imitating B, and the intruder now knows 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 which 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.

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 achieve authentication between a pair of agents, an *initiator* and a *responder*, possibly with the help of a third party, a *server*. We use the word "rôle" 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 extend all these ideas to include extra agents playing additional rôles.

We do not restrict ourselves to the case where a particular agent may only ever adopt a single rôle; 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 (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.).

In each of the following definitions, we consider the case where a responder B is authenticated to an initiator A; it should be obvious how to generalize this to the opposite case. 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; for example: a number of attacks described by Bird et al. in [BGH$^+$93]; a variant on the attack on the Wide Mouthed Frog protocol [BAN89] due to Anderson and Needham [AN95]; the attack on the Neuman-Stubblebine protocol [NS93] due to Hwang et al. [HLL$^+$95]; and my attack upon the Andrew protocol [Sat89, BAN89] in [Low96c]. 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 agent as an oracle; for example: the attack on the BAN version of the Yahalom protocol [BAN89] described by Syverson in [Syv94]; and my attack on the Woo and Lam mutual authentication protocol [WL94] in [Low96c]. Other attacks are due to more blatant errors; for example, Hwang and Chen's attack on the SPLICE protocol [YOM90] in [HC95], 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. This is an interesting question, but one we choose not to consider in this paper, for it applies to a set of protocols, while we want to restrict our attention to a single protocol.

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: the attack on the CCITT X.509 protocol [CCI87] reported in [BAN89]; my attack on the Needham-Schroeder Public Key protocol [NS78] in [Low95]; Clark and Jacob's attack on Hwang and Chen's version of the SPLICE protocol [HC95] in [CJ95]; and my attack on the Station-to-Station protocol [DvOW92] in [Low96c].

2.3 Non-injective agreement

The following definition adds the condition that the two agents agree as to which rôles 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.

Clearly, a protocol cannot guarantee agreement between A and B on a particular data item d unless the protocol is such that both agents really do see d in a normal run. Further, A cannot be assured of agreement with B on d in protocols where A creates d and sends it to B (either directly or via a third party), but never receives any form of acknowledgement back from B (either directly or via a third party): A cannot be sure that the message to B ever got through. Cases such as these can be found by a static analysis of the protocol, without considering the effect of an intruder. We will normally consider the more interesting cases where there appears to be some chance of agreement being established, and ask whether this is still the case in the presence of an intruder.

A few protocols achieve a guarantee of weak agreement, but not non-injective agreement. For example, the Canadian attack on an early version of a protocol from ISO/IEC 9798-3 [ISO91a, ISO91b] involved an intruder tricking an agent B into thinking that he had completed a run with A, with A as initiator, whereas A thought he was the responder in his run. Similarly, in my attack [Low96c] on the KSL protocol [KSL92], the intruder instigates a pair of runs, one each with A and B, where each thought they were running the protocol with the other, but where both thought they were the responder. Finally, if the original Andrew protocol [Sat89] is adapted to detect mirror attacks, as described above, then it achieves weak agreement, but does not achieve non-injective agreement on all the data values: an attack in [BAN89] 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 is clearly of importance 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 [MNSS87], 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 [BM90] that early implementations did not perform this check. Similar attacks are possible against the X.509 [CCI87] and Wide Mouthed Frog [BAN89] protocols.

Also, in the Otway-Rees protocol [OR87], the responder receives no guarantee of freshness from the initiator, merely a guarantee that the initiator has been running the protocol with this responder at some time in the past [BAN89]; hence, an intruder could replay the appropriate messages to complete multiple runs.

2.5 Recentness

Finally, we lift the above definitions to ensure recentness. 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.

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, which is a strengthening of Key Establishment Mechanism 2 of [ISO94]:

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

Our notation is very standard: $\{m\}_k$ represents message m encrypted with key k; kab is a key shared between A and B. 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: this gives Key Establishment Mechanism 4 of [ISO94].

Also, in a setting where key compromise is possible, there is a well known attack on the Needham-Schroeder Secret Key Protocol [NS78], presented in [DS81]: 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.

Finally, the attack on the Otway-Rees protocol mentioned above can be launched a long time after the original run, which shows that this protocol fails to achieve recent aliveness.

2.6 Discussion

All of the above definitions used a phrase of the form "B has previously been running the protocol". We make clear here what we mean by "running the protocol". We specify that the other agent A should be assured that B has progressed a reasonable way through the protocol. In the case where the last message of the protocol is sent from A to B, A can never be sure that B has completed the protocol run, for the last message may not have got through. However, in a correctly functioning protocol, A should at least be assured that B sent the last message that he was supposed to send. Hence, in all the above definitions, when A completes a run, B should have at least sent this last message. 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 [Ros96] 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. To see that it is at least as strong as full agreement, 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 rôles 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.

To see that the intensional specification is strictly stronger than full agreement, consider the original Yahalom protocol [BAN89]:

$$\begin{array}{lrcl} \text{Message 1.} & A \rightarrow B & : & A, na \\ \text{Message 2.} & B \rightarrow S & : & B, \{A, na, nb\}_{kbs} \\ \text{Message 3.} & S \rightarrow A & : & \{B, kab, na, nb\}_{kas}, \{A, kab\}_{kbs} \\ \text{Message 4.} & A \rightarrow B & : & \{A, kab\}_{kbs}, \{nb\}_{kab} \end{array}$$

where kas is a key shared between A and the server S; kbs is a key shared between B and S; kab is intended as a session key to be shared between A and B; na and nb are nonces invented by A and B respectively. Note that A cannot decrypt the second component of message 3, because it is encrypted with a key shared between B and S; he merely forwards this component to B in message 4. There is a simple attack where the intruder replaces this component in message 3 with an arbitrary message, but when A tries forwarding this in message 4, the intruder replaces it with the original component. This attack shows that the protocol does not meet the intensional specification, but the protocol does seem to achieve full agreement (at least, I know of no attack that suggests it doesn't).

The intensional specification also insists that all messages occur in the order given in the protocol definition. Consider a protocol where the 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 [DvOW92] 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 [Gol96] 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:

> **G3.** A cryptographic key associated with B was used during the protocol run. It is not necessary for A to receive a message where this key had been used. The protocol run is defined by A's challenge or a current time stamp.

(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:

> **G4.** 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.

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 [Low96a]. 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 [NS78, Low95]. 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:

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

$RESPONDER_0(B, nb, skb) \mathrel{\widehat=}$
$\quad \square_{A:Agent;na:Nonce}\, comm.A.B.(Msg1, Encrypt.(inverse(skb), \langle na, A\rangle)) \to$
$\quad signal.Running.RESP\text{-}r\hat{o}le.B.A.na.nb \to$
$\quad comm.B.A.(Msg2, Encrypt.(PK(A), \langle na, nb\rangle)) \to$
$\quad comm.A.B.(Msg3, Encrypt.(inverse(skb), \langle nb\rangle)) \to$
$\quad signal.Commit.RESP\text{-}r\hat{o}le.B.A.na.nb \to$
$\quad SKIP$

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 $signal.Running.r\hat{o}le.A.B.na.nb$ represents that the agent A believes that he is taking the rôle *rôle* 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 $signal.Commit.r\hat{o}le.A.B.na.nb$ represents that the agent A thinks he has successfully completed a run of the protocol, taking rôle *rôle*, 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 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; 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 timing issues. All the specifications are of the form: if A completes a run of the protocol, apparently with B, then B has previously been running the protocol (possibly along with other conditions relating A's and B's runs); however these specifications do not insist that B's run was recent. In the next section we adapt these specifications to include recentness.

We begin by introducing a few pieces of notation that we will need. If P is a process, we define αP to be its alphabet.

We write $P \upharpoonright S$, where S is a set of events, for the abstraction of P to the events in S; this may be defined using the familiar hiding operator:

$$P \upharpoonright S \;\widehat{=}\; P \setminus (\Sigma - S),$$

where Σ is the set of all events.

We write P^n for the interleaving of n copies of P:

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

Formally:

$$\begin{aligned} P^0 &\;\widehat{=}\; STOP \\ P^{n+1} &\;\widehat{=}\; P \mid\mid\mid P^n. \end{aligned}$$

Note that 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 rôles $A\text{-}r\hat{o}le$ and $B\text{-}r\hat{o}le$, 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 rôle $A\text{-}r\hat{o}le$, agreement with an agent B, taking the rôle $B\text{-}r\hat{o}le$, 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 rôle $B\text{-}r\hat{o}le$, 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 $signal.Running.B\text{-}r\hat{o}le.B.A.d_1'.d_2'$; similarly, A completing the protocol, taking the rôle $A\text{-}r\hat{o}le$, apparently with B, using the same values for the data items, is represented by the event $signal.Commit.A\text{-}r\hat{o}le.A.B.d_1'.d_2'$. Thus we will want to ensure that the abstraction of the system to the appropriate alphabet satisfies the following specification:

$$\begin{aligned}
&\mathit{Agreement}(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1, d_2\})(B) \mathrel{\widehat{=}} \\
&\quad \mathit{signal.Running.B\text{-}r\hat{o}le.B?A?d_1'?d_2'} \to \\
&\quad \mathit{signal.Commit.A\text{-}r\hat{o}le.A.B.d_1'.d_2'} \to \mathit{STOP}\ .
\end{aligned}$$

This specification says that whenever A performs an event of the form $\mathit{signal.Commit.A\text{-}r\hat{o}le.A.B.d_1'.d_2'}$, then B has previously performed a corresponding event $\mathit{signal.Running.B\text{-}r\hat{o}le.B.A.d_1'.d_2'}$. That is, whenever A signals that he thinks he has completed the protocol, taking the rôle $A\text{-}r\hat{o}le$, with B, using the data values d_1' and d_2', then B has previously been running the protocol, taking the rôle $B\text{-}r\hat{o}le$, 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; indeed, it is easy to see that the intruder can launch a lack of service attack—preventing A from ever committing—by simply intercepting all messages sent to A.) Figure 1 illustrates the meanings of the arguments.

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

$$\mathit{Agreement}(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)$$

Figure 1: The arguments of *Agreement*

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 rôle $B\text{-}r\hat{o}le$, to an arbitrary agent A, taking rôle $A\text{-}r\hat{o}le$, with agreement on all the data items, by testing:

$$\begin{aligned}
&\mathit{Agreement}(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1, d_2\})(B) \sqsubseteq \\
&\quad \mathit{SYSTEM} \upharpoonright \alpha \mathit{Agreement}(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1, d_2\})(B).
\end{aligned}$$

(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 rôle $A\text{-}r\hat{o}le$. The reason why we test whether a *single* agent B is correctly authenticated, rather than testing whether all agents who take the rôle $B\text{-}r\hat{o}le$ are correctly authenticated, is that, in practice, the latter check would become very slow and inefficient when we consider the case where each agent may perform several runs, below. 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.

$$SYSTEM \text{ meets } SPEC \;\widehat{=}\; SPEC \sqsubseteq SYSTEM \upharpoonright \alpha SPEC.$$

So the above test is written:

$$SYSTEM \;\text{ meets }\; Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1, d_2\})(B).$$

For example, the Needham-Schroeder Public Key Protocol guarantees to an arbitrary initiator A agreement with the responder Bob on the nonces na and nb, (that is, Bob is authenticated to an arbitrary initiator) if:

$$SYSTEM \;\text{ meets }\; Agreement(RESP\text{-}r\hat{o}le, INIT\text{-}r\hat{o}le, \{na, nb\})(Bob).$$

Similarly, the protocol guarantees to an arbitrary responder B agreement with the initiator $Alice$ on the nonces na and nb, if:

$$SYSTEM \;\text{ meets }\; Agreement(INIT\text{-}r\hat{o}le, RESP\text{-}r\hat{o}le, \{na, nb\})(Alice).$$

It turns out that the first of these specifications is met (the protocol correctly authenticates the responder), but the second is not (the protocol does not correctly authenticate the initiator), as can be seen from the attack in [Low95, Low96a].

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:

$$SYSTEM \;\text{ meets }\; Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1, d_2\})(B)^n.$$

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 test:

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

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

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

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 $signal.Running.B\text{-}rôle.B.A.ds'$ 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. Suppose a protocol guarantees to an agent A taking rôle $A\text{-}rôle$ agreement with another agent B taking rôle $B\text{-}rôle$ on a set of data values ds_1:

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

Suppose further that

$$ds_2 \subseteq ds_1.$$

Then the protocol guarantees to A taking rôle $A\text{-}rôle$ agreement with B taking rôle $B\text{-}rôle$ on ds_2:

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

Proof: We have:

$$\begin{array}{ll} & SYSTEM \upharpoonright \alpha Agreement(B\text{-}rôle, A\text{-}rôle, ds_2)(B) \\ \sqsupseteq & \left\langle \begin{array}{l} \alpha Agreement(B\text{-}rôle, A\text{-}rôle, ds_1)(B) \\ \quad = \alpha Agreement(B\text{-}rôle, A\text{-}rôle, ds_2)(B) \end{array} \right\rangle \\ & SYSTEM \upharpoonright \alpha Agreement(B\text{-}rôle, A\text{-}rôle, ds_1)(B) \\ \sqsupseteq & \langle \text{hypothesis} \rangle \\ & Agreement(B\text{-}rôle, A\text{-}rôle, ds_1)(B)^n \\ \sqsupseteq & \langle \text{definition of } Agreement\text{; monotonicity of interleaving} \rangle \\ & Agreement(B\text{-}rôle, A\text{-}rôle, ds_2)(B)^n, \end{array}$$

as required. □

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:

$$\begin{array}{l} NonInjectiveAgreement(B\text{-}rôle, A\text{-}rôle, \{d_1, d_2\})(B) \mathrel{\widehat{=}} \\ \quad signal.Running.B\text{-}rôle.B?A?d_1'?d_2' \rightarrow \\ \quad RUN(\{signal.Commit.A\text{-}rôle.A.B.d_1'.d_2'\}). \end{array}$$

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

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

$$\begin{array}{l} NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, \{d_1\})(B) \mathrel{\widehat{=}} \\ \quad signal.Running.B\text{-}r\hat{o}le.B?A?d_1'?d_2'' \to \\ \quad RUN(\{signal.Commit.A\text{-}r\hat{o}le.A.B.d_1'.d_2' \mid d_2' \in Data\}). \end{array}$$

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

$$SYSTEM \ \mathsf{meets} \ NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^n.$$

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:

$$SYSTEM \ \mathsf{meets} \ NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds_1)(B)^n,$$

then:

$$SYSTEM \ \mathsf{meets} \ NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds_2)(B)^n.$$

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

Lemma 4.3. Suppose a protocol guarantees to an agent A taking rôle $A\text{-}r\hat{o}le$ agreement with another agent B taking rôle $B\text{-}r\hat{o}le$ on a set of data values ds:

$$SYSTEM \ \mathsf{meets} \ Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^n.$$

Then the protocol guarantees non-injective agreement (for the same agents, rôles, and data):

$$SYSTEM \ \mathsf{meets} \ NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^n.$$

Proof: This follows from the fact that $\alpha Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^n = \alpha NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^n$, and

$$\begin{array}{l} Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B) \sqsupseteq \\ \quad NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B). \end{array}$$

□

4.3 Weak agreement

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

$$
\begin{array}{l}
WeakAgreement(A\text{-}r\hat{o}le)(B) \mathrel{\widehat{=}} \\
\quad signal.Running?B\text{-}r\hat{o}le!B?A?ds' \to \\
\quad RUN(\{signal.Commit.A\text{-}r\hat{o}le.A.B.ds \mid ds \in Data^*\}).
\end{array}
$$

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

Weak agreement can then be checked using the refinement test:

$$SYSTEM \ \mathsf{meets}\ WeakAgreement(A\text{-}r\hat{o}le)(B)^n.$$

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 rôle $B\text{-}r\hat{o}le$, and n runs with other rôles. If:

$$SYSTEM \ \mathsf{meets}\ NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^m,$$

then:

$$SYSTEM \ \mathsf{meets}\ WeakAgreement(A\text{-}r\hat{o}le)(B)^{m+n}.$$

Proof: Assume the hypotheses. We have:

$$
\begin{array}{ll}
 & SYSTEM \restriction \alpha WeakAgreement(A\text{-}r\hat{o}le)(B) \\
= & \left\langle \text{definitions of alphabets} \right\rangle \\
 & SYSTEM \restriction (\alpha NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B) \cup \\
 & \qquad \{\!|\, signal.Running.B\text{-}r\hat{o}le'.B \mid B\text{-}r\hat{o}le' \in R\hat{O}LE - \{B\text{-}r\hat{o}le\} \,|\!\}) \\
\sqsupseteq & \left\langle \text{using the law } P \restriction (A \cup B) \sqsupseteq (P \restriction A) \mathbin{|||} (P \restriction B) \text{ if } A \cap B = \{\} \right\rangle \\
 & SYSTEM \restriction \alpha NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B) \\
 & \mathbin{|||} SYSTEM \restriction \{\!|\, signal.Running.B\text{-}r\hat{o}le'.B \mid B\text{-}r\hat{o}le' \in R\hat{O}LE - \{B\text{-}r\hat{o}le\} \,|\!\} \\
\sqsupseteq & \left\langle \text{hypotheses} \right\rangle \\
 & NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)^m \\
 & \mathbin{|||} (signal.Running?B\text{-}r\hat{o}le'!B?A?ds' \to STOP)^n \\
\sqsupseteq & \left\langle \text{definitions} \right\rangle \\
 & WeakAgreement(A\text{-}r\hat{o}le)(B)^m \\
 & \mathbin{|||} WeakAgreement(A\text{-}r\hat{o}le)(B)^n \\
= & \left\langle \text{definition of } (\_)^n \right\rangle \\
 & WeakAgreement(A\text{-}r\hat{o}le)(B)^{m+n}.
\end{array}
$$

$\square$

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:

$$\begin{array}{l} Aliveness(A\text{-}r\hat{o}le)(B) \mathrel{\widehat{=}} \\ \qquad signal.Running?B\text{-}r\hat{o}le!B?C?ds' \rightarrow \\ \qquad RUN(\{signal.Commit.A\text{-}r\hat{o}le.A.B.ds \mid ds \in Data^* \wedge A \in Agent\}) \end{array}$$

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

Agreement can then be checked using the refinement test:

$$SYSTEM \ \textsf{meets} \ Agreement(A\text{-}r\hat{o}le)(B)^n.$$

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

Lemma 4.5. If:

$$SYSTEM \ \textsf{meets} \ WeakAgreement(A\text{-}r\hat{o}le)(B)^n,$$

then:

$$SYSTEM \ \textsf{meets} \ Aliveness(A\text{-}r\hat{o}le)(B)^n.$$

Proof: This follows from the fact that

$$\alpha WeakAgreement(A\text{-}r\hat{o}le)(B)^n = \alpha Aliveness(A\text{-}r\hat{o}le)(B)^n$$

and

$$WeakAgreement(A\text{-}r\hat{o}le)(B) \ \sqsupseteq \ Aliveness(A\text{-}r\hat{o}le)(B).$$

□

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 some of the alternatives in the next three subsections, before describing our preferred approach in Section 5.4.

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 time span 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:

$$SYSTEM \text{ meets } Agreement(RESP\text{-}r\hat{o}le, INIT\text{-}r\hat{o}le, \{na, nb\})(Bob)^n,$$

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 Recentness through overlapping runs

One possibility would be to insist that, in the case that B is being authenticated to A, B's run must overlap with A's. If this specification were met, then A could be assured that B has been running the protocol at some time since A began his protocol run.

In order to test such a requirement, it is necessary to alter the definitions of the agents so as to include an extra signal event, marking the point at which the agent begins the run. It is also necessary to include an extra field in the signals, which we term a *run identifier* (ranged over by α, β, etc.); the purpose of this field is to identify which $Begin$ signals correspond to which $Commit$ signals.

The $Agreement$ specification would then be altered to insist that B's $Running$ signal occurs between A beginning the run and committing:

$$\begin{array}{l} Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B) \mathrel{\widehat{=}} \\ \quad signal.Begin.A\text{-}r\hat{o}le?A?B?\alpha \rightarrow \\ \quad signal.Running.B\text{-}r\hat{o}le.B.A?ds' \rightarrow \\ \quad signal.Commit.A\text{-}r\hat{o}le.A.B.\alpha.ds' \rightarrow STOP\ . \end{array}$$

The above approach works fine for a protocol consisting of an initiator and a responder without a server. However, on closer inspection, this is not enough in more general settings. Consider a one-way authentication protocol that begins with B authenticating himself to a trusted server S, followed by S authenticating himself to A, authenticating B's identity. Here B's run finishes at the end of the first stage, which is before A's run begins. The two runs should still be considered recent, though, because they are spanned by the server's run.

It is possible to adapt the approach to deal with this situation, so as to insist that either: (1) A's and B's runs overlap, as above; or (2) the server's run spans A's and B's runs.

However, even this isn't enough in a system with several servers. Consider a protocol with n servers, $S_1, \ldots, S_n$, where: in the first phase, the initiator carries out a handshake with S_1; in the second phase, S_1 carries out a handshake with S_2; and so on until in the final phase, S_n carries out a handshake with the responder. Then no single server's run spans both the initiator's and responder's run. However, these runs should still be considered contemporaneous, because between them the servers' runs span the period.

Again, it is possible to adapt the specification to deal with such a situation, but it is sufficiently complicated to be unattractive. We therefore seek an alternative approach.

5.3 Recentness through timestamping beliefs

We now consider an alternative method of ensuring recentness, namely through timestamping the signal events, writing, for example,

$$signal.t.Running.B\text{-}r\hat{o}le.B.A.ds$$

to represent that at time t the agent B taking the rôle B believes that he is running the protocol with A using the data values ds. 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*. We can then 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 $AuthTime$ time units (possibly with other conditions, corresponding to the different forms of authentication).

In order to timestamp the signals, we assume the existence of a reliable, authenticated timeserver. This can be modelled by a single shared clock. This clock should be willing to synchronize on any signal event with the current timestamp; the other agents should be willing to perform appropriate signals with any timestamp; by synchronizing on these signal events, we can ensure that the signals are correctly timestamped.

The clock should also be willing to perform an event *tock*, representing the passage of one unit of time, and increase its record of the time. However, in order to keep the state space of the system finite (to allow model checking), we need to restrict the time domain to some finite set, which we take to be $\{0 \mathrel{..} MaxTime\}$ for some value $MaxTime$.

This clock can be represented by the CSP process $CLOCK(0)$, where:

$$\begin{aligned} CLOCK(t) \;\widehat{=}\; & \text{if } t > MaxTime \text{ then } STOP \\ & \text{else } (signal!t?sig \rightarrow CLOCK(t) \;\Box\; tock \rightarrow CLOCK(t+1)) \,. \end{aligned}$$

We may now specify agreement as follows:

$$
\begin{array}{l}
\mathit{TimedAgreement}(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, \mathit{AuthTime})(B) \mathrel{\widehat{=}} \\
\qquad signal?t_b!Running.B\text{-}r\hat{o}le.B?A?ds' \rightarrow \\
\qquad signal?t_a\!:\!\{t_b \mathinner{..} t_b + \mathit{AuthTime}\}!Commit.A\text{-}r\hat{o}le.A.B.ds' \rightarrow STOP\ .
\end{array}
$$

If A commits to a session at some time t_a, then B should have been running the protocol at some time t_b such that t_a is at most $\mathit{AuthTime}$ time units after t_b.

In order for the protocol to meet this specification, certain implementation assumptions will have to be met. For example, it will normally be the case that each agent should timeout if the current run has been going on for longer than some time $\mathit{MaxRunTime}$. (Clearly if such an implementation assumption is not met, we have little chance of meeting the above specification.)

Below, we will define a function that takes an untimed definition of an agent, as in Section 3, and gives a timed version, which will timeout if a run lasts longer than $\mathit{MaxRunTime}$. First, we define two subsidiary processes: the process $TOCKS(n)$ will perform at most n *tock*s before terminating; the process $TSKIP$ will perform an arbitrary number of *tock*s before terminating:

$$
\begin{array}{rcl}
TOCKS(n) & \mathrel{\widehat{=}} & \text{if } n = 0 \text{ then } SKIP \\
 & & \text{else } (tock \rightarrow TOCKS(n-1) \,\Box\, SKIP), \\
TSKIP & \mathrel{\widehat{=}} & tock \rightarrow TSKIP \,\Box\, SKIP.
\end{array}
$$

We are now able to define a function $\mathit{AddTime}$ that turns an untimed representation P of an agent into a timed one. The timed process initially allow an arbitrary amount of time to pass before a run begins. Once the run has begun, at most $\mathit{MaxRunTime}$ *tock*s should occur during the run itself: if an extra *tock* occurs, then the agent should time out, and stop performing messages from the protocol, but should instead just allow time to pass before terminating. This may be defined as follows:[1]

$$
\begin{array}{l}
\mathit{AddTime}(P, \mathit{MaxRunTime}) \mathrel{\widehat{=}} \\
\qquad tock \rightarrow \mathit{AddTime}(P, \mathit{MaxRunTime}) \\
\qquad \Box \\
\qquad (P \mathbin{|||} TOCKS(\mathit{MaxRunTime})) \mathbin{\triangle} tock \rightarrow TSKIP.
\end{array}
$$

In fact, the above definition allows the agent to nondeterministically abort the run after fewer than $\mathit{MaxRunTime}$ time units; this corresponds to the agent aborting for some reason we are not modelling, such as user intervention. However, if more than $\mathit{MaxRunTime}$ *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 time-out.

5.4 Recentness through counting tocks

The above method works reasonably well, but suffers from a couple of weaknesses when we use these techniques with a model checker such as FDR. Firstly,

[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.

in order to keep the state space finite, it is necessary to consider a finite time domain; making such an assumption risks not discovering certain attacks that require longer to occur. Secondly, the presence of explicit timestamps increases the size of the state space, so increases the time taken to perform a check.

A related but better method is not to explicitly timestamp belief signals, but instead to have the specification count the number of *tock*s between the corresponding signals, and to allow a *Commit* signal only if there have been at most *AuthTime* *tock*s since a corresponding *Running* signal. This specification is simply defined by applying the *AddTime* function of the previous section to the untimed agreement specification:

$$\begin{array}{l} TimedAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B) \mathrel{\widehat{=}} \\ \quad AddTime(Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B),\ AuthTime). \end{array}$$

The above process allows at most *AuthTime* *tock*s during the execution of $Agreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B)$, i.e., between the *signal.Running* and *signal.Commit* events.

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

$$\begin{array}{l} SYSTEM \ \textsf{meets} \\ \quad TimedAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^n. \end{array}$$

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

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

Formally:

$$\begin{array}{rcl} P^0 & \mathrel{\widehat{=}} & RUN(\{tock\}) \\ P^{n+1} & \mathrel{\widehat{=}} & P \underset{\{tock\}}{\|} P^n. \end{array}$$

As in the previous section, it is necessary to ensure that each agent will timeout and abort a run if that run has taken more than some maximum allowed run time.

The other untimed authentication specifications can be lifted to timed specifications by using the *AddTime* function, as above:

$$\begin{array}{l} TimedNonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B) \mathrel{\widehat{=}} \\ \quad AddTime(NonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds)(B),\ AuthTime), \\ TimedWeakAgreement(A\text{-}r\hat{o}le, AuthTime)(B) \mathrel{\widehat{=}} \\ \quad AddTime(WeakAgreement(A\text{-}r\hat{o}le)(B),\ AuthTime), \\ TimedAliveness(A\text{-}r\hat{o}le, AuthTime)(B) \mathrel{\widehat{=}} \\ \quad AddTime(Aliveness(A\text{-}r\hat{o}le)(B),\ AuthTime). \end{array}$$

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

Lemma 5.1.

- If $ds' \subseteq ds$ and $SYSTEM$ meets $TimedAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^n$, then $SYSTEM$ meets $TimedAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds', AuthTime)(B)^n$.

- If $ds' \subseteq ds$ and $SYSTEM$ meets $TimedNonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^n$, then $SYSTEM$ meets $TimedNonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds', AuthTime)(B)^n$.

- If $SYSTEM$ meets $TimedAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^n$, then $SYSTEM$ meets $TimedNonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^n$.

- Suppose the agent B can perform a maximum of m runs with rôle B-$r\hat{o}le$, and n runs with other rôles. If $SYSTEM$ meets $TimedNonInjectiveAgreement(B\text{-}r\hat{o}le, A\text{-}r\hat{o}le, ds, AuthTime)(B)^m$, then $SYSTEM$ meets $TimedWeakAgreement(A\text{-}r\hat{o}le, AuthTime)(B)^{m+n}$.

- If $SYSTEM$ meets $TimedWeakAgreement(A\text{-}r\hat{o}le, AuthTime)(B)^n$, then $SYSTEM$ meets $TimedAliveness(A\text{-}r\hat{o}le, AuthTime)(B)^n$.

We now show that 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, $TimedSpec(t) = TimedAliveness(A\text{-}r\hat{o}le, t)(B)$), and let $t \leq t'$. Then if

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

then

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

Proof: If $t \leq t'$ then $TOCKS(t) \sqsupseteq TOCKS(t')$, by analysis of their traces. Hence, for all processes P, we have $AddTime(P, t) \sqsupseteq AddTime(P, t')$ by monotonicity of the CSP operators. Hence $TimedSpec(t) \sqsupseteq TimedSpec(t')$, and so $TimedSpec(t)^n \sqsupseteq TimedSpec(t')^n$. Thus, if $SYSTEM$ meets $TimedSpec(t)^n$ then $SYSTEM$ meets $TimedSpec(t')^n$, from the definition of meets, noting that $\alpha TimedSpec(t) = \alpha TimedSpec(t')$. □

We now show that 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 $Spec$.

Proof: We have:

$$\begin{array}{ll} & SYSTEM \text{ meets } TimedSpec \\ \Rightarrow & \left\langle \text{definition} \right\rangle \\ & TimedSpec \sqsubseteq SYSTEM \upharpoonright \alpha TimedSpec \\ \Rightarrow & \left\langle \text{monotonicity of hiding} \right\rangle \\ & TimedSpec \setminus \{tock\} \sqsubseteq SYSTEM \upharpoonright \alpha TimedSpec \setminus \{tock\} \\ \Rightarrow & \left\langle TimedSpec \setminus \{tock\} = Spec\text{; definition of alphabets} \right\rangle \\ & Spec \sqsubseteq SYSTEM \upharpoonright \alpha Spec \\ \Rightarrow & \left\langle \text{definition} \right\rangle \\ & SYSTEM \text{ meets } Spec. \end{array}$$

□

Note that in the above lemma we were considering a system where the agents could timeout, even when we were considering an untimed specification.

Finally, we ask whether allowing for the possibility of a timeout can make a difference when considering an untimed specification. Surprisingly, the answer is yes. Consider the following, slightly unusual, protocol, which aims to achieve weak agreement, authenticating B to A:

$$\begin{array}{lll} \text{Message 1.} & A \rightarrow B & : \; A, B, ts \\ \text{Message 2.} & B \rightarrow A & : \; \{ts'\}_{SK(B)}. \end{array}$$

Here ts and ts' are timestamps. Suppose that when A receives message 2 he performs the test $ts' \geq ts + 10$, accepting B's identity if this is true, aborting otherwise; thus he will accept a message 2 only if it appears to have been sent at least 10 time units after the first message[2].

Consider the operation of this protocol in a system where A can timeout if a run takes longer than 5 time units. Then if A tries running the protocol with an honest B, and their clocks are synchronized, then A will always timeout before receiving the message 2. Hence the protocol meets the authentication specifications vacuously. However, in a system without timeouts, A may be able to complete a run. But there is a very straightforward attack against this protocol, where the intruder simply changes A's identity in message 1 to that of a third party C; thus the protocol fails to achieve weak agreement (or any of the stronger untimed specifications).

The above observations show that if we consider an untimed specification, but fail to model timeouts, then—for a small class of strange protocols—we risk detecting a false positive; fortunately, this error is on the safe side.

6 Conclusions

In this paper we have considered a number of different forms of authentication. We have formalized them using the process algebra CSP, and shown how to use

[2]Don't ask why such a test should be made; maybe the designer got the "$\geq$" sign the wrong way round.

the model checker FDR to test whether a system running a protocol achieves the various authentication requirements.

Our approach was to start by defining various notions of authentication that did not necessarily guarantee that any actions by the agent being authenticated were recent. In many protocols, recentness can be checked by testing whether the protocol guarantees agreement on a piece of data that is freshly introduced by the agent to whom authentication is made; however, when this method cannot be used, we lift the authentication specifications so as to produce timed versions.

Our authentication specifications have much in common with Schneider's work [Sch96a, Sch96b]. He uses a trace specification T $\mathsf{authenticates}$ 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:

$$T \ \mathsf{authenticates}\ R \ \widehat{=}\ tr \restriction R = \langle\rangle \Rightarrow tr \restriction T = \langle\rangle.$$

He shows that

$$P \ \mathsf{sat}\ (T \ \mathsf{authenticates}\ R) \ \Leftrightarrow\ P \underset{R}{\|} STOP \ \mathsf{sat}\ tr \restriction T = \langle\rangle.$$

He then uses this to test whether, in the Needham-Schroeder Public Key Protocol, messages received by one agent authenticate messages sent by the other.

Note that Schneider's test T $\mathsf{authenticates}$ R does not capture our notion of injectivity: if an event from R occurs, then there may be several subsequent events from T; however, it is not hard to adapt the specification to as to capture injectivity, for example by:

$$\#(tr \restriction R) \ \geq \#(tr \restriction T).$$

Schneider's test does not capture our notion of recentness, either: the events from T are allowed some time after the corresponding events from R; again it is possible to adapt the specifications to include recentness, as in this paper.

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:

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

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

$SYSTEM$ meets $Aliveness(A\text{-}rôle)(B)$
iff
$SYSTEM$ sat

$$\{signal.Commit.A\text{-}rôle.A.B.ds \mid ds \in Data^* \wedge A \in Agent\} \text{ authenticates}$$
$$\{signal.Running.B\text{-}rôle.B.C.ds' \mid B\text{-}rôle \in ROLE \wedge ds' \in Data^* \wedge C \in Agent\}.$$

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 [Low96b]; these case studies demonstrate the practicality of our methods.

When analyzing protocols using FDR we normally make a number of implementation assumptions, which are reflected in the way we model the agents taking part in the protocol, and also the intruder. For example:

- When we discussed recent authentication above, we made the assumption that agents would time out if runs are taking too long;
- When we model the intruder, we normally make the assumption that the intruder is not able to learn keys by cryptographic techniques: he may only learn a key by seeing it in a message;
- We normally assume that the cryptographic techniques we use have no "interesting" algebraic properties;
- We normally assume that the types involved in the protocol (such as nonces, keys, agents' identities) are disjoint, so, for example, an agent will not accept a nonce in place of a key.

However, none of these assumptions are really necessary; by altering the way we model the system, we can remove these assumptions (although this may lead to an increase in the state space that FDR needs to check). It is worth noting that the specification techniques discussed in this paper are independent of the implementation assumptions; if we were to change the implementation assumptions, we would still be able to use the same specifications.

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

[AN95] Ross Anderson and Roger Needham. Programming Satan's computer. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[BAN89] Michael Burrows, Martín Abadi, and Roger 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.

[BGH+93] 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.

[BM90] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, 1990.

[CCI87] CCITT draft recommendation X.509. the directory-authentication framework, version 7, 1987. Gloucester.

[CJ95] John Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, 1995.

[DS81] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[For96] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2—User Manual*, 1996. Available via URL `ftp://ftp.comlab.ox.ac.uk/pub/CSP/FDR`.

[Gol96] Dieter Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Research in Security and Privacy*, 1996.

[HC95] Tzonelih Hwang and Yung-Hsiang Chen. On the security of SPLICE/AS—the authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[HLL+95] Tzonelih Hwang, Narn-Yih Lee, Chuan-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53:103–107, 1995.

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

[ISO91a] International Organization for Standardization. *Information Technology—Security Techniques—entity authentication mechanisms; part 3: Entity authentication mechanisms using a public key algorithm*, March 1991. ISO/IEC JTC1/SC27/WG2 N51.

[ISO91b] International Organization for Standardization. *Information Technology—Security Techniques—Summary of voting on letter ballot No. 6, document SC27 N277. CD 9798-3.3 "Entity authentication mechanisms; part 3: Entity authentication mechanisms using a public key algorithm"*, October 1991. ISO/IEC JTC1/SC27 N313.

[ISO94] *ISO/IEC DIS 11770-2:1994 "Information technology—Security Techniques—Key management—Part 2: Mechanisms using symmetric techniques"*, 1994.

[KSL92] A. Kehne, J. Schönwälder, and H. Landendörfer. A nonce-based protocol for multiple authentications. *Operating Systems Review*, 26(4):84–89, 1992.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[Low96a] Gavin 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.

[Low96b] Gavin Lowe. Casper: A compiler for the analysis of security protocols, 1996. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~gl7/Security/Casper/index.html`.

[Low96c] Gavin Lowe. Some new attacks upon security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 162–169, 1996.

[MNSS87] 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`.

[NS78] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. A note on the use of timestamps as nonces. *Operating Systems Review*, 27(2):10–14, 1993.

[OR87] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, January 1987.

[Ros94] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[Ros96] A. W. Roscoe. Intensional specification of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.

[Sat89] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[Sch96a] Steve Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Security and Privacy*, Oakland, 1996.

[Sch96b] Steve Schneider. Using CSP for protocol analysis: the Needham-Schroeder Public Key Protocol. Technical Report CSD-TR-96-14, Royal Holloway, 1996.

[Syv94] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136, 1994.

[WL94] Thomas Y. C. Woo and Simon S. Lam. A lesson on authenticated protocol design. *Operating Systems Review*, 28(3):24–37, 1994.

[YOM90] 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.