Internalising agents in CSP protocol models

P.J. Broadfoot and A.W. Roscoe
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD

1 Introduction

We carry forward the work described in our previous papers [3, 12, 10] on the application of data independence to the model checking of cryptographic protocols using CSP [11] and FDR [5], often via extensions to Casper [6]. Since FDR can only check a finite instance of a problem it was originally only possible to check small instances of security protocols (only involving a few agents and runs). This was excellent for finding attacks, but unsatisfactory as a method of proof of correctness. There has been work on getting round this limitation in a variety of related approaches to protocol modelling, for example [7, 9, 13].

In our previous papers we showed how techniques based on *data independence* [4, 11] (where programs treat certain types simply and can be viewed as having these types as parameters) could be used to justify, by means of a single finite FDR check, systems in which the number of agents was unbounded and in which each could undertake an unbounded number of runs of the protocol. Most of this work was devoted to showing how a finite type could give the illusion (in a way guaranteed to preserve any attack) of being infinite by a careful process of on-the-fly mapping of values of this type (which might be nonces or keys) once they have been forgotten by trustworthy processes (i.e., become *stale*). Since the necessary CSP codings of security protocols, having been rather complex prior to this work, became far worse with these mappings implemented, their creation was automated in Casper.

Aside from restrictions necessary to make our results work (see below), and assumptions common across the whole field arising from the symbolic representation of cryptographic primitives, there was one significant incompleteness in the results we obtained. This was that, while each individual identity could perform an unlimited number of protocol runs, it usually had to do them in sequence. (For small protocols it was possible to run two parallel instances of an agent, but even that was of course far from unbounded!)

We now report significant progress towards the solution of this problem, by means anticipated in [3], namely by "internalising" all or part of each agent identity within the "intruder" process. The internalisation of agents (initially only server roles) was introduced in [12] as a state-space reduction technique (for which it was usually spectacularly successful). It was quickly noticed that this had the beneficial side-effect of making the internalised server arbitrarily parallel, at least in cases where it did not generate any new values of data independent type. But there were two problems which prevented us from immediately internalising all agents so that the sequentiality problem disappeared.

- The first (which applies to servers as well) is that an internalised agent which creates a value during a run can, if it has arbitrarily many protocol runs "live" at the same time, require an unbounded number of fresh values. Our existing methods of mapping stale values could not handle this situation, so there was no way of achieving the essential goal of keeping types small and finite.
- An essential part of our CSP models is knowing what a given agent believes about the progress of its protocol runs. To this end we have typically either treated specific protocol messages they send or receive as evidence for this or included specific signals (to the environment) in the definitions of processes representing trustworthy agents. This is not an issue for server processes, but it is much harder to "get into the minds" of internalised agents, something necessary if their progress on protocol runs plays a part in our specification.

In the rest of this extended abstract we summarise the techniques we have evolved for internalising agents, as well as the solutions we have devised for the two problems described above.

As in our previous papers, we restrict our attention to protocols where each run involves a fixed number of participants (in our examples invariably two plus perhaps a server). While agents can rely on equality between two values of a given type (e.g. nonces) for progress, they never rely on inequality (except perhaps with the members of a fixed finite set of constants). A similar condition, termed *positive deductive system* applies to the inferences made by the intruder. For more details see [12].

This paper is designed to give the reader an understanding of the most important ideas and definitions behind our work. However there is not space here for many technical details or examples of sufficient complexity to demonstrate the capture of interesting parallel attacks. These can all be found, together with many examples, in (mainly Chapters 5 and 6 of) the first author's D.Phil. thesis [2].

2 Internalising agent roles

The natural view of an intruder is of an entity who is trying to break the protocol by manipulating the messages that pass between well-behaved agents and the server (if any). Therefore placing either a server (alternatively known as "trusted third party") or an agent we wish to trust within the intruder seems bizarre. However that is not really what we are doing, which is to replace an agent/server with a set of inferences of the style used within our coding of the intruder that reflect what the intruder would see if it communicated with the trustworthy agent. The intruder is never given the secrets of a trustworthy process, only a logical picture of what it looks like from the outside when using the other party as an *oracle*.

Deductions performed by the intruder are usually modelled by pairs of the form (X, f), where X is a finite set of facts and f is a fact that it can construct if it knows the whole of X. The functionality of internal agents that do not introduce any fresh values is captured by this type of deduction within the intruder: we get a deduction (X, f) if, after the agent is told the messages in X, it can be expected to emit f (where f will be functionally dependent on X). The server role in the TMN protocol [14] is such an example. Internal agents that do introduce fresh values are captured by a special type of deduction, known as a *generation*. A *generation* has the form (t, X, Y), where t is a non-empty sequence of the fresh objects being created, X is a finite set of input facts, and Y is the set of facts generated containing the fresh values in t. The processes (known as managers) responsible for supplying the necessary fresh values must synchronise with the intruder upon these generations. Internal servers that introduce fresh keys or agents introducing fresh nonces are captured by these generations, where t contains the fresh keys and nonces respectively.

Example 1 *Consider the following first two messages of a hypothetical protocol description:*

$$\begin{array}{llll} \textit{Message 1.} & A \rightarrow S & : & \{B,\, n_a\}_{SKey(A)} \\ \textit{Message 2.} & S \rightarrow A & : & \{n_a,\, k_{ab}\}_{SKey(A)} \end{array}$$

where A is an agent introducing the fresh nonce n_a and S is a server introducing the fresh key k_{ab}. If S is modelled as internal, then its functionality is captured by the following generation:

$$\langle k_{ab}\rangle,\ \{B,\, n_a\}_{SKey(A)} \quad \vdash \quad \{n_a,\, k_{ab}\}_{SKey(A)} \tag{1}$$

Each time such a generation takes place, the key manager synchronises with the intruder and determines which fresh key is bound to k_{ab}.

∎

When internalising roles (especially non-server ones) it is often necessary to restrict the patterns of these deductions and generations within the intruder so that they correspond better to the behaviour of real agents. This is done (see [2]) by means of a special class of constraint processes called *Supervisors*. These are designed to ensure that the internal agent's behaviour, after a given generation, follows the protocol sequentially and most particularly does not miraculously "branch" into several continuations of the same run. These processes are, like the rest of the CSP model of a protocol, created automatically by Casper.

3 Bounding the appetite of internal agents

Where an internal agent can perform a generation because it has the job within the protocol of generating fresh values, there is no way we can run the resulting system unrestricted since the intruder can gain knowledge of an unbounded collection of messages, all containing different fresh members of the type.

Example 2 *Consider the generation of S in Example 1. If the key manager is given n fresh values to supply the network with and the intruder has intercepted the message 1 $\{Bob, N_A\}_{SKey(Alice)}$, then he could perform the following sequence of generations:*

$$\begin{array}{rcl} \langle K_1 \rangle,\ \{Bob, N_A\}_{SKey(Alice)} & \vdash & \{N_A, K_1\}_{SKey(Alice)} \\ \langle K_2 \rangle,\ \{Bob, N_A\}_{SKey(Alice)} & \vdash & \{N_A, K_2\}_{SKey(Alice)} \\ & \vdots & \\ \langle K_n \rangle,\ \{Bob, N_A\}_{SKey(Alice)} & \vdash & \{N_A, K_n\}_{SKey(Alice)} \end{array}$$

thereby always being able to cause the key manager to run out of values, irrespective of the value bound to n.

∎

Hence, the only way to keep the number of fresh values manageable (or even bounded) is to prevent the intruder from storing any number of fresh values for later use. What we do in this section is to show how, in certain circumstances, such restrictions can be justified formally.

A generation performed by an internal agent offers two distinct opportunities to the intruder: to perform inferences on the information it gains and to deliver messages containing the new fresh values to trustworthy agents. Of course these might be combined to create a different message with the fresh values. In seeking results about bounding the number of fresh values not yet delivered to an external agent that the intruder can hold, we have had to analyse these two functions carefully.

A key idea which allows us to derive rigorous bounds on number of fresh values that need to be held by the intruder is the following definition.

Definition 1 (Just-in-time) *Suppose we have a CSP protocol model with a number of externally modelled agents, together with an internal agent S, where S introduces fresh values of some type T.*

We say that a fresh value t of type T, received by an external agent, is generated "just-in-time" precisely when t is freshly introduced (via the corresponding generation of S) after all the protocol messages that precede the receipt of t (in some message M) by the external agent.

We say that S satisfies the "just-in-time" property with respect to type T precisely when, for every value t of type T received by an external agent, t can be generated just-in-time.

∎

Intuitively, if the just-in-time property holds in a given protocol model, then there is no advantage to be gained by the intruder to store fresh values, unknown to any external agent processes, that will only be introduced into the network later on in the trace. Notice that this property is concerned only with those fresh values that *are* eventually passed on to external agent processes and the point at which *they* are generated; the fact that the intruder can store fresh values that he never passes on to external agent processes is an issue we discuss later on.

When a protocol model satisfies this property, a version with an infinite supply of fresh values has an equivalent *reduced* system with a finite source of fresh values. By equivalent, we mean that no attacks are lost through the mapping; thus, if no attack is found upon the reduced system, then we can conclude that none exists upon the original infinite version of the system. This is achieved by extending our CSP models as follows.

The fresh values requested by the intruder on behalf of internal agents can (with the benefit of hindsight after a run is complete) be divided into two classes. The first are those values that get passed on to external agents (though perhaps not in the format in which they were generated); the second are those that remain internal within the intruder. Note that the just-in-time definition covers the first of these but not the second. Our extension is designed to deal with the latter. The observation we make is that it does not actually matter *which* internal fresh values are supplied; what is important is that they exist in some form for the purposes of allowing the intruder to perform whatever manipulations he needs in order to construct the messages he does. It is not even necessary for these values to be *fresh*, since they are never passed on to external agents. The intruder could just as easily perform subsequent deductions and generations with any values that were strictly for internal use only.

Based on this observation, we introduce a new class of values, referred to as *dummy* values, that will be added to the data independent types being generated. These extra values have the special characteristic that they are not accepted as genuine by any honest process (so the latter will never accept any message involving one). The intruder can use these values itself like any others, in particular doing deductions involving them. The trick is that we allow the intruder to perform, at any time, a "generation" based on a valid input set X, but unless the number of fresh values he is currently storing is less than the given bound, the result will always be based on a dummy value; otherwise, the result may be either a fresh or dummy value. By doing this, we ensure that any bound we place on the intruder's memory places no restriction on his ability to use generated messages in his internal workings.

We can then show that if a protocol model satisfies the just-in-time property and implements the dummy value strategy, then the appetite of the intruder need never be larger than the maximum number of fresh values that an external agent can learn in a single message.

However this property by no means applies to all agent roles, so we need to understand when it does. One trick that is frequently useful in the manipulations is to split a single run of the internal agent into several: one might be done early so as to allow the intruder to use the messages generated for an earlier message (either a message not involving the generated value which nevertheless appears after it might be sent, or messages might be used as the premises of a deduction). The other run would be done just in time to deliver the fresh values. Such splitting involves use of replays and *dummy* values. The best tool we have developed to understand when this type of manipulation can be done is that of the *factorisability* of an agent role within a protocol: this applies when we can split any run into several, each one containing at most one non-*dummy* value from the data independent type, together with other conditions.

Definition 2 (Factorisability) *We say that an internal agent A is factorisable with respect to some data independent type T precisely when, for each run R of A that generates fresh values $v_1, \ldots, v_k$ of type T, the following conditions are satisfied:*

1. *We can construct runs $R_1, \ldots, R_k$ of A, where each run R_i contains the fresh value v_i and the dummy value only.*
2. *For each output message M in R, there exists at least one R_i that contains M, where $i \in \{1, \ldots, k\}$.*
3. *R never depends on receiving any of the fresh values $v_1, \ldots, v_k$ back from any external agent processes. In other words, a* send *event in R is never preceded by a* receive *event in R of some message M where M contains any values in $\{v_1, \ldots, v_k\}$.*

∎

The relationship between the just-in-time property and factorisability is captured by the following proposition:

Proposition 1 *Suppose we have a CSP protocol model with a number of externally modelled agents, together with an internal agent S, where S introduces fresh values of type some T. If S is factorisable, then S satisfies the just-in-time property.*

A formal proof is presented in [2].

∎

Our aim was to be able to identify classes of protocol models that satisfy the just-in-time principle (with regards to the internal agent roles). This is not an intuitive task. On the other hand, verifying whether a given protocol model is factorisable is relatively straightforward. Thus the result above provides a useful way of determining and using the just-in-time arguments (together with the dummy value strategy) described earlier.

Using these results, we are able to capture attacks upon protocols that require a higher degree of parallelism than previously possible; for some classes of protocols, we have shown that attacks requiring any degree of parallelism amongst internal agents will be found within the confines of a finite refinement check. An example of such a class is defined as follows. Suppose $System_P$ is the CSP model for some protocol P and A is a participating role modelled as internal that introduces fresh values of some type T. We show that if the following conditions are satisfied:

1. A introduces at most 1 fresh value of type T per message.
2. For every message M of A, the generation of M never depends on A receiving a fresh value previously introduced by A.
3. The intruder can store N fresh values of type T, unknown to any external agents, where N is the maximum number of values of type T in any single protocol message.

then no attack found upon $System_P$ implies that no attack exists upon P for any degree of parallelism within A. A formal proof is presented in [2], together with other results of the same type.

Our ultimate goal, of course, is to be able to prove as wide a range of security properties as possible for protocols in which not just one but all the roles are arbitrarily parallel. The following result represents some progress towards this.

Suppose we model a protocol P consisting of two roles A and B, and wish to verify the secrecy specification $Secret(A, s, [B])$; this means that A is a trustworthy agent who believes that the value bound to variable s is a secret shared only with agent B. We require signal events bound to appropriate events performed by A only (as formally defined in [6]). We can capture secrecy attacks for any degree of parallelism for *all* agents roles (in this case A and B) within the confines of a finite refinement check, by constructing the CSP model as follows: (i) The roles A and B are modelled as internal, (ii) one instance of A is modelled as an external agent process and (iii) the necessary signal events are constructed for the external instance of A only.

We justify this claim as follows. The functionalities of all the agents in AS are internalised within the intruder and therefore, given that we have calculated an appropriate bound upon him, this represents an unbounded degree of parallelism amongst them (for the same reasons described in the earlier propositions). By definition, a secrecy specification of the form $S(A,\ s,\ [B_1, \ldots, B_n])$ is broken precisely when there exists an instance of A who believes that the value v bound to s is shared only with $B_1, \ldots, B_n$, while in fact the intruder has been able to deduce v. It does not matter which instance of A this is; the only requirement is that there exists one of them. Our model has one external instance A_E of A with the appropriate signal event linked to it, and so any attack the intruder can perform upon an internalised instance of A, he can also perform upon A_E. Therefore, it suffices to have signal events for the secrecy specifications linked only to A_E, and none associated to the internal instances of A.

We expect that similar arguments to those used earlier for a single internal agent will allow the restriction of the intruder's appetite for fresh values to manageable proportions in many cases, but this, together with widening the range of specifications, remains work in progress.

4 Basing specifications on internal agents

When an agent A is modelled as a standard external process, signal events (capturing the state of mind of A for specification purposes) are constructed through the appropriate renaming of messages sent and received by A. An internal agent no longer performs *send* and *receive* events, since its functionality is solely captured within the intruder's deductive system.

Capturing the sending of a message M by A is relatively straightforward, as this corresponds to the deduction or generation of A resulting in M. On the other hand, constructing signal events for an internal agent A that are bound to the *receiving* of some message M by A is more complicated, since the intruder's deductive system does not directly capture this information. A solution to this is to ensure (artificially if necessary) that such receipts are immediately followed by the same agent performing some *send*.

It is thus possible to verify authentication specifications where those roles satisfying the just-in-time principle are arbitrarily parallel. Such proofs do not at present exclude one-to-many attacks in which A and B can think they are having different non-zero numbers of runs with each other.

5 Conclusions

As well as proving to be a highly effective state space reduction strategy, we have shown that the internal agent model frequently permits protocols to be analysed with some agents having an arbitrary degree of parallelism. An example protocol we used for testing purposes was an extended version of the hypothetical *ffgg* protocol by Millen [8], where the secrecy attack requires three instances of an initiator agent, together with a single instance of a responder. Using old modelling techniques, this model is infeasible to run; using our new techniques and internalising the initiator role, this attack was found very easily.

Further work planned includes broadening classes of conditions and specifications where we can use these techniques, wherever possible on all the identities present in a given protocol.

Blanchet [1] uses a similar idea to ours, especially with regards to the internalisation strategy of agents (referring to the early stages of this development in [3]). The author presents a prolog based framework with the following two abstractions: (i) fresh values are represented as functions over the possible pairs of participants and (ii) a protocol step can be executed several times, instead of only once per session. With these abstractions, the author is able to capture unbounded agent runs and degrees of parallelism within their identities. Similar to ours, his analysis is fail-safe in the sense that no attacks are lost; however, the potential for false attacks does arise. However, by modelling values that are expected to be fresh for every run as functions over the participants, it is no longer possible to distinguish between old values used in previous ones and new. This means that one cannot verify properties that depend upon freshness. Blanchet currently only considers secrecy specifications.

Acknowledgements

This work has benefited enormously from discussions with Gavin Lowe. It was supported by funding from US ONR, DERA and EPSRC. We are also grateful for the comments by anonymous referees.

References

[1] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 82-96. IEEE Computer Society Press, 2001.

[2] P.J. Broadfoot. *Data independence in the model checking of security protocols.* D.Phil thesis, University of Oxford, submitted September 2001.

[3] P.J. Broadfoot and G. Lowe and A.W. Roscoe. Automating data independence. In *Computer Security - ESORICS 2000*, volume 1895 of LNCS, pages 175-190. Springer, 2000.

[4] R.S. Lazić, *A Semantic Study of Data Independence with Applications to Model Checking*, Oxford University D.Phil. thesis, 1999.

[5] *Failures-Divergences Refinement: FDR2 Manual.* Formal Systems (Europe) Ltd, 1997.

[6] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53-84, 1998.

[7] C. Meadows. The NRL Protocol Analyser: An overview. *Journal of Logic Programming*, 26(2):113-131, 1996.

[8] J. Millen. A Necessarily Parallel Attack. In *Proceedings of the Workshop on Formal Methods and Security Protocols.* 1999.

[9] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 1998.

[10] A.W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *11th IEEE Computer Security Foundations Workshop.* IEEE Computer Society Press, 1998.

[11] A.W. Roscoe. *The Theory and Practice of Concurrency.* Computer Science. Prentice Hall, 1998.

[12] A.W. Roscoe and P.J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security: Special Issue CSFW11*, 7(2,3):147-190, 1999.

[13] D.X. Song, S. Berezin and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security: Special Issue CSFW12*, 9(1,2):47-74, 2001.

[14] M. Tatebayashi and N. Matsuzaki and D.B. Newman. Key distribution protocol for digital mobile communication systems. In *Advances in Cryptology: Proceedings of Crypto '89*, volume 435 of *LNCS*, pages 324-333. Springer-Verlag, 1990.