Modelling and Analysis of Security Protocols

Peter Ryan and Irfan Zakiuddin
DRA, Malvern

Abstract

We outline the aims and principle results of a research programme aimed at developing tools and techniques for the analysis and design of security protocols. The aim was to move away from specially tailored frameworks and tools and apply well-established, mainstream formal methods and tools to the problem. The main thrust was to apply CSP and model-checking however some other strands: the B-method, Action Systems and theorem proving (using PVS), are also outlined.

1 Introduction

Security protocols are increasingly ubiquitous in the information age. The need to establish a high level of confidence that they do indeed provide the security services claimed for them has long been recognised. At the same time the difficulty of establishing such assurance or even of stating precisely what these security services and properties should be has also been acknowledged.

Various approaches to the problem have been developed in the past. The seminal work of Burrows Abadi and Needham who developed the BAN logic, [BAN], of authentication has greatly influenced the subject. Early on Kemmerer, [KMM], experimented with Ina Jo. Another line of attack was that of Meadows. [KMM], to treat the problem as one of algebraic rewriting and develop the NRL Analyser. A similar approach was pursued by Millen with the MITRE Interrogator.

It seemed widely held in the community that some kind of belief logic was required to address the problem of security protocols. This is quite plausible in that ultimately one is concerned about certain states of belief of the participants in the protocol: when Bob receives a certain message that purports to be from Alice did it really originate from Alice and was it really intended for him, can he be confident that a certain session key really is safe for private communication with Alice etc. The BAN logic, and derivatives, introduce axioms that are intended to represent how the belief states of the participants change in the light of new observations.

BAN logic has proved itself to be very useful and indeed has, as Needham puts it, "several scalps under it's belt". The drawback is that it can be difficult to interpret the exact implications of a proof performed using the logic. The idealisation process in particular is rather delicate and error prone as pointed out by Boyd and Mao, [BM].

On further reflection however we realise that it is perhaps not inevitable that we use a belief logic as our principal tool for analysis. The modelling and analysis of secure systems in general has long been conducted using non-logical frameworks: Z, CSP etc. Eventually one seeks to derive from such analysis notions of assurance i.e. belief that certain security services can be relied on but is derived from the analysis rather than being central to it. Thus for example the analysis might establish that a message purportedly from Alice can only appear on Bob's user channel if indeed Alice had earlier dispatched such a message and addressed it to Bob. Thus on receiving such a message Bob can be confident that it really was from Alice. Similarly we might show that a new session key Kab that the server generates for Alice and Bob will only be known by, at most, Alice and Bob (and maybe the server) and so on.

In this way we see that a perfectly conventional mathematical/formal methods framework can be used to reason about security protocols. This makes available all the theory, tools and experience built up in this area. It also has the advantage that we don't have to rely on intuitions about, for example, the meaning of authentication.

Various people, including one of the authors, [PYAR], have used CSP to formulate security properties such as non-interference, non-deducability etc. In this CSP had shown itself well suited. Add to this the observation that CSP is ideally suited (by construction) to modelling and analysing systems interacting via messages (i.e. protocols) and we conclude that CSP should be an ideal framework for analysing security protocols. The final ingredient that really makes this approach fly is the availability in recent years of a powerful, highly automated tool to support CSP, namely the FDR refinement/model-checker.

These considerations led to the formulation of a research proposal entitled "Modelling and Analysis of Security Protocols" to the DRA/MoD "Corporate Research Programme". The aim was to apply mainstream formal methods, in particular CSP but also the B-method. The programme started April 1994 and ended in March of this year.

In this paper we give an overview of the aims and results of this programme. More detailed descriptions of some of the strands of this programme can be found elsewhere in these proceedings: Goldsmith and Roscoe on "The perfect spy", Bryans and Schneider on the application of CSP and PVS. Consequently we will not dwell on these aspects of the programme but seek only to put them in the broader context.

2 Structure of the Project

The central thrust of the project has been the use of Communicating Sequential Processes (CSP) and the model-checker FDR. CSP is a mathematical framework for the description and analysis of systems consisting of components (processes) interacting via the exchange of messages, [CARH, AWR1]. It is clear that CSP is well suited to the study of protocols, indeed it has already been used extensively in the study of communication protocols. In particular this means that the passage from the conventional description of the protocol to CSP is fairly straightforward and indeed largely automatable, as has been demonstrated with Lowe's CASPER compiler, see section 8.

A model of the system in question along with any hostile agents is constructed in CSP. We outline this in section 3. Then the security properties required of the system are formulated in CSP. This is outlined in section 4. It then remains to analyse the system against the properties. The main effort in this respect was based on the use of the model-checker FDR to perform this analysis. This checks that one CSP specification refines another or returns a counter-example if the refinement is false. This is outlined in section 5. In the latter part of the project the use of theorem proving supported by PVS was also investigated by Schneider. This reported in Bryans and Schneider in this proceedings so we will not elaborate on this here.

Note that both the system and the properties are formulated in the same framework, i.e. CSP. It might be objected that it is restrictive to use the same notation for the properties but in fact we have found that all the properties of interest can be coded in CSP in a fashion that is both easy to grasp intuitively and practical for analysis.

The advantage of the model-checking approach is that it is highly automated. Another appealing feature is that if the check fails then FDR returns a counterexample: a trace that violates the refinement. In the context of security protocols such a counterexample is in fact the interactions of the spy in an attack.

Another attractive feature of this style of analysis is that an explicit and flexible model of the hostile capabilities is incorporated. This allows us to tailor a realistic threat model appropriate to the application. For example, if we are dealing with a system using broadcast messages then it might be deemed unreasonable to give the spy the ability to kill messages. This is in contrast with certain other approaches in which the spy has fixed capabilities. A prime example is the elegant spi-calculus approach of Abadi and Gordon in which the spy is incorporated as an arbitrary environment.

The drawback of model-checking is that the models have to be finite and indeed rather small. This means that in practice considerable ingenuity may be required to keep the models compact enough whilst not sacrificing any significant behaviour. Indeed in many ways this is the major challenge facing this approach. We will discuss how such reductions can be performed in more detail in section 4 and future directions in tackling this in section 12.

Another strand was the application of the B-method. Indeed it was a talk by Abrial in 1992 on applying the B-method to the Needham-Schroeder secret key protocol than was instrumental in bolstering the idea that the problem could be tackled without needing to reason directly about beliefs. As part of the project Abrial built on this earlier work and developed a very elegant analysis technique using B-refinement. We briefly outline this in section 9 and refer the interested reader to, [JRA1, JRA2] for more details.

In the final year of the project Michael Butler was funded to investigate the application of Action Systems and in particular the links between the B-method and Action Systems. A number of previous results of the project were reworked in the Action Systems formalism. The correspondence between CSP, Action Systems and the B-method potentially allows the integration or at least federation of techniques and tools developed under the project. See [MJB1, MJB2].

3 The Model

3.1 Architecture

In this section we outline the way the abstract system models are constructed in CSP. This will form the basis for the work described later on the use of FDR and the construction of proofs using rank functions.

Typical of the kind of architecture that we have in mind is that shown in the diagram below.

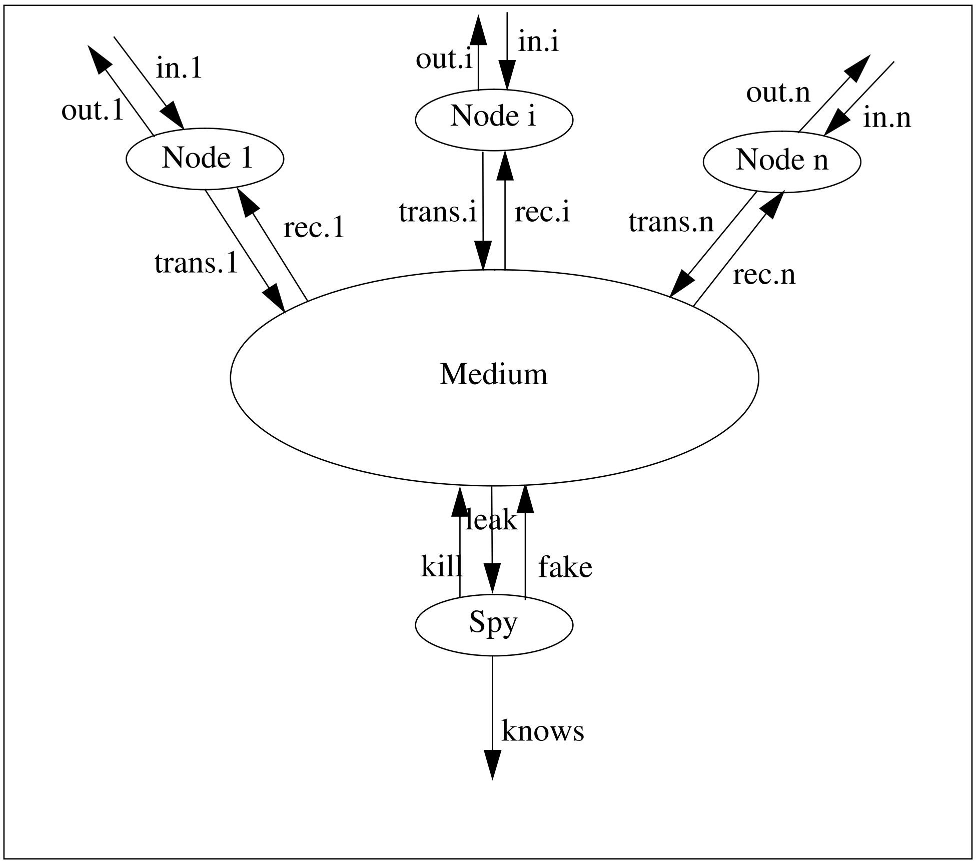

Figure 1. Typical Architecture

Each of the circles represents a CSP process. The user nodes are linked to the medium. Each user node has an *in* and an *out* channel to communicate in the clear with its user. Thus for example user i might pass a message m to its node with a request that it be send to user j. The node then performs all the key establishment and encryption as required to achieve this in a secure manner and transparently to the users.

We assume an open-addressing scheme in which messages passed to the medium have addressing information in clear allowing the medium to pass messages on to the appropriate destination. We thus assume that messages will have the form:

rec.i.j.m or trans.i.j.m

depending on whether they are entering or exiting the medium. Here, i denotes the sender, j the recipient and m the message contents. Of course, these address fields can be tampered with by the spy.

Note also the presence of a spy node equipped with *leak*, *fake* and *kill* channels to the medium and a *knows* channel to the outside.

3.2 Messages

The data-type of messages is in general quite elaborate. We assume a number of basic types: Keys, Nonces, Identifiers, Plaintext and possibly others. Messages are built up from these using concatenation, encryption and hashing etc. More formally, in BNF notation we have:

a:= Keys | Nonces | Plaintext |.....

m ::= a | m.m | $\{m\}_k$ | hash(m) | ....

We typically assume that everything is strongly typed: that nodes can distinguish between keys, nonces, plaintext, ciphertext etc. This assumption is rather delicate and may not always be valid. This issue and its possible consequences are discussed further in section 7. This assumption can be relaxed as appropriate.

3.3 Medium

The default behaviour of the medium is to deliver messages correctly to their destination, by which we mean that when it receives a message rec.i.j.m from the input channel of node i it will duly output trans.i.j.m to the input channel of node j. However a number of other behaviours are possible that can be thought of as being controlled by the spy:

- It may kill a message.
- It may leak a message, i.e. deliver it correctly but also deliver it to the spy.
- It may kill and leak a message, i.e. deliver it only to the spy.
- It may accept a message from the spy and deliver it as specified by the spy.
- It may reroute a message.

These are not all independent but we list them all to make clear the full capability. It may be appropriate in some applications to modify the spy's capabilities. For example it may not be possible for the spy to kill messages if for example we are dealing with a protocol for communications broadcast over the airwaves.

3.4 Nodes

As indicated in figure 1, the nodes are thought of as having external inputs and outputs as well as input/output channels to the medium. Data is transmitted in clear to and from the users over the external channels. The nodes perform the various encryptions, signing etc. on the data in accordance with the protocol before transmitting to the medium. Similarly they perform the appropriate decryption and checks on messages input to them from the medium before transmitting the clear text to the external user.

Their coding in CSP therefore follows straightforwardly from the protocol description except that it is often necessary to make certain design decisions: we need to be explicit what checks need to be performed and in dealing with the exception behaviour, e.g. what happens when a node receives a message that does not correspond to what it is expecting at that stage of the protocol or a check fails. We also have to be explicit about whether a node can participate in or initiate multiple runs of the protocol simultaneously. Such details are typically omitted from protocol descriptions which just give the intended correct sequence of events of a single run.

3.5 Servers

Not all protocols require a server. For those that do the server's role is usually fairly straightforward: typically it will do little more than accept a request message and respond with a message of the appropriate form. Again its coding as a CSP process is straightforward given the protocol description.

3.6 Spy

The spy is far and away the most fascinating character in this game. His aim is to subvert the protocol. Exactly what powers we imbue him with can be varied according to circumstance but typically we will perform a worst-case analysis: we might allow him to tap, intercept, reroute or fake any message at will, limited only by cryptographic considerations. In some models we may even allow him to acquire keys in certain circumstances to mimic the possibility of keys being broken after some time or otherwise compromised.

Note that in some analyses we may want to allow the spy, in addition to the capabilities described above, the status and capabilities of a legitimate user. In particular we may give him a recognised identity and corresponding keys etc. The server and other nodes thus regard him as a legitimate player and will communicate with him accordingly. We see this in Lowe's attack on the Needham-Schroeder Public-Key protocol for example. [Lowe].

From an abstract point of view we can include as many spies as we want. In some cases it can be shown that we can restrict the model to a single spy process without loss of generality: it can be shown that 1 spy is equivalent to n spies. Occasionally it may be necessary to include more than one if we need to consider the possibility of collusion attacks. This might occur if for example we are considering a protocol designed to enforce an n-man rule. Even here it is often sufficient to give a single spy access to multiple identities.

The spy will be provided with certain initial knowledge and will subsequently be able to deduce further knowledge from traffic that he observes through the medium. The spy is equipped with a simple inference system that governs what he can deduce from given information. So, for example, if he sees an encrypted message he can extract the plaintext if and only if he also possesses the corresponding decryption key. Given some text he can compute a crypto-hash of the text but not vice versa.

The spy will be supplied with some initial information and will typically be assumed to overhear all traffic. From this he will be able to make various deductions. If amongst these deductions he is able to hit on some sensitive item he will signal this to the outside. Besides passively overhearing traffic he can further construct fake messages from the information he can deduce and try to use these to trick the other principals. he can thus mount interleaving, replay, oracle attacks but note that such strategies are not coded explicitly but are emergent. In this respect the spy is more like Deep Blue than Kasparov: he tries everything until something works.

Where appropriate we may include algebraic identities in the spy's inference system. Some crypto-algorithms are commutative for example and this fact may be exploitable by the spy. For some protocols such facts may actually be vital to allow the protocol to function, Diffie-Hellman key exchange depends on the commutativity of exponentiation in a finite field for example.

The spy has three channels over which he can interact with the medium to tap, kill or fake messages. He also has a knows channel to the outside over which he can announce his success. This is of course not typical spy behaviour but is a modelling device to allow us to know when a breach has occurred.

4 Formalisation of the Security Properties

In order to be able to demonstrate that a particular protocol ensures a particular security requirement we need a precise characterisation of both the system in which the protocol is implemented (including hostile agents) and the security property in question. In the previous section we indicated how a system model is constructed in CSP, in this we turn our attention to the security properties. We have already alluded to some of these: confidentiality, authentication etc. The meaning of these terms is usually assumed to be intuitively clear but trying to pin down such intuitions in a precise mathematical form can often be surprisingly tricky and unexpected subtleties often show up.

In the following subsections we outline possible formalisations of these properties in CSP. These should be regarded as indicative rather than definitive. They serve to show that such properties can be simply and intuitively formalised in CSP. In practice we may not use precisely the formalisation given here for reasons of efficiency etc. More detail can be found in [SAS].

4.1 Confidentiality

Confidentiality has a number of different possible meanings and which is appropriate will depend on the application. We might have a very strict interpretation in which the spy should not be able to deduce anything about the legitimate users' activity, i.e. he should not even be able to perform any traffic analysis on who is talking to whom, at what volume of traffic etc. let alone derive the content of any messages. For this a non-interference style characterisation is required, and implementations would require hot-line style dummy traffic etc. This tends to be too strong a characterisation for most applications and in most cases we find simpler, safety style formalisations are sufficient. These are both simpler to understand and to check and also seem more appropriate to most applications so we will confine ourselves to these in this discussion.

Other approaches do utilise ideas of process equivalence to code non-interference style properties for security protocols, for example Focardi and Gorrieri using a CCS based approach and Abadi and Gordon using the spi-calculus.

More typically we are just concerned to prevent the spy from being able to deduce the plaintext of messages passing between the nodes. This has the advantage that it can be formulated as a safety style property, namely that certain undesirable events or behaviours do not occur. This means that it is sufficient to use the traces model of CSP which makes the model-checking more efficient.

Suppose that there is some set M of sensitive messages that the users can transmit to each other. We want to ensure that no message from this set can be deduced by the spy. We can now formalise confidentially as the requirement that no event of the form:

$$\text{knows!m} \tag{EQ 1}$$

with m in M can occur. This can expressed as a simple predicate over traces or alternatively, and more conveniently for FDR:

$$\text{SYSTEM refines SYSTEM}\|_{\text{knows!.M}}\text{STOP} \tag{EQ 2}$$

The effect of composing STOP in parallel with another process P over an alphabet A is to prevent P from performing any events from A. Thus if SYSTEM can perform any event of the form knows!M the refinement will be violated. Such a refinement can be presented to FDR for verification.

Note that the spy may of course see messages of the form $\{m\}_K$ but as long as he is not able to get hold of the corresponding K he is still unable to deduce m and so the system is still deemed secure.

4.2 Authentication of origin

Intuitively authentication of origin is taken to mean that we can be sure that a message that purports to be from a certain party was indeed originated by that party. In terms of our model we can phrase this as:

if a message of the form out.i.j.m occurs if must have been preceded by an message of the form in.i.j.m.

Thus: if j receives a message m that appears to have come from i then it should indeed be the case that i did send out that message for j.

Again we can formulate this neatly in CSP:

$$SYS\|_{a}STOP \text{ refines } SYS\|_{a,b}STOP \quad \text{(EQ 3)}$$

where a= in.i.j.m and b = out.i.j.m.

This asserts that stopping just in.i.j.m events is the same as stopping in.i.j.m and out.i.jm events. If the system could perform an out.i.j.m. without first having performed in.i.jm. then the above equation would be violated and indeed this corresponds to a failure of authentication.

4.3 Integrity

Integrity is usually taken to mean that data cannot be corrupted or at least that any such corruption will always be detected. In this sense integrity is seen as a corollary of our authentication property above. This follows from the fact that in this formulation we required that the *m* of the output message match that of the input message. If for some reason we're only interested in the originator and not the content then we could allow the message content to differ:

$$SYS\|_{a}STOP \text{ refines } SYS\|_{a,b}STOP \quad \text{(EQ 4)}$$

where a= in.i.j.m and b = out.i.j.n.

It is not clear that this is a terribly useful property but might be useful if you're primarily just interested in knowing that i exists and is prepared to talk to you. This could perhaps be thought of as a form of entity authentication. A detailed discussion of a hierarchy of possible notions of entity authentication is given by Lowe, [Lowe3].

4.4 Key-exchange/distribution

We will want protocols to keep keys confidential to their intended users. This is related to our confidentiality property above but differs slightly: note that it is quite possible for a protocol to keep keys confidential and yet leak plaintext if plaintext is transmitted in clear.

Note that if we have included a mole in our model then of course he will get to see issued keys quite "legitimately", i.e keys intended for him to communicate with another user. What we need to establish here is that he never gets to know any keys that have been accepted as "good" for communication between any other two nodes. If our spy is an outsider on the other hand it may be simpler and more appropriate to require that he never gets to know any issued keys. These can be coded as special cases of confidentiality.

4.5 Non-repudiation

Non-repudiation remains a hotly debated notion but it does seem generally agreed that in essence the idea is to provide the various parties with evidence that certain steps of the protocol have occurred. For example, non-repudiation of transmission should provide the recipient with proof that the message was indeed sent by the claimed sender. Furthermore this proof must be convincing to a third party, not just to the recipient himself.

A typical scenario might be: A sends a command to B. B executes the command but it all goes horribly wrong. A later claims that he sent no such command. B would like to be able to convince the court martial that A did indeed issue the command. For this he needs some piece of (electronic) evidence and it is vital that this could not have been fabricated by him (or more precisely that it can be argued that it would have been infeasible for him to have fabricated it).

As such we see that non-repudiation appears to be very similar to authentication but with the additional requirement that the recipient of an output message that is to serve as proof should not himself be capable of creating the message. This suggests that we may be able to capture this property as an variant of authentication but ensuring that our model endows the agents with spy-like capabilities. That is the participants are given capabilities to fake messages up to the usual cryptological constraints. If we can then shown that Bob could only have come into possession of a message of certain form if Alice had actually send an appropriately related message to him then we will have shown the non-repudiation property.

In essence this is how we proceed but it should be noted that there remain certain issues arising from the fact that there is almost inevitably an asymmetry between the players: one gets ahead of the other in terms of the evidence available as the protocol unfolds. This is analogous to the issues of commitment points in a key-exchange protocol.

4.6 Anonymity

Anonymity is a property that seems to have been very little explored from a formal point of view. Intuitively a system that is anonymous over some set of events A should have the property that an observer, though he may be able to deduce that an event from A has occurred he will be unable to identify which. Of course this will in general depend on the status of the observer so we must be clear about the observer viewpoint in formalising this property.

A possible formalisation, ref [SA], is the following:

A process P with events from A hidden. It is strongly anonymous over A iff

$$f^{-1}{}_A(f_A(P)) = P \qquad \text{(EQ 5)}$$

where f_A is the renaming function defined by:

$$f_A(x) = e, \text{ if } x \in A \qquad \text{(EQ 6)}$$

$$f_A(x) = x, \text{ otherwise.} \qquad \text{(EQ 7)}$$

In other words f maps every element of A to a single event e say (which is not an element of the alphabet of P). All other events are left unchanged.

A simple example of a process that, from the point of view of an observer who can only see "c" events, is clearly anonymous over the set {a,b} is:

$$P = a \rightarrow c \rightarrow STOP \; [] \; b \rightarrow c \rightarrow STOP \qquad \text{(EQ 8)}$$

Use f defined by;

$$f(a) = e, f(b) = e \text{ and } f(c) = c \qquad \text{(EQ 9)}$$

then

$$f(P) = e \rightarrow c \rightarrow STOP \qquad \text{(EQ 10)}$$

and now:

$$f^{-1}(f(P)) = f^{-1}(e \rightarrow c \rightarrow STOP) = a \rightarrow c \rightarrow STOP \; [] \; b \rightarrow c \rightarrow STOP = P \qquad \text{(EQ 11)}$$

as required. One can think of the $f^{-1}{}_A f_A$ operation as shuffling the events from A.

Note that the operator $f^{-1}f$ is idempotent and when applied to a process yields that is anonymous.

It is worth noting that this formulation of anonymity could have ramifications for notions of confidentiality (and vice-versa). Strong forms of confidentiality require that high level activity has no observable effects on low level observers. This is typically formalised as some flavour of non-interference. In some situations, and the analysis of crypto-protocols is a case in point, we may be happy with a weaker form: we are prepared to tolerate some, limited, information about the high level users' activity to be observable at the low level. Thus we may be prepared to allow the spy to deduce that a message (of length l) has passed from A to B but we want to be pretty certain he can't figure out exactly which particular message of the appropriate message space it is. This can clearly be formulated as an anonymity style property as described above. Here however the anonymity is over the message space rather than the agent space.

Another example is the downgrading of statistical information from a highly classified data-base. This can formulated by constructing suitable equivalence classes over the space of possible data sets and then requiring anonymity over each equivalence class.

In the above description we have assumed that the anonymous events are concealed from the observer using the hiding operator. We are also investigating the effect of using alternative forms of abstraction such as the "lazy" abstraction: camouflaging events by interleaving them with RUN and combinations of these. This gives rise to different flavours of anonymity.

A number of examples of systems displaying various degrees of anonymity such as Chaum's "dining cryptographers" have been analysed using FDR, reported in ref[SA].

Notice that this approach is essentially "possibilistic" in style, that is to say we have abstracted away from issues of probability and considered only whether or not certain events are possible. Others have taken a probabilistic style, for example Pfitzmann et al. Here the idea is basically to show that the a postiori probability distribution over some event set after some observation is equal to the a priori distribution. Using probabilistic CSP the two approaches could doubtless be related. This is similar to the use of probabilistic vs possibilistic formulations of non-interference in the presence of non-determinism.

4.7 Weak anonymity and constrained properties

The discussion of anonymity above assumed that anonymity should apply to each occurrence of certain events, in effect that the events from the anonymity set could be freshly shuffled on each occurrence. In some situations we might not expect the occurrence of events from the set to be wholly independent. Voting provides a good example: we do not want a particular vote to be associated with an individual but on the other hand we want to prevent double voting. The events are thus not independent but are subject to the constraint that for a given run each event should occur at most once.

This and other similar conditional forms of anonymity are most elegantly formalised as constrained forms of the strong anonymity described above. Thus for the voting example we define a process OneVote[A] that allows each voter drawn from some set A to vote at most once. We then compose this in parallel with the system and require that the result satisfy strong anonymity:

$$\text{OneVote}[A] = \text{if } A = \{0\} \text{ Stop else } [\,]\ a{:}A \text{ -> OneVote}[A-\{a\}] \qquad \text{(EQ 12)}$$

$$f_A^{-1}(f(\text{System}||\text{OneVote}) = \text{System}||\text{OneVote} \qquad \text{(EQ 13)}$$

Notice that this style of weakening a property by putting a constraining process in parallel with the system can be used for properties other than anonymity. Thus, for example, weakened forms of non-interference can be coded neatly in this way in situations in which we are concerned only about leaking information regarding certain sensitive behaviours.

4.8 Key recovery

The tension between the rights of citizens and companies to use strong encryption and the requirements of law enforcement and security services to access communications has led to proposals for key recovery (escrow). The idea is to allow the use of strong encryption algorithms but enforce protocols that ensure that any session key is available to the appropriate authorities. Such protocols should therefore provide an additional property that a session key cannot be established between A and B without it also being known to the third party. This is clearly just a flavour of authentication.

We may require elaborations of this: for example we may want some form of n-man rule to control the access to the third party knowledge of the keys. Again such requirements can be coded straightforwardly, in most cases again as flavours of authentication: that the release of the key must be preceded by the appropriate warrant events. We see that a large class of properties in fact turn out to be essentially authentication style properties: that certain events can only occur if preceded by other events.

We note in passing that the converse style of property: that if certain events occur they must be followed by others are a little more delicate to code and will typically need the infinite traces semantics. However it is often the case that we actually want to state that the follow-on events occur within a certain time, e.g. alarm events. In this case of course we need a timed model but can avoid the complications of infinite traces.

4.9 Availability

Thus far we have discussed the rather negative aspects of analysing protocols: establishing that they do not allow certain undesirable behaviours. It is clearly also of importance to be able to establish that, under appropriate assumptions, the protocol will achieve some desired effect. Thus, for example, for a key exchange protocol, we would like to establish that the session can indeed be established. Of course in order to do this we will need to curtail the spy capabilities to some extent, in particular we cannot allow him unlimited ability to kill messages.

There is nothing explicit in the model so far to signal success so we need to introduce explicit events to indicate this. We do this by introducing a special channel "done" over which a node i can transmit the message:

$$done!i.j.k_{ij}$$

to indicate that it is now willing and able to talk to j using key k_{ij}. The done channel is of course purely a modelling device to give us a God's eye view and the spy, in particular, will have no access to it.

Thus we can specify success by the process:

$$\begin{aligned} SUCCESS := req?a?b \rightarrow (\sqcap k: FreshKey \bullet (done!a.b.k \rightarrow done!b.a.k \rightarrow \\ STOP \sqcap done!b.a.k \rightarrow done!a.b.k \rightarrow STOP)) \end{aligned} \qquad \text{(EQ 14)}$$

where $\sqcap$ denotes the internal choice operator.

Thus after a request from a to communicate with b, a new key should be chosen followed by two done events one from each of a and b. It is not important what order these done events occur in and so the inner choice is simply to allow these two possible orderings. In practice we need to elaborate this somewhat to allow for certain other contingencies such as one node backing out, but this conveys the essence.

We now require that an appropriate abstraction of our system (with all channels except the *req* and *done* channels hidden) refines the above specification. Here we must perform a failures divergence refinement check as we are concerned with a liveness property:

$$\text{SYSTEM}\backslash\{\text{Sigma-}\{\text{req, done}\}\}\ \text{refines}_{\text{Failures}}\ \text{SUCCESS} \qquad \text{(EQ 15)}$$

Our model as described earlier will of course inevitably fail because the spy can do anything after the request message. In order to get any interesting results we must limit the spy's behaviour. Typically we block his activity after some non-deterministically chosen point and check that the protocol recovers within some finite number of steps.

For key-exchange protocols we might be able to perform slightly more elaborate analyses but it is not clear that we can do much better than this. For example if we allowed the spy just to kill one message per protocol run this would be sufficient to block it, certainly as it is currently specified and it is hard to see how this could be avoided.

On the other hand if we are in the situation in which a session key is already in place we can hope to establish stronger results. In this case we might hope to be able to develop a protocol that could function in the presence of a spy who could kill some proportion of the messages but not all. For example, a cryptographically protected alternating bit protocol could achieve success in such a situation.

4.10 Intensional specifications

The specifications of security properties given above are all quite generic, i.e independent of the details of the any particular protocol. This is useful in that a single specification can be used to analyse a number of different protocols. In some situations it appears to be useful to employ a more tailored specification for a particular protocol. In essence, such a specification states the sequence of message transmissions and receives at each node that the protocol designer intends to reach a final success state, [AWR2]. The analysis will now show if one of the nodes can reach a commit state without the expected sequence being followed. Such a possibility is not necessarily a vulnerability but may indicate a security weakness. In any case it will point out some unintended behaviour which could prove significant.

This approach is not proposed as an alternative to the generic specifications but as a complementary approach that can potentially flag a class of vulnerabilities that might be missed by the generic approach. It is of course true that a new intensional specification is required for each new protocol, but on the other hand it can be very easily, and potentially even automatically, derived from the protocol description. Such a style of analysis is applicable in domains other than security protocols or indeed security. Many safety-critical systems are also amenable to such analysis.

5 FDR Coding

The framework described in section 3 describes how the abstract system models and security properties can be formalised in CSP. We now turn our attention to how such models can be analysed. In this section we consider the model-checking approach, exploiting the CSP refinement checker FDR. The models described in the previous section are not suitable for model-checking. The data-types are at the very least extremely large (e.g. key spaces) or in some cases infinite (the space of possible messages). In order to perform automatic analysis using the FDR model-checker we need to incorporate certain simplifications and coding tricks to get around the limitations on size of state spaces whilst as far as possible retaining the completeness of the models.

A proof theoretic approach such as that developed by Schneider will not be constrained by state space size though simplifications in the models may still be necessary to keep the proofs tractable.

Data independence style arguments allow us to use key and nonce spaces with only a few elements. For example the exact value of a key is not relevant, only whether or not it is equal to some other key value. It may thus be possible to get away with just two key values in order to represent the situations of equality or non-equality of the values.

We can argue further, that it is sufficient to consider encryption depths of no more than two say (in the case of Needham-Schroeder for example) in order to keep the message spaces manageable. Here the argument would be that generating messages with deeper nestings would not give the spy any added advantage. We might limit the spy to generating messages of the correct form for the protocol on the grounds that anything else would be ignored by the nodes. Such arguments are rather more delicate than the data independence arguments above and must be carefully examined in the case of a particular protocol. Examples can be constructed in which the spy does launch attacks that require encryption depths greater than those of the protocol itself, but these tend to be rather contrived. Any assumptions about strong typing also need to be carefully examined: a nonce could easily be confused with a key or ciphertext or indeed nested ciphertext and so on.

Other arguments allow us to use models with only say two nodes and a single spy, i.e. that having more than one spy acting in collusion does not give rise to any attacks not available to the lone spy. Sometimes it is possible to limit the number of runs that can interleave as was done by Lowe in his analysis of the fixed Needham-Schroeder Public-Key protocol.

Coding tricks for the spy, for example the coding of the algebraic relations using CSP renaming, are described by Goldsmith and Roscoe in this proceedings.

6 Introducing Time

Up to now we have made no explicit mention of time in our models. For many purposes this is fine but if we want to reason about timeliness, freshness, lifetimes of keys etc. we need to introduce time into our framework. A theory of Timed CSP has been developed, see for example [DS], that deals with a continuous form of time. This is too rich a model for our purposes and in any case would not be amenable to model-checking because it involves modelling continuous time. We will therefore adopt a discrete version of time: the so-called “tock dialect”. A special tock event is introduced on which processes synchronise.

For model-checking purposes we need to keep the state-space small. One way to do this is to keep the time dimension small, maybe with just two elements: {timely, tardy}. Timestamped events then age when a tock event occurs. Such a model, actually with a slightly larger time dimension, is sufficient for example to recreate the ping-pong attack on the Wide-Mouthed Frog protocol.

To cause timestamps to age in the lazy spy's knowledge, an additional “aging deduction” is added to the set of deductions it was possible for the spy to make. Given a timestamped message, which was known to the spy, on the occurrence of a tock event the spy “forgets” this message and “learns” the corresponding message with an older timestamp. Eventually the timestamps reach the bottom age value and age no further.

This analysis assumes that a reliable and trusted global clock exists over the network, i.e that we can rely on the clocks of the various nodes to be close to synchronous. It is possible to model the possibility of clock skews but in practice this greatly increases the state-space size. In some cases it must further be assumed that some trusted mechanism is in place to ensure that timestamps can’t be faked: i.e. a false time inserted. In practice this is not easy to achieve and any such assumption would need to be closely examined.

In this respect nonces have a distinct advantage over timestamps but on the other hand timestamps do provide additional capabilities. Deciding between them can thus involve a delicate trade-off. This is a area that is still poorly understood in the security community and merits further study.

7 Role and limits of formal analysis

Formal techniques inevitably make simplifying assumptions about the systems in question. For example, in the context of security protocols, we make certain assumptions about the observations and deductions that the spy can make: e.g. that he can only observe messages in the medium, that he only has certain algebraic identities to manipulate terms, that fields of one type cannot be mistaken for fields of another, that fields are of particular block lengths etc. Work at York University has concentrated on identifying areas which need to be borne in mind when abstract specifications are refined to working code. [CJ1,CJ2].

It is quite possible to implement a perfectly secure protocol using high grade crypto-algorithms in such a way that the result is hopelessly flawed. We have recently come across a very real example of exactly this: an implementation proposed by APM of a protocol that had been verified by Paulson [Paul]. When this was studied by Steve Schneider and one of the authors (PYAR) it was found to have a glaring vulnerability. Paulson’s analysis was perfectly valid but the way APM proposed to implement the encryption give rise to some additional algebraic identities not included in the model. It was these identities that the spy could exploit. This is documented in [RS].

This illustrates the point that our models are, like all mathematical models, only ever approximations to reality. We are usually assuming that the spy's only source of information is through monitoring traffic. In reality he may have other sources such as EM radiation or timing information. It seems best to deal with such vulnerabilities separately to keep the models tractable. We must bear in mind that the validity of our analysis will always be relative to the threat model and to the faithfulness of our system model so proofs of security will never be absolute.

A promising avenue to pursue is that of combining the best features of both model-checking and theorem proving and this will be the subject of future work.

8 CASPER

The construction of the FDR code corresponding to a given protocol requires specialist skills. However it quickly became apparent that the production of such code from the description of the protocol is largely systematic. Based on this observation Lowe, then an RA working with Roscoe at the PRG, developed a compiler to take a description of the protocol and of the security property and automatically generate the corresponding FDR code. Descriptions of this can be found for example in [Lowe2] and Lowe's web page: http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html.

In practice this is not entirely straightforward:

- the standard notation for describing protocols is notoriously ambiguous.
- Security protocols differ widely in their security goals and the mechanisms they employ. This means that it is difficult to have a general purpose compiler able to handle them all.

Lowe introduces an enriched notation to describe the protocols that resolves the ambiguities and makes explicit certain aspects of the protocols. This has much in common with the CAPSL notation proposed originally by Millen as a standard notation for describing security protocols.

CASPER has considerable theoretical as well as practical significance as it addresses the issue of ensuring a clear correspondence between the formal framework and a notation accessible to the protocol designers and implementers. The process of analysis is thus much more efficient and less error prone.

CASPER continues to be developed to handle a wider class of protocols and security properties: timing properties, algebraic rewriting etc.

9 The B-method

The B-method is another promising formal method developed by J-R Abrial, [JRA1]. It too has a suite of powerful tools and has been used extensively on real, industrial applications like the verification of the control system for the new driverless trains in the Paris Metro. It differs from CSP in being more state-based whilst being less suited to dealing with concurrency. This made it an interesting alternative to CSP: the lack of direct access to state-space in CSP sometimes makes coding a little awkward. On the other hand CSP's easy handling of concurrency is very useful.

As part of the project Abrial has developed a refinement technique to verify security protocols, reported in refs[JRA2. JRA3]. The key idea is to specify the objective of the protocol as a single instantaneous operation along with a suitable number of skip (void) steps. One by one the skip events are refined into actual steps of the protocol until the entire protocol is reached. Using the transitivity of refinement, as long as each of the refinement steps has been checked, then the entire development is verified. In effect time and space are gradually unfolded from the initial single, instantaneous event.

A further development of this technique allows the approach to work even where the number of steps of the final protocol is not known in advance. This may prove particularly useful for recursive protocols such as APM's proposed authentication protocol.

10 Action Systems

Action Systems, like the B-method, are essentially state-based and indeed the two are closely related. At the same time a precise correspondence had already been established between Action Systems and CSP, initially by Carroll Morgan [CCM] and further developed by Butler in his DPhil, [MJB1].

This gives rise to the possibility of taking the best features of all three techniques and producing a powerful synthesis. The idea is not to try to produce a single combined framework but to be able to move smoothly between the frameworks as convenient. Thus for example, Action Systems look promising as a framework for a notation for describing the protocols that would be more familiar to protocol engineers than CSP. One might thus start with an Action System description and translate this to CSP in order to perform model-checking. In fact Lowe's CASPER effectively serves this role but it would still be interesting to understand the notation of CASPER in terms of Action Systems or AMN (Abstract Machine Notation, the formal notation of the B-method).

Abrial's refinement technique has been reworked by Butler in Action Systems as have several of the formalisations of security properties and protocols, [MJB2]. This work has shown that such a synthesis is both possible and useful and indeed it will have repercussions beyond the area of security protocols.

11 Conclusions

CSP has been shown to be a very effective framework in which to formulate the behaviour and properties of security protocols. Furthermore the model-checking technique based on FDR has be shown to be very effective in performing automated analysis of protocols.

The effectiveness of this approach has been demonstrated by the uncovering of several new attacks on well established protocols. Its utility as design tool is also apparent, as exemplified by its use to progressively strengthen the TMN protocol, [LR]. Here a weak version of the protocol was progressively strengthened in the light of vulnerabilities revealed by the analysis until a version was arrived at that stood up to analysis. Alternatively an over-engineered design can be progressively "weakened" to probe for redundancies.

More recently we have introduced discrete time models to address issues of freshness, lifetimes of keys etc. This has been used to find Denning-Sacco style attacks on Needham-Schroeder as well as to establish various liveness properties.

12 Further Work

The model-checking approach does of course run into problems with the limitations on the size of the state spaces that can be handled. The technology is steadily improving so this envelope of tractable models is steadily growing. With ingenuity these limitations can worked around to a remarkable extent. Even so we have reached the stage at which to make major strides forward the approach will need to be complemented by other techniques. Principal amongst these are techniques for exploiting symmetries of the system. Here the recent work of Lazic and Roscoe, [Laz1, Laz2, LR], is highly relevant. Another angle is to incorporate theorem proving techniques. Theorem proving has already been used to a limited sense in justifying some of the simplifying assumptions used in the FDR codings. It would be interesting to more fully integrate a proof assistant with the model-checking tool. Symbolic normalisation techniques may prove relevant here.

It is also clear that this technology has application to areas other than security protocols. Those currently under investigation include:

- fault tolerant systems/algorithms
- distributed simulation protocols/algorithms
- secure architectures and components (gateways, sandboxes etc.)

Another growing area to which this technology has application is that of systems that combine safety and security mechanisms in a way that enhances both. An example being the authenticated byzantine generals algorithms.

An area that should be explored further is that of containment and resilience: assessing and preferably minimising the effect of a breach and ensuring that the system can be restored to a secure state. In practice breaches will occur and it is essential that systems be designed to be as resilient as possible in a security sense (this is analogous to the requirement for systems to be robust against malfunctions).

12.1 Acknowledgments

It is a pleasure to acknowledge the contributions of Bill Roscoe, Paul Gardiner, Michael Goldsmith, Dave Jackson, Gavin Lowe, Steve Schneider, Abraham Sidiropolos, Jean-Raymond Abrial, Michael Butler, Jeremy Jacob and John Clark. Indeed the credit for most of what is reported here lies with them.

We would also like to thank the DRA/MoD Strategic Research Programme for funding this work. Lowe's work was funded by the EPSRC.

13 References

[JRA1] J-R ABRIAL, The B Book: Assigning Programs to Meanings, CUP 1996.

[JRA2] J-R ABRIAL, Extending B without Changing it (for Developing Distributed Systems), November 1996, Proceedings of the 1st B Conference, Nantes.

[JRA3] J-R ABRIAL and I MUSSAT, Specification and Design of a Transmission Protocol by Successive Refinements Using B, to appear in the Proceedings of the Marktoberdorf Summer School (July 1996) Springer LNCS.

[MJB1] M J BUTLER, A CSP Approach to Action Systems, D.Phil. thesis, PRG Oxford University 1992.

[BAN] BURROWS, ABADI, NEEDHAM. *A Logic of Authentication.* Digital Systems Research Centre Report 39, 1989.

[BM] C. BOYD, W MAO. On a Limitation of BAN Logic. In *Lecture Notes in Computer Science,* Volume 765. Springer-Verlag 1993, pages 240-247.

[BS] J BRYANS and S A SCHNEIDER, CSP, PVS, and a Recursive Authentication Protocol, to be presented at DIMACS Workshop on Formal Verification of Security Protocols, Sept. 1997.

[MJB2] M J BUTLER, Action Systems and Security Protocols, April 1997, to appear as DRA Tech Report.

[MJB3] M J BUTLER, Stepwise Refinement of Communicating Systems, Science of Computer

Programming, 27(2): 139-173, September 1996.

[CJ1] J A CLARK and J L JACOB, Attacking Authentication Protocols, High Integrity Systems 1(5):465-474, August 1996.

[CJ2] J A CLARK, J L JACOB. On the Security of Recent Protocols. In *Information Processing Letters*, Volume 56, 1995, pages 151-155.

[CARH] C A R HOARE. *Communicating Sequential Processes*. Prentice Hall, 1985.

[DS] J DAVIES and S A SCHNEIDER, A Brief History of timed CSP, Theoretical Computer Science, 1995, volume 138.

[KMM] A. KEMMERER, C. MEADOWS, J. MILLEN. Three Systems for Cryptographic Protocol Analysis. In *Journal of Cryptography,* 1994.

[CCM] C C MORGAN. Of wp and CSP. In FEIJIN et al. *Beauty is our Business, A Birthday Salute to Edsger W Dijkstra*. Springer Verlag, 1990.

[Laz1] R S. LAZIC, A Semantic Study of Data-independence with Applications to Mechanical Verification f Concurrent System, Merton College, May 1997, A Dissertation Submitted for the Senior Mathematical Prize.

[Laz2] R.S. LAZIC, Theorems for mechanical verification of data-independent CSP, Oxford University Computing Laboratory technical report, 1997.

[LR] R.S. Lazic and A.W. Roscoe, A study of data-independence in CSP, Submitted for publication, 1997.

[Lowe1] G LOWE, Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In MARGARIA and STEFFEN, eds. *Tools and Algorithms for the Construction and Analysis of Systems,* Volume 1055 of *Lecture Notes in Computer Science*, Springer Verlag, 1996, pages 147-166.

[Lowe2] G LOWE, Casper: A Compiler for the Analysis of Security Protocols, IEEE Computer Security Foundations Workshop, 1997.

[Lowe3] G LOWE, A Hierarchy of Authentication Specifications, IEEE Computer Security Foundations Workshop, 1997

[LR] G LOWE, A W ROSCOE. Using CSP to Detect Errors in the TMN Protocol. To appear in *IEEE Transactions on Software Engineering.*

[LCP] L PAULSON, Mechanised proofs for a recursive Authentication Protocol, IEEE Proceedings of CSFW 1997.

[AWR1] A W ROSCOE. The Theory and Practice of Concurrency, Prentice Hall 1997 to appear.

[AWR2] A W ROSCOE. Modelling and Verifying key-exchange Protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, 1995, pages 98-108.

[GR] A W ROSCOE and M H GOLDSMITH, The Perfect ``Spy'' for Model-Checking Crypto-protocols, to be presented at DIMACS Workshop on Formal Verification of Security Protocols Sept. 1997.

[PYAR] P Y A RYAN. A CSP Formulation of Non-interference and Unwinding. Presented at the CSFW 1990 and published in *Cipher* Winter 1990/1.

[RS] P Y A RYAN and S A SCHNEIDER, An Attack on a Recursive Authentication Protocol, A Cautionary Tale, DRA report April 1997, submitted to Information Processing Letters.