Modelling unbounded parallel sessions of security protocols in CSP

E. Kleiner and A.W. Roscoe

Oxford University Computing Laboratory
{Eldar.Kleiner, Bill.Roscoe}@comlab.ox.ac.uk

Abstract. We show that a simplification to earlier CSP models designed to prove protocols correct on the FDR model checker is valid. This both allows us to extend the scope of our proofs and produce checks that are enormously more efficient.

1 Introduction

Model checkers such as FDR have been extremely effective in checking for, and finding, attacks on cryptographic protocols. However, their use in proving the correctness of such protocols was orginally limited, restricted by the finiteness of some set of resources such as keys and nonces.

In [16], it was demonstrated how, using techniques based on *data independence*, a simulation of a finite number of agents executing a protocol an unbounded number of sequential runs can be set up. The essence of that technique was to identify cryptographic objects such as keys and nonces once they had become stale (i.e. no longer known to a trustworthy agent). Therefore a finite collection of values could be recycled so as to achieve the effect of an infinite one.

In [4], the model of [16] was extended to allow multiple instances of a given identity with the motivation of detecting parallel attacks: ones that require one or more identities to be running several instances of the protocol at the same time. This depended on *internalising* at least part of each identity within the intruder. In effect this gives the intruder an oracle based on the behaviour of these agents. [4] did not completely solve the problems associated with parallel attacks, for example in the area of injective authentication. Essentially for historical reasons, that paper created a model with both internalised agents and the recycling model.

Some work has also been done in finding thresholds on the number of agents and runs beyond which the behaviour of a system does not change (e.g. [11, 21].) The problem with this approach is that it tends to generate large models which are sometimes infeasible for a model checker to handle.

In this paper we show that, for *static* protocols in the presence of internalised agents, most of the complex machinery developed in [16] to handle an infinite number of sequential runs becomes unnecessary. This produces models which

are orders of magnitude more efficient to run on FDR as well as being conceptually easier. The latter means that we are able to demonstrate clearly that our method can detect both secrecy and authentication attacks which require arbitrary numbers of parallel sessions from both participants in the protocol – this is an advance on [4]. It follows that when our methods do not find an attack then no attack (with the Dolev-Yao model with perfect cryptography) exists.

This paper is based mainly on a single example protocol. However the methods we develop work for just about all the static protocols we could have attempted with previous methods.

2 Modelling protocols with CSP

CSP [9] is well suited to modelling protocols since it is a language of interaction. The CSP_M language of FDR [15] contains a wide variety of datatype and functional programming constructs which make it possible to model cryptographic protocols succinctly and elegantly. In this section we summarise how a security protocol is modelled using CSP (a process which has been automated in Casper [10]) and how the model allows us to reason about it. Many further details can be found in [19].

2.1 The *Fact* datatype

The datatype *Fact* represents all the messages that pass between agents, their constituent parts and anything else (e.g. secret keys) that is relevant to building and understanding them. It is based on a set of atoms called *Atom* where the sets *Key*, *Agent* and *Nonce* are defined to be subsets of the atom set ($Key \subseteq Atom$, $Agent \subseteq Atom$ and $Nonce \subseteq Atom$.) This (and all subsequent constructs we describe) can be extended to deal with other primitives, such as hashing, and constructing forms such as set formation.

$$Fact ::= \text{ENCRYPT}.Fact.Fact \mid \text{SQ}.Fact^* \mid \text{ATOM}.Atom$$

In this paper we will use the Casper notation of writing $\{m\}_k$ for $\text{ENCRYPT}.k.m$ and will abbreviate $\text{SQ}.\langle m_1, \ldots, m_n \rangle$ to $\langle m_1, \ldots, m_n \rangle$ or $m_1, \ldots, m_n$. For example, we will denote the construct $\text{ENCRYPT}.k.(\text{SQ}.\langle a, na \rangle)$ by $\{a, na\}_k$.

2.2 Trustworthy agents

Every agent taking a part in the protocol can be modelled as a CSP process (As will be shown later, an agent can also be internalised in the intruder deduction set, but for now we will assume that each honest agent is implemented as a separate CSP process). We define process P_A denoting agent A, using the following events:

- *send.A.B.F* - symbolises agent A sending message F to agent B.
- *receive.A.B.F* - symbolises agent B receiving message F apparently from A.

In addition we define the following events for representing the state of mind of the agents while they are running the protocol.

- *claimSecret.A.B.*$\langle F_1, \ldots, F_n \rangle$ symbolises that A thinks that facts $F_1, \ldots, F_n$ are secrets shared only with agent B.
- *Running.A.B.*$\langle F_1, \ldots, F_n \rangle$ symbolises that A thinks he started a new run of the protocol with B where $\langle F_1, \ldots, F_n \rangle$ contains facts identifying this run.
- *Commit.A.B.*$\langle F_1, \ldots, F_n \rangle$ symbolises that A thinks he has just finished a run of the protocol with B where $\langle F_1, \ldots, F_n \rangle$ contains facts identifying this run.

The specific definition of the agent's process depends on the protocol we want to analyse. For example if we look at the modified NSPK protocol [12]

Message 1 $A \rightarrow B : \{na, A\}_{PK(B)}$
Message 2 $B \rightarrow A : \{na, nb, B\}_{PK(A)}$
Message 3 $A \rightarrow B : \{nb\}_{PK(B)}$

the process that represents one instance of a trustworthy agent A initiating this protocol is defined as follows:

$$
\begin{array}{l}
P_A^I(na) \widehat{=} \\
\bigsqcap_{B \in Agent} send.A.B.\{na, A\}_{PK(B)} \rightarrow \\
\Box_{nb \in Nonce} \left(\begin{array}{l} receive.B.A.\{na, nb, B\}_{PK(A)} \rightarrow \\ send.A.B.\{nb\}_{PK(B)} \rightarrow \\ claimSecret.A.B.\langle na, nb \rangle \rightarrow Session(A, B, na, nb) \end{array} \right)
\end{array}
$$

The responder role is similarly defined: henceforth we will assume that $P_A(na)$ can perform *either* the initiator *or* the responder role once (as agent A, using nonce na). The *claimSecret* event expresses the expectation of A that na and nb are secrets shared only between herself and agent B.

2.3 Modelling the intruder and putting the network together

Based on the Dolev-Yao model [7], we allow the intruder to have the following abilities when attacking a set T of trusted agents: (i) overhearing all the messages flowing through the network, (ii) intercepting messages, (iii) faking messages based on what he knows limited only by cryptography, and (iv) behaving as would any agent outside of T. As is customary, we do this via a deductive system: we say that $B \vdash F$ if fact F can be constructed by the intruder from the set of facts B. For example the following rules capture the intruder's ability to encrypt and decrypt.

$$\{F, K\} \vdash \{F\}_K \qquad \{\{F\}_K, K^{-1}\} \vdash F$$

The process representing the intruder is parameterised by X, which ranges over subsets of *Fact*, and represents all the facts the intruder has discovered to date. In this model the intruder gets every message sent by the honest agents

or by the server via the *send* channel. He then can pass it to the agents via the appropriate *receive* channel unless he decides to block it or fake a new message instead.

$$\begin{aligned} Intruder(X) \mathrel{\widehat{=}} & \ \Box_{F \in Fact \cap Messages}\ send?A?B!F \rightarrow Intruder(\overline{X \cup \{F\}}) \\ & \Box \Box_{F \in Fact \cap Messages}\ receive?A?B!F \rightarrow Intruder(X) \\ & \Box \Box_{F \in Secrets}\ leak.F \rightarrow Intruder(X) \end{aligned}$$

The initial state of the intruder is $Intruder(IIK)$ where IIK is the *Intruder Initial Knowledge*.[1]

What we then want to do is show that no attacks are possible in the infinite composition

$$SYSTEM = (|||_{(A,n) \in Inits} P_A(n)) \underset{\{|send,receive|\}}{\parallel} INTRUDER(IIK)$$

where $Inits$ is a set of pairs of honest agents and their initial nonces with the property that each agent has infinitely many nonces, and no nonce is owned by more than one agent or is owned by an agent and is in IIK. IIK should give the intruder the ability to behave arbitrarily as all non-honest agents and should contain all agents' names and public keys.

Thus $SYSTEM$ contains infinitely many copies of each role, each of which can run the protocol once. It contains all the behaviours, however, of indefinitely many copies that can each perform a protocol run as often as it likes in sequence, because parallel and independent threads can still perform their runs in sequence. This observation is the key to why our work in this paper provides significantly quicker protocol proofs than earlier CSP models. Naturally we cannot run $SYSTEM$ itself on FDR: this paper is devoted to developing methods of building an efficient finite model of it which we can run to discover its properties.

3 Internalising agents

The technique of incorporating (*Internalising*) the functionality of some participants into the intruder was originally suggested in [6] as a strategy for reducing the state space of protocol checks. The latter is achieved by broadening the deduction set of the intruder. It was subsequently noticed that *internalising* an agent A creates the effect of enabling A to participate in an infinite number of parallel instances of the protocol.

Internal agents were classified into two main categories. The first includes agents who do not, at any point of the protocol, introduce fresh values. The server role in the TMN [22] protocol is a good example for such agents. The server's task in this protocol is to receive two messages M_1 and M_3 and then to construct a corresponding message M_4 which only contains variables appearing

[1] In practice, in FDR, we use a model of the intruder which is equivalent to the process above but vastly more efficient to run: it treats the intruder as the parallel composition of one process for each learnable fact and uses a partial order reduction method called *chase* on this. This is discussed more later.

in M_1 and M_3. Therefore, this functionality can be accurately internalised in the intruder using the deduction $\{M_1, M_3\} \vdash M_4$.

The second and larger category contains agents that generate fresh values (such as nonces, keys and identities.) In [4], Broadfoot and Roscoe made the first attempt to internalise this type of agent. The authors defined a new type of deduction, a *generation*, which has the form $\mathtt{t}, X \vdash Y$ where $\mathtt{t}$ is a non-empty sequence of the fresh values, X is a finite set of input facts, and Y is the resulting set of facts the intruder deduce containing the fresh values from $\mathtt{t}$. Special processes, known as *manager processes*, were responsible for supplying the fresh values upon a *generation* request. Something like this is necessary whenever it is necessary for an internal agent to create actual fresh values (possibly subject to recycling as discussed earlier).

The problem with this modelling approach is that if the intruder is unrestricted, then he can perform any number of *generations* he wishes, each time requesting fresh values. Hence, since the manager process has a finite source, this will result in running out of fresh values. In order to overcome it, the authors of [4] introduced a protocol model property referred to as *just-in-time* (JIT). In the next sections we propose a different way to bound a system by dispensing with the need for *internalised* agents to produce genuinely fresh values.

4 Data Independence

Process algebra models of security protocols usually treat infinite types like keys, nonces and names *data independently* [18, 15] since all they do is to compare them for equality and to apply polymorphic constructors over them like tupling and (symbolic) encryption. For example, consider Message 2 in the the modified NSPK protocol: $Encrypt.(SKey(A), Sq.\langle Na, Nb, B\rangle)$ It is obvious that this construct does not depend upon the structure of the values it contains (Nb, Na, A and B) and does not expose supplementary information related to them.

Data independence results for security protocols [16] are based on a collapsing function ϕ to the data independent type(s) T of a process $P(T)$ and proving that the behaviour of $P(T)$ is *lifted* through the function and is therefore sufficiently similar to the behaviour of $P(\phi(T))$ to expose any attack. In the traces world, it is easy to see that this is always the case when ϕ is injective:

$$traces(P(\phi(T))) = \{\phi(t) \mid t \in traces(P(T))\} \qquad (*)$$

However, when ϕ is a non-injective function, it much harder to tell whether $(*)$ holds since it can change the results of the equality tests occur within P. In [16, 6], conditions were established that are sufficient for this collapsing to work. In essence these, **PosConjEqT'**$_C$, and *Positive Deductive System* state that no external agent ever requires an inequality (except with members of the set C of constants) to make progress, and that the intruder never depends on an inequality to make a deduction. Fortunately, these are true of most protocols and of the standard deduction system. We will assume these properties of the protocols and models we consider.

5 Modelling an infinite number of internal agents

We distinguish between two types of security protocol. The first one consists of those in which each participant might perform an unbounded number of sequential steps in any session, or in which there is an unbounded chain of participants (known as *stream* or *erecursive* protocols). The techniques developed by Roscoe in [16], were later adapted and used by Broadfoot and Lowe [2] to analyse such a protocol, known as the *Timed Efficient Stream Loss-tolerant Authentication protocol* (TESLA) [14]. The second type consists of *static* protocols in which every agent performs a fixed finite number of steps in each run of the protocol. By far the majority of the literature on model checking cryptographic protocols has been devoted to the latter. In this section we propose a method for modelling infinite number of internal agents in static protocols and demonstrate how it can be employed for proving the correctness of the modified NSPK protocol.

Suppose we wish to model a protocol involving two trustworthy identities, $\{Alice, Bob\}$, in the context of a potentially infinite number of other identities, none of whom are assumed to be trustworthy, and the intruder (who uses identity $Mallory$). In the particular protocol we are studying, each participant generates one nonce per run. We suggest the following collapsing function to reduce this to a finite model.[2]

$$\phi(X) = \begin{cases} X & \text{if } X \in Agent \text{ is the identity of one of the external agents} \\ Mallory & \text{if } X \in Agent \text{ is any other agent's identity} \\ X & \text{if } X \in Nonce \text{ is generated by one of the external agents} \\ N_s & \text{if } X \in Nonce \text{ is a secret and is generated by an honest internal agent in a session with an honest agent} \\ N_p & \text{if } X \in Nonce \text{and } X \text{ is generated by an honest internal agent in a session with the intruder} \\ N_p & \text{if } X \in Nonce \text{ is } \textbf{not} \text{ a secret} \end{cases}$$

Therefore $Alice, Bob, N_{Alice}, N_{Bob}$ and the intruder identity will be mapped by ϕ to themselves. All identities apart from *Alice*'s and *Bob*'s will be mapped to $Mallory$ who is under the control of the intruder. All nonces used by internal agents in sessions with honest agents will be mapped to N_s and the rest to N_p.

It might be the case that one does not know which values are secret and which are not. In such cases, a secrecy analysis needs to be performed before authentication checks to make sure which values are indeed secret. Secondly, by mapping all agents except *Alice* and *Bob* to $Mallory$, we are reflecting our assumption that that they are all potentially dishonest. In our example, nonces and identities were the only infinite type. ϕ can similarly extended to keys, indices or any other data independent type as required for different protocols.

Since all the processes in $SYSTEM$ are data independent in the types of identities and nonces, each of the CSP processes representing *Alice* and *Bob*

[2] ϕ can be extended in the obvious way to the whole of *Fact*

satisfy the **PosConjEqT'**$_C$ where $C = \{Alice, Bob\}$ and the intruder process, through having a positive deductive system, satisfies the **PosConjEqT'**$_C$, it follows that the behaviour before the mapping is equivalent to the behaviour after the mapping:

$$traces(SYSTEM(\phi(\Pi))) = \{\phi(t) \mid t \in traces(SYSTEM(\Pi))\}$$

for parameters Π. In this model we do not need the concept of *generations*, since all messages the intruder (including all other agents other than the external $\{Alice,\ Bob\}$) can generate are mapped by ϕ to one of

$$\{Alice, N_s\}_{PK(Bob)},\ \{Alice, N_p\}_{PK(Mallory)},$$
$$\{Mallory, N_p\}_{PK(Mallory)},\ \{Mallory, N_p\}_{PK(Bob)}$$

We therefore can include these messages in the intruder initial knowledge. The activity of the internal copies of $\{Alice,\ Bob\}$ are captured by the following deductions.

deduction (2.1)	$\forall A \in Honest \bullet \{A, na\}_{PK(B)} \vdash \{na, N_s, B\}_{PK(A)}$
deduction (2.2)	$\{Mallory, na\}_{PK(B)} \vdash \{na, N_p, B\}_{PK(Mallory)}$

Using this fixed collapsing function ϕ rather than the dynamic collapsing required in [4] has in effect made the internal agents unable to create genuinely fresh values: rather they create pre-collapsed ones, making for a much simpler model as we can use smaller types and do not need manager processes.

However the dynamic techniques are invaluable for modelling stream protocols as in TESLA [2] and WS-SecureConversation [8], something that many protocol analysis tools are unable to do.

6 Restricting internal agents' behaviour

In the previous sections we showed how actions taken by the honest agents can be internalised independently. Though this allows us to model an infinite number of agents, it also introduces surplus behaviours. For example, an internalised agent might perform step 3 without performing step 1 at all. In addition, the internalised agent cannot "remember" values he generated in previous steps of the protocol. For example, consider Message 4 in the following example taken from [5]:

Message 1. $A \rightarrow S : \{B, na\}_{ServerKey(A)}$
Message 2. $S \rightarrow B : ns, \{A, na\}_{ServerKey(B)}$
Message 3. $B \rightarrow S : \{nb\}_{ServerKey(B)}$
Message 4. $S \rightarrow A : \{ns, na, nb\}_{ServerKey(A)}$

Since the internalising technique deals with each step independently, there is no way to create a deduction for modelling S sending A Message 4. The problem is that such deductions cannot extract the value bound to ns in the second step.

In [5], Broadfoot offers to restrict the internal agents behaviour using *supervisor processes* put in parallel with the intruder constraining its behaviour and prompting it with the values that were introduced in previous stages of the protocol run.

The last problem lies in the way we model *internalised S* sending Message 2. if a *generation* is employed then ns might be bound to any value the *manager process* supplies. On the other hand, when using ϕ, all these values are collapsed into N_p. Therefore, there is no need to restrict the intruder from performing this *generation* many times. Subsequently, a deduction for modelling Message 4 can easily be constructed since we know that ns can only be bound to N_p.

Thanks to these arguments, we are able to offer a different method which does not require interference by external processes in the intruder's behaviour. We define a new construct $\textsc{Sent}.Message.Tag$ abbreviated by $\textsc{Sent}(M,T)$ which states that message M has been sent by an internal agent where T are the variables which had been instantiated by this agent up to the point in time M was sent. For example, when observing the external process representing the initiator (see Section 2), by the time Message 1 is sent variables A, B, na are instantiated. The latter is represented using the new constructor as follows:

$$\begin{array}{c}\textsc{Sent}(\{Alice, N_s\}_{PK(Bob)}, \langle Alice, N_s, Bob\rangle),\\ \textsc{Sent}(\{Mallory, N_p\}_{PK(Bob)}, \langle Mallory, N_p, Bob\rangle)\\ \textsc{Sent}(\{Alice, N_p\}_{PK(Mallory)}, \langle Alice, N_p, Mallory\rangle)\\ \textsc{Sent}(\{Mallory, N_p\}_{PK(Mallory)}, \langle Mallory, N_p, Mallory\rangle)\end{array}$$

Similarly, after the third message, the external process has instantiated nb as well. *deduction*(3) can be amended as follows to capture this fact.

$$deduction(3) \quad \left\{\begin{array}{l}\textsc{Sent}(\{A, na\}_{PK(B)},\\ \langle A, na, B\rangle), \{na, nb, B\}_{PK(A)}\end{array}\right\} \vdash \textsc{Sent}(\{nb\}_{PK(B)}, \langle A, na, B, nb\rangle)$$

This illustrates the restrictions placed upon agent A. A will not be able to send Message 3 unless he has sent Message 1 in the past. If *deduction*(3) was using the "pure" Message 1 construct instead of its *sent* version, the intruder would be able to perform this deduction even when Message 1 could be deduced independently, without agent A sending it.

In the same way agents playing the responder role are internalised using the following deductions:

deduction (2.1) $\quad \forall A \in Honest \bullet \{A, na\}_{PK(B)} \vdash \textsc{Sent}(\{na, N_s, B\}_{PK(A)}, \langle A, na, B, N_s\rangle)$

deduction (2.2) $\quad \{Mallory, na\}_{PK(B)} \vdash \textsc{Sent}(\{na, N_p, B\}_{PK(A)}, \langle Mallory, na, B, N_p\rangle)$

For performance reasons, we can let the external agents use N_p instead of their own nonce when they think they run a session with the intruder. Finally, since we want to give the intruder the ability to manipulate the messages sent by the internal agents, we add the following deduction:

$$deduction\ (sent) \qquad \textsc{Sent}(M, T) \vdash M$$

Even after restricting the internal agents' behaviours using the methods offered in this section, a limited number of false attacks may be found by our model. This number can be reduced (possibly even to none) at the expense of enlarging the state space of the model. We have to omit here the technical details regarding this matter due to lack of space.

6.1 Chasing the intruder's internal agents

It was observed in [17] that the deductive function of the intruder is *monotonic*; if $B \vdash f$ and $B' \supseteq B$ then $B' \vdash f$. So, performing one inference will never disable the inference of another fact. In particular it does not matter what order deductions are performed in. Since by default FDR considers all possible orders, a new external function, *chase*, was introduced in FDR input language. $chase(P)$ behaves like P except when it can perform a τ action; in that case, it executes τ actions until no more are available. If two τ actions can be performed, one of them will be chosen randomly and the other one will not be considered. Hence, the efficiency of the model can be improved by hiding the *infer* events and chasing the intruder process. This development reduced dramatically the state space of the model and in effect made complete security analysis using FDR possible. Since we want the number of chased τs to be finite the standard intruder model does not allow an *infer* action whose conclusion is already known.

The *internal deductions*[3] are no different to the standard ones in this respect. They keep the monotonicity of the deductive function and they are only dependent on whether facts are known rather then being enabled only if some facts are not known to the intruder. Thus, two deductions will not disable each other as long as they do not have the same conclusion.

Where intruder inferences are doubling as actions which can be observed from the outside by a specification (see next section), we can no longer hide them and therefore cannot *chase* them. For the same reason is neither necessary nor appropriate to prevent one whose conclusion is already known.

7 Specification considerations

As with the implementation, we can apply the collapsing function ϕ proposed in Section 5 to processes representing protocol specifications (we use the secrecy and authentication specifications from [19]). Since both secrecy and authentication specifications satisfy the $\mathbf{PosConjEqT'}_C$ property then, by data independence results, the behaviour of these processes before and after the mapping is congruent.

Yet one problem still remains. When an agent is modelled as a standard CSP process, *signal* events, encapsulating the agents' beliefs for specification purposes

[3] We use this term to refer to the set of all deductions that model the internal agents behaviours.

(as described in Section 2), can be constructed by appropriate renaming.[4] However, an internal agent does not perform any event, its functionality is embedded within the intruder deductive system.

Constructing a *signal* event related to a message dispatch of an internal agent is relatively straightforward. Since each *send* event is internalised using a deduction, the same effect can be achieved by not hiding the suitable *infer* event of the intruder process and renaming it in the top level system process. In our example, the *Running* event of *Alice* when model as a CSP process is constructed as follows:

$$INITIATOR(Alice, Na)$$
$$[\![send.Alice.B.\{nb\}_{PK(B)} \leftarrow signal.Running.INITIATOR\_role.Alice.B.Na.nb]\!]$$

This can be constructed in a similar way for internalised *Alice*:

$$INTRUDER(X)$$
$$[\![infer.\textsc{Sent}(\{nb\}_{PK(B)}, \langle Alice, B, na, nb\rangle) \leftarrow signal.Running.INITIATOR\_role.Alice.B.na.nb]\!]$$

The construction of the *signal* events associated with the receiving of some messages by internal agents is more intricate as there is no corresponding event in the intruder deductive system. However, as we show later in this section, these events are not needed for constructing the various specifications.

One problem might arise due to the way we construct the *Running* events. Since FDR calculates all the facts the intruder might deduce from her initial knowledge using her deductions at compile time, we might "lose" *Running* events which are a result of sessions between two internal agents or an internal agent with the intruder. This may lead to "false attacks". Additionally, since the first message of the protocol is already in the intruder's mind, it will never be inferred and therefore *Running* events corresponding to this message will never be performed. The former can be resolved by not letting the intruder "deduce" any fact using the deductions modelling internal agents at compile time. However, this looks unnecessary since it increases the state space and the problem it is rectifying will very rarely introduce false attacks. The latter can be easily mended by modifying the specification process or forcing the system to perform these *Running* events at the beginning without the intruder inferring the relevant messages.

The rest of this section is devoted to discussing how various security properties can be verified using the model described in this paper and thus how a more complete result upon protocol analysis can be achieved.

[4] Originally, these signal events were interleaved with the events representing the messages of the protocol. For efficiency reasons the model was redesigned such that the signals events are introduced at the topmost level, via renaming of message events of the overall system process [3].

7.1 Secrecy

The authors of [4] observed that symmetry can be exploited for checking efficiently whether a protocol P satisfies a secrecy specification. We describe briefly this technique and the theory underpinning it.

A specification of the form $Secret(A, s, [B_1, \ldots, B_n])$ specifies that in any completed run, A expects all values from set s to be a secret shared only with $B_1, \ldots, B_n$. The only type of signal event required for checking this specification is *claimSecret*. It is constructed through renaming of the last message of agents playing the role A.

Proposition 1 *If we model a protocol P with the set of roles AS and verify specifications of the form $Secret(A, s, [B_1, \ldots, B_n])$ such that*

1. *All the roles in AS are internalised.*
2. *One instance of role A is modelled externally.*
3. *The claimSecret signal event is constructed for the external instance of A, while no signal event is constructed for any other internal agent.*

Then, if no secrecy attack is found in this model, then none exists upon unbounded number of parallel sessions of P.

Proof: As was shown in Section 5, the $SYSTEM$ process represents infinite number of agents running P in parallel. By definition, the secrecy specification is broken if at least one instance of A believes that a secret is shared only with $B_1, \ldots, B_n$ while the intruder has been able to deduce it. Since all the instances of A are symmetric in the definition of $SYSTEM$ if such an attack exists upon any instance of A then it could be exercised upon the external one in the model we have created.

7.2 Authentication

In this section we demonstrate how to verify authentication properties using the model described in this paper. We consider two type of authentication specifications: injective and non-injective [13].

Non-injective authentication In a similar way to secrecy verification, symmetry can be used for capturing non-injective authentication properties for any degree of parallelism.

$NonInjectiveAgreement(A, B, [v_1, \ldots, v_n])$ specifies that if B thinks he has successfully completed a run of the protocol with A, then A has previously been running the protocol, apparently with B and both agents agree as to which roles they took and the values bound to $v_1, \ldots, v_n$. We use two types of signal events to verify this property - *Running* and *Commit*. A protocol satisfies this property if for every *Commit* event there has been a *Running* event that agreed with it (but not necessarily in a 1-1 relationship).

Proposition 2 *If we model a protocol P with the set of roles AS and verify specifications of the form $NonInjectiveAgreement(A, B, [v_1, \ldots, v_n])$ such that*

1. *All the roles in AS are internalised.*
2. *One instance of role B is modelled externally.*
3. *The Commit signal event is constructed for the external instance of B, while no Commit signal events are constructed for any internal agent.*
4. *The Running events are constructed for the internal instances of A.*

Then, if no non-injective authentication attack is found on B in this model, then none exists upon the unbounded number of parallel sessions of P modelled in $SYSTEM$.

Proof: As was shown in Section 5, the $SYSTEM$ process represents an infinite number of agents running P in parallel. The non-injective authentication specifications do not demand a one-to-one relationship between the runs of agents playing role A and those who play role B. Therefore, an attack exists if at least one agent of role B commits himself to a run with an agent that either has not been running the protocol or has done so with different set of variables. Due to the symmetry of the system it does not matter which instance of B participates in the run that leads to the attack. Therefore we can collapse the system down in such a way that the B process issuing this *commit* signal is external.

Injective authentication The specification $Agreement(A, B, [v_1, \ldots, v_n])$ for injective agreement is similar to the non-injective one set out in Section 7.2 only that a one-one relationship between the runs of B and A is now required. In other words each *Commit* event require a separate *Running*. As described in Section 2 we check whether an implementation meets this requirement by constructing a specification process that stipulates that there is a one-one relationship between the *Running* and *Commit* events.

Proposition 3 *If we model a protocol P with the set of roles AS and verify specifications of the form $Agreement(A, B, [v_1, \ldots, v_n])$ such that*

1. *All the roles in AS are internalised.*
2. *Two instances of role B are modelled externally.*
3. *The Commit signal event is constructed for the external instances of B.*
4. *The Running events are constructed for the internal instances of A.*

Then, if no injective authentication attack is found on B in this model, then none exists upon unbounded number of parallel sessions of P

Proof: As was shown in Section 5, the $SYSTEM$ process represents infinite number of agents running P in parallel. By definition if an attack as such exists, two different instances of B will commit themselves to the same run of an instance of A. Thanks to the symmetry if an such attack exists the intruder can exercise upon any pair of instances of B. Therefore, the attack can be exercised upon the two external instances of B. ∎

Some protocols guarantee B a one-one relationship by making A send values that were generated earlier in the protocol run by B (e.g. nonces). Alternatively, the injective correlation in other protocols might also be dependant upon fresh values introduced by A. In the latter, our proposition will introduce false attacks since all the fresh values of internal instances of role A are mapped to the same one. We believe that correct protocols that assure B injective authentication based on fresh values generated by A are rare or even do not exist. Unless B remembers which values were used in previous runs, it is hard to see how he can verify freshness without introducing freshness himself.

Although it seems unnecessary, Proposition 3 can be rectified such that these sorts of false attack on the second class of protocols will not be found by FDR. This can be achieved by adding an external instance of A to the model and devising a specification that allows only one of the external instances of B to commit to the external instance of A but not to both. Since we know that the injective authentication specification is broken if at least two instances of B commit themselves to A then due to symmetry if such attack exists it can be exercised upon the external instances of A and B.

8 Discussion

[4] created a model for protocols which was, in essence, a superset of the one we have built here. The crucial simplification we have made is based on the observation that we do not need to create an external agent which can run the protocol an unbounded number of times sequentially. It is enough (thanks to symmetry arguments) to model externally the specific runs of the protocol that might be attacked, and leave the rest internal to the intruder model. This allows us to dispense with the recycling and generation management functions of the earlier paper, meaning that ours is both conceptually simpler and typically very much faster to run.

The time taken to run FDR checks attempting to prove protocols has been reduced from perhaps hours to less than a second by this simplification, making it competitive with all other tools for this purpose. (No protocol has taken more than a second using our new model.)

Formally, our methods can introduce false attacks, but we have not found this to be a problem. There are obvious similarities between our models and the Strand Space model [23] and therefore *Athena* [20], which should be investigated, which should also lead to criteria under which our method is guaranteed not to introduce false attacks.

Since Athena uses logic for specifying the requirements of the protocol, it cannot be used for reasoning about freshness properties such as injective authentication. This problem is also common in tools that are not based on model checking. For example, *Proverif* [1] uses the following abstractions to capture unbounded runs and degree of parallelism. (1) fresh values are represented as functions over the possible pair of participants and (2) a protocol step can be executed several times, instead of one per session. Clearly when using the first

abstraction it is not longer possible to distinguish between old values from previous runs and new ones. Since FDR achieves similar performance when analysing protocol using the model presented in this paper, its ability to reason about freshness properties is an advantage of our approach over such tools.

We see no reason in principle why the techniques presented in this paper will not be applicable to other notations and in particular other model checkers.

It should be straightforward to integrate our new techniques into Casper [10], but this has not been done at the time of writing.

References

1. B. Blanchet. An Efficient Cryptographic protocol verifier based prolog. *14th IEEE Computer Security Foundations Workshop (CSFW 14)*, 2001.
2. P. J. Broadfoot and G. Lowe. Analysing a stream authentication protocol using model checking. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security*, pages 146–161, London, UK, 2002. Springer-Verlag.
3. P. J. Broadfoot, G. Lowe, and A. W. Roscoe. Automating data independence. In *ESORICS '00: Proceedings of the 6th European Symposium on Research in Computer Security*, pages 175–190, London, UK, 2000. Springer-Verlag.
4. P. J. Broadfoot and A. W. Roscoe. Embedding agents within the intruder to detect parallel attacks. *Journal of Computer Security*, 12(3-4):379–408, 2004.
5. P.J. Broadfoot. *Data Independence in the model checking of security protocols.* PhD thesis, University of Oxford, Oxford, UK, 2001.
6. P.J. Broadfoot and A.W. Roscoe. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security: Special Issue CSFW12*, 1999.
7. D. Dolev and A.C. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.
8. E. Kleiner and A.W. Roscoe. On the relationship of traditional and Web Services Security protocols. In *Mathematical Foundations of Programming Semantics (MFPS 05)*, Birmingham, UK, 2005.
9. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
10. G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
11. G. Lowe. Towards a completeness result for model checking of security protocols. In *CSFW '98: Proceedings of the 11th IEEE Computer Security Foundations Workshop*, page 96, Washington, DC, USA, 1998. IEEE Computer Society.
12. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.
13. Gavin Lowe. A hierarchy of authentication specifications. In *CSFW '97: Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97)*, page 31. IEEE Computer Society, 1997.
14. Adrian Perrig, J. D. Tygar, Dawn Song, and Ran Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy (S&P 2000)*, page 56, Washington, DC, USA, 2000. IEEE Computer Society.

15. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.
16. A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop, 9–11 June, 1998*, pages 84–95. IEEE, 1998.
17. A. W. Roscoe and M. H. Goldsmith. The perfect "spy" for model-checking cryptoprotocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, Rutgers University, September 1997.
18. R.S. Lazić. *A semantics study of data-independence with applications to the mechanical verification of concurrent systems.* PhD thesis, University of Oxford, Oxford, UK, 1999.
19. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. Modelling and analysis of security protocols, 2001.
20. Dawn Xiaodong Song, Sergey Berezin, and Adrian Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.
21. Scott D. Stoller. Justifying finite resources for adversaries in automated analysis of authentication protocols. In *Workshop on formal methods and security protocols*, June 1998.
22. M. Tatebayashi, N. Matsuzaki, and D.B. Newman. Key distribution protocol for digital mobile communication systems. In *Advance in Cryptology — CRYPTO '89*, volume 435 of *LNCS*, pages 324–333. Springer-Verlag, 1989.
23. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 1999.