

# Intensional specifications of security protocols

A.W. Roscoe

Oxford University Computing Laboratory  
Wolfson Building, Parks Road, Oxford OX1 3QD, UK\*

April 21, 1998

## Abstract

It is often difficult to specify exactly what a security protocol is intended to achieve, and there are many examples of attacks on protocols which have been proved to satisfy the ‘wrong’, or too weak a specification. Contrary to the usual approach of attempting to capture what it is that a protocol achieves in abstract terms, we propose a readily automatable style of specification which simply asserts that a node can only complete its part in a protocol run if the pattern of messages anticipated by the designer has occurred. While this *intensional* style of specification does not replace more abstract ones such as confidentiality, it does appear to preclude a wide range of the styles of attack that are hardest to exclude by other means.

## 1 Introduction

Over the past two years, the author and associates in Oxford have been investigating the modelling of security properties and cryptographic protocols in CSP, with particular reference to testing and verification on the model-checker FDR<sup>1</sup> [5, 6, 10, 8, 9]. Work on protocols has fallen into two styles: safety and liveness (no loss of service).

The overall approach taken has always been to infer the behaviour of individual nodes from the protocol description and implement these as processes in a CSP network. This network typically consists of a pair of trustworthy nodes, a server if necessary, and a ‘spy’ process with two functions:

- to provide the behaviours of other nodes in the system, which are thereby assumed untrustworthy but may, of course, behave properly, and

---

\*Email: Bill.Roscoe@comlab.ox.ac.uk Copyright 1996 IEEE. Published in *Proceedings, 1996 IEEE Computer Security Foundations Workshop*, Kenmare, Eire, June, 1996.

<sup>1</sup>FDR is a product of Formal Systems (Europe) Ltd.

- to interfere with communications between the trustworthy nodes by over-hearing them, taking them from the communication medium so that they do not reach their destinations, and faking messages (say introducing a message that purports to be from  $A$  to  $B$  that the medium will deliver to  $B$ ).

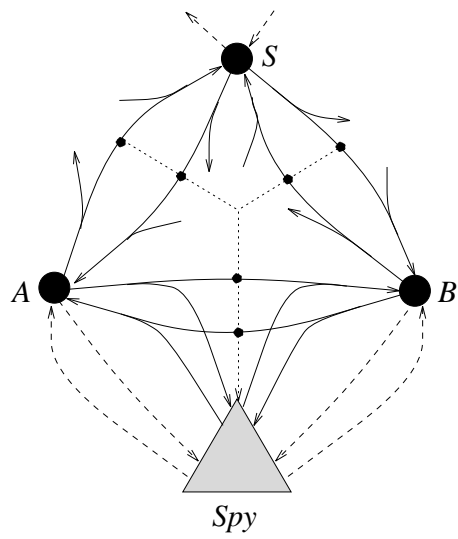
The spy is allowed to perform any of these actions at any time, subject only to the limitation that the messages it introduces must be feasible to produce from its current state of knowledge (initial and what it has learned since) under the rules of the cryptographic systems in use. The resulting network is shown in Figure 1.

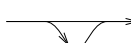
In *coding* protocols in such a way that liveness analysis makes sense, we have had to address issues not usually covered in academic papers on this subject, such as the handling of exception conditions and the synchronisation of commitment between nodes. This inevitably leads to significantly more involved node descriptions, which both involves extra effort and leads to proofs of correctness (where possible) of protocols that are significantly refined beyond the ‘standard’ versions.

On the other hand, establishing the right *specifications* of systems designed to meet liveness properties tend to be unproblematic. For a typical protocol that is designed to set up a key for use in a session between a pair of nodes, the specification comes in two parts. The first is that whenever either  $A$  or  $B$  seek a session with the other then – provided the spy does not perform infinitely many actions – they reach a state where they believe a session exists between them, agreeing on the key, and that neither can ever reach this belief otherwise. The second is that these keys never become known to the spy. The first of these properties encompasses both the no-loss-of-service specification and that of authentication (a safety property), while the second represents the safety property of confidentiality.

While any actual implementation of a protocol is likely to contain features designed to ensure either transparent (from the external users’ point of view) no-loss-of-service or some other way of handling and recovering from interference, it remains true that the published descriptions of these protocols do not generally encompass this. Thus to verify one of these protocols on its own terms we must set up a CSP network describing the protocol as described and, since this will certainly not meet no-loss-of-service specifications, seek only to prove safety specifications.

The coding of the network is generally substantially easier, though one is still forced to make some decisions such as what nodes do when a run is abandoned (the obvious decision being that they back-off to their initial states). What one quickly discovers, however, is that it tends to be harder to find the right specifications, particularly in areas related to authentication. The problem here is that it is inevitable that sometimes one participant in a run will believe it has completed successfully while the other will believe it has been abandoned.



- Connections with "imaginary" nodes
- ..... Spy overhearing normal communications
-  Channel where communications can be killed or faked by the spy

*Note that not all connections of channels to the spy are shown*

Figure 1: Network for testing protocols

This is because the spy can always intercept the last message of the run. Nevertheless a wide variety of published attacks on protocols relate to failures of authentication so we do need an agreed way of specifying it.

In this paper we examine the modes of specification for authentication-like properties that can be used for protocols implemented in CSP as described above. I will concentrate on the distinction between *extensional* specifications, which seek to establish what is achieved by the protocol without making detailed analyses of the actual communications nodes have sent, and *intensional* specifications which concentrate on checking that the designer's expectations about these communications are justified.

My main purpose is to introduce the *canonical* intensional specifications of a protocol, which can be derived more-or-less automatically from its description as a series of messages. While this style was motivated by the work on CSP, and is readily automatable in that context, it can certainly be understood without any knowledge of CSP and has potential both for automation in other tools and for use in judging and classifying attacks found by other means.

## 2 Extensional specifications

We classify a specification as *extensional* when it is independent of the details of the protocol and would apply to any other protocol designed to achieve the same effect. Thus, inevitably, it cannot mention the actual messages passing between nodes during a protocol since these vary a great deal from one to another. Instead it will test the states of mind (knowledge, belief, etc.) of the various participants including the spy.

To make such a specification you clearly have to understand what you are intending to achieve with the protocol. The properties proved then depend on whether the specifications capture the right thing. A good example of what can be achieved and what can be missed is provided by analysis of the TMN protocol (which is well-known to suffer from flaws and therefore provides a good test-bed for analytic techniques).

The TMN protocol [12] is described by the following series of messages:

1.  $A \rightarrow S : B, e_P(pks, ra)$
2.  $S \rightarrow B : A.req$
3.  $B \rightarrow S : e_P(pks, rb)$
4.  $S \rightarrow A : v(ra, rb)$

where  $e_P(k, m)$  is the public-key encryption of  $m$  under key  $k$  (the only such key being  $pks$ , the public key of the server  $S$ ) and  $v(ra, rb)$  denotes Vernam encryption (bit-wise exclusive or).  $ra$  and  $rb$  are random objects generated by

$A$  and  $B$  respectively. The purpose of this protocol is to establish a session between  $A$  and  $B$  using  $rb$  as the key.

This, and a series of strengthened versions, were studied by the author using FDR using a single extensional safety specification described as follows.

He programmed nodes so that they send each other messages using the keys that have been exchanged, and making the contents of messages depend on whom the node thinks it is connected with. Specifically  $A$  and  $B$  send each other initially secret messages when they believe they are connected with each other, so we can identify an error if either the spy learns one of these secrets (a failure of confidentiality) or if  $A$  or  $B$  receives a message that fails to tie up with the node it believes it is connected to (a failure of authentication). One should regard these special messages as symbolic objects allowing us, god-like, to examine system behaviour from the outside.

These rather appealing specifications (which are completely independent of which key-exchange protocol is under consideration) allowed him to find many different attacks (approximately 13, all of which would have worked on the original protocol; some of these were variants of the well-known Simmons attack [11] that depends on the symmetry of Vernam encryption) on a series of successively stronger versions of TMN. A typical attack found is set out below (in precisely the form produced by FDR except for the insertion of message numbers):

```
1: fake.A.S.B.Encrypt.pks.rspy.EndEncrypt,
2: comm.S.B.A.req,
3: comm.B.S.Encrypt.pks.rb1.EndEncrypt,
4: grab.S.A.Encrypt.rspy.rb1.EndEncrypt,
5: comm.A.S.B.Encrypt.pks.ra1.EndEncrypt,
6: grab.S.B.A.req,
7: fake.B.S.Encrypt.pks.rb1.EndEncrypt,
8: comm.S.A.Encrypt.ra1.rb1.EndEncrypt,
9: comm.B.A.Encrypt.rb1.dataBA.EndEncrypt
```

Here on message 1 the spy pretends (fakes) to be  $A$  and instigates a protocol run with  $B$  using a key ( $rspy$ ) known to it. In messages 2 and 3 the server and  $B$  respond to this, and in message 4 the spy intercepts (grabs) the signal which the server was sending back to  $A$ . The spy then, of course, understands  $rb1$ , the key  $B$  has invented. Now when  $A$  does ask for a session with  $B$  (message 5) the spy can intercept the second message and replay (for message 7) the third message of the first run, and the server then sends the key  $rb1$  back to  $A$ . Now  $A$  and  $B$  both believe they have a session open with the other using  $rb1$ , so  $B$  can send to  $A$  one of the symbolic message values  $dataBA$ . But the spy, since it knows  $rb1$ , thus learns this value which raises an error.

By successively strengthening the protocol to eliminate the attacks found at the previous stage, he eventually (in fact, the sixth version) arrived at a protocol which apparently<sup>2</sup> satisfies the above specification. This is described below.

---

<sup>2</sup>Because FDR is only able to cope with finite-state processes, checks of protocols have

1.  $A \rightarrow S : e_P(pks, m1.B, ra, S_A)$
2.  $S \rightarrow B : A.req$
3.  $B \rightarrow S : e_P(pks, m3.A, rb, S_B)$
4.  $S \rightarrow A : v(ra, rb)$

Note that the names of the proposed partners now appear encrypted in the first and third messages, together with secrets  $S_A$  and  $S_B$  which allow the server to authenticate the origin of the encryptions. ( $S_X$  is known only to the server and to  $X$ . It is just a simplified form of cryptographic signature that works here because it is only ever used to the trusted server.)  $m1$  and  $m3$  are just labels indicating which message in the run this is: without them messages 1 and 3 have identical format and can be confused, giving the spy extra ammunition.

But close examination of even this protocol reveals various failures of authentication that are too subtle for the extensional specification described above to find. These are, on the whole, less severe but do represent patterns of behaviour under which  $A$  and  $B$  are deceived by the intruder about some aspect of connections between them. The most obvious such attack is that the spy can always generate message 2 which will lead  $B$  (naïvely, but then most computer programs are naïve!) to believe  $A$  has requested a session with it. When  $B$  responds to this its part in the protocol run is complete so it can move into a state where it sends messages to  $A$  encrypted with the key it has invented. We will see another, rather more convincing, attack of the same general sort in the next section. In general, while an ‘attack’ like this one will often be discounted as irrelevant, anyone using a protocol must be made aware of it because:

- it is likely to cause problems in achieving liveness on top of the protocol and
- depending on the circumstances in which the protocol is used, there may be insecurities that result, even though these are not apparent in the protocol standing alone.

The essential problem here is that while the limitations of safety coding prevent us from making strong assertions about the two ends of a protocol always agreeing about the completion of a run, we can find ways of persuading one end that it has completed a run in circumstances appear ridiculous when we see them. It is difficult or impossible to find an extensional specification that will find such attacks for us, since one’s interpretation of ‘ridiculous’ here is certain to vary with the structure of the underlying protocol.

---

to be done on versions restricted to a fairly small number of sessions per node (perhaps 2) and with small numbers of objects like keys and nonces. Therefore all one can show is that no attack exists within the limits imposed for the check performed. Proving the adequacy of finite checks like these is a current topic of research.

### 3 Intensional specifications

The term ‘extensional’ refers to a property of an object that reflects its externally observable effect rather than how it was put together. There are well-known principles of extensionality in set theory – two sets are equal if and only if they have the same members – and function spaces – two functions of a domain  $D$  are the same if and only if they produce the same value on each  $x \in D$ . Thus an extensional property of a protocol is one referring to the effects produced by running it, but not about how it runs.

Most programming language semantics, particularly denotational semantics, strive hard to achieve extensionality: they seek to model a program by the essence of its effect rather than how this effect is achieved. But there are semantics which try to capture the way a computation is achieved, and in some circles these are termed *intensional* semantics (note the spelling). This word, in a sense the opposite of extensional, means “relating to the intent of”. Therefore we describe as intensional any specification whose primary purpose is to assert a property of the way, in terms of communications within the protocol, a particular state is reached.

The intent (in this restrictive sense) of most security protocols is extremely clear: they are usually described as a sequences of messages parameterised by things like node names, nonces, keys and timestamps. But there is a structure-clash between this style of description and the programs within the participating nodes which have to run the protocol. From the overall description we have to infer what each node’s view is, with the following natural assumptions:

- Provided a node gets all the communications *it* expects in a protocol run, it will carry on with the protocol, irrespective of what other nodes may or may not have done.
- When a node has completed all of its own communications it will assume that the protocol has completed successfully relative to whatever parameters have occurred within the communications it has seen.

Most protocols are sufficiently well designed that if the sequence of messages specified actually occurs between the intended nodes, then the appropriate result is obtained. Usually, attacks on these result from the spy participating in a run and fooling one or more of the other nodes into believing it is someone it is not. In other words, attacks generally result not from the overall protocol runs proceeding as intended, but from various nodes *believing* a run has been completed when it has not. A very natural specification we might want a protocol to satisfy is that no node can believe a protocol run has completed unless a correct series of messages has occurred (consistent as to all the various parameters) up to and including the last message the given node communicates. We call this the *canonical intensional specification* because it simply asserts that the protocol runs as expected.

This form of intensional specification is on the one hand highly dependent on the nature of the protocol under consideration, since it must contain a description of what a correct protocol run is. On the other hand it is a style of specification which can be used independently of what the intended effect of the protocol is. There is much similarity between this definition and the definition of security given in [4]. We will give a detailed comparison in a later section, since it is advantageous to understand our definition better before attempting this comparison.

Because of the expressive power of CSP, this style of specification is relatively easy to describe for FDR. The approach we have taken is set out below.

- The first thing to do is to identify a stage in the protocol, or other event, which should not be reached without a legitimate run having occurred. The natural points to identify here are the last communications in the run of each participating node, or specially introduced events that are communicated by the nodes when they think a successful run has occurred. Ordinarily, one would do this for each ‘proper’ node (i.e., not a server) that participates on the basis that these are the nodes which are likely to take further action on the basis that the run has completed. Thus we will get a separate canonical specification for each such node.

Since there is likely to be more than one node, we are likely to get several (probably similar but not identical) stages in the protocol from this analysis. We will simply assume that separate checks are performed for each, though it is possible to combine them (though, in our experience, quite complex). For example, in the well-known Needham Schroeder Secret Key protocol (NSSK, whose messages may be found below) since *A* sends and *B* receives the last message, *A* believes the run is complete once it is *sent*, and *B* when it is *received*.

- We then give a description of the possible sequences of messages that ought to have occurred before this stage in the proceedings. Now it is important to realise that each communication has two significant moments: when it is sent and when it is received. Note that the actions of the spy can make either of these happen without the other, and that the nodes cannot distinguish between the events corresponding to successful communications (in our CSP codings, typically the `comm` channel representing both output and input) and those interfered with by the spy (typically `grab` and `fake`).

A run of the NSSK protocol can be defined by the following CSP process, which contains in a slightly different form the same sort of information contained in our earlier descriptions of TMN protocols:

```

comm.A.server.B?Na ->
  [] k:FRESH_KEY @
    (comm.server.A.encrypt(pk(A),

```



```

Na.B.k.encrypt(pk(B),k.A) ->
comm.A.B.encrypt(pk(B),k.A) ->
[] Nb:FRESH_NONCE @
  (comm.B.A.encrypt(k,Nb) ->
   comm.A.B.encrypt(k,Nb.yes) ->
    STOP))

```

The ranges of the three parameters of the run (two nonces and one key) are introduced in the above by input (*?Na*) or by specifying their ranges explicitly.  $pk(X)$  is a private key between the server  $S$  and node  $X$ .

This can be compared with a more traditional description of the same protocol:

1.  $A \rightarrow S : B, N_A$
2.  $S \rightarrow A : e(pk(B), N_A, B, k_{AB}, e(pk(B), k_{AB}, A))$
3.  $A \rightarrow B : e(pk(B), k_{AB}, A)$
4.  $B \rightarrow A : e(k_{AB}, N_B)$
5.  $A \rightarrow B : e(k_{AB}, N_B - 1)$

Clearly we would expect that the successive messages in the protocol have first been output, and then input, by the appropriate nodes in order. We have to transform the above process a little to take account of this distinction, and it is also helpful to put some ‘signal’ events in. The sequence of messages representing a single legitimate session up to the point where  $A$  has completed a run with  $B$  in NSSK is represented by the following process. Apart from the non-protocol events `call_req.A.B` (the external request to  $A$  to connect with  $B$  and `connected.A.B` (which  $A$  emits for our convenience when it thinks it has connected with  $B$ ), all the communications are labelled either `outp` (representing a node sending a communication) and `inp` (a node receiving one). All but the last communication appear as `outp`, then `inp`, for the reasons discussed above.

```

OneSessionA = call_req.A.B ->
outp.A.server.B?Na ->
inp.A.server.B!Na ->
[] k:FRESH_KEY @
  (outp.server.A.encrypt(pk(A),
    Na.B.k.encrypt(pk(B),k.A)) ->
   inp.server.A.encrypt(pk(A),
    Na.B.k.encrypt(pk(B),k.A)) ->
    outp.A.B.encrypt(pk(B),k.A) ->
    inp.A.B.encrypt(pk(B),k.A) ->

```

```

[] Nb:FRESH_NONCE @
(outp.B.A.encrypt(k,Nb) ->
inp.B.A.encrypt(k,Nb) ->
outp.A.B.encrypt(k,Nb.yes) ->
CABs))

```

CABs = connected.A.B -> sync -> CABs

You should think of this as the series of messages that must have occurred to *justify* the event connected.A.B. The role of the process CABs and the event sync will be explained below.

- We can now make up the specification by combining together as many of the above processes as we wish to consider sessions. They can be combined together in different ways depending on whether we are prepared to allow our system to engage in parallel sessions or not. Assuming that we are, then the process

```

NSessionsA = (OneSessionA
  [|{sync}|]
  ...
  [|{sync}|]
  OneSessionA) \ {sync}

```

(with  $N$  copies of OneSessionA) is one that will allow each of the first  $N$  connected.A.B events only if they have *disjoint* but possibly overlapping (in time) protocol runs between  $A$  and  $B$  justifying them. After this the processes can all communicate sync and allow more connected.A.B's (since we are only considering the first  $N$  sessions).

We can now define the complete specification by

```

IntSpec =
  NSessionsA |||
  RUN(diff(Sigma,{connected.A.B}))

```

This is a trace specification that allows any events at all except that it only allows the first  $N$  connected.A.B's with justification.

- This is then a specification for trace refinement of the complete system modified so that the communications' names match those in the specification. This is a simple CSP renaming together with a simple trick for turning each ordinary communication between nodes into both an output outp and an input inp.

The specifications we are using here are really quite sophisticated: they say that disjoint sequences of communications must occur *somewhere* in the sequence in order for the chosen stage in the protocol to be reached. In some cases we have found that the normalisation which FDR performs on specifications is expensive because of this, but it has always been possible to overcome this by following a few simple strategies.

We will discuss the behaviour of the NSSK protocol relative to this specification and the complementary one looking for intensional attacks on *B* later.

### 3.1 When an intensional specification fails

When a canonical intensional specification fails you will have shown that the protocol under consideration can reach the chosen stage without the correct communications happening in the order used to describe the protocol. This may or may not represent something one might legitimately call an “attack”, but it will certainly illustrate a way in which the system can reach the chosen point in a way where the communications happened in an unexpected order – almost certainly because the spy was able to take part in a less trivial way than simply acting as a faithful messenger.

A good example of a seemingly harmless intervention by the spy comes in both versions of TMN. Running the intensional specification for *A*’s completion (the opposite one to the one which would produce the simple attack described at the end of the last section) produces a behaviour in which the spy anticipates the server sending the completely unprotected second message to *B*, placing this message before the arrival of the first. Indeed, *B* might send the third message (which is trapped by the spy) before *A* does anything. The actual order of visible actions of the spy found in our version of this is:

```
grab.A.S.Encrypt.pks.m1.ra1.B.sa.EndEncrypt
fake.S.B.A.req,
fake.A.S.Encrypt.ra1.B.sa.EndEncrypt,
grab.S.B.A.req,
comm.B.S.Encrypt.pks.m3.rb1.A.sb.EndEncrypt,
comm.S.A.Encrypt.ra1.rb1.EndEncrypt
```

This sort of behaviour, while seemingly innocent, was almost certainly unexpected by the protocol designer. If something like this is found you have two real options:

- The first is to take on board fully the implications of the unexpected behaviour, for example making sure it does not interfere with any other analysis you may have done which assumed the order of communications was the expected one. This will result in you weakening your mental picture of the protocol to encompass a wider range of actions on the part of the nodes, and should result in you weakening the intensional specification so that it now allows the behaviour which it previously pointed out. This

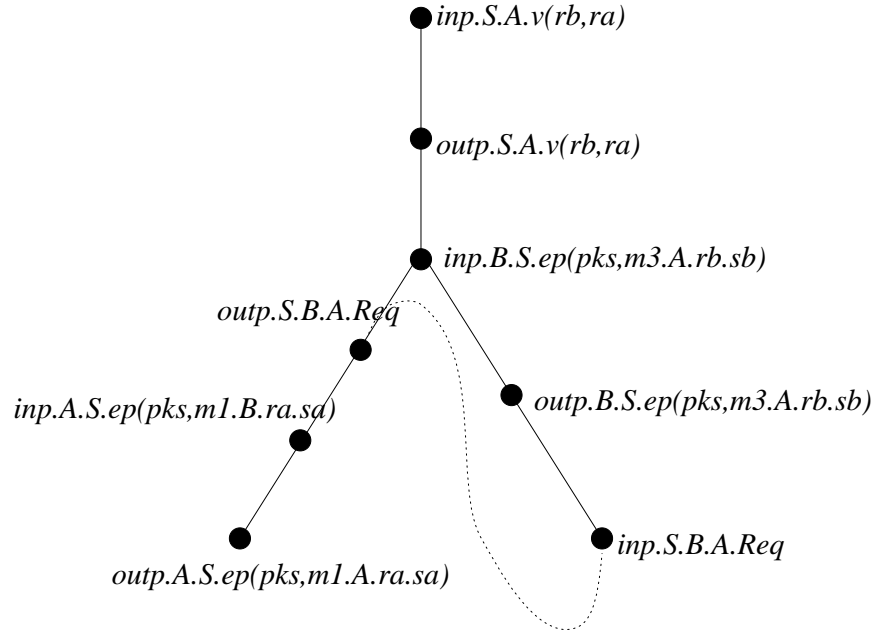


Figure 2: Temporal order of a weakened intensional specification

can readily be done for TMN. Rather than quote the CSP description of this it is perhaps clearer to show the weakened intensional specification as a diagram (Figure 2) illustrating the temporal order between actions that we are now requiring. (The meaning of the figure is that, with time increasing upwards, the two sequences of events below the vertex can be interleaved arbitrarily. The dotted line represents the ‘natural’ order in which everything on the left-hand leg precedes everything on the right-hand one.)

The range of possibilities here is quite large. For example, in protocols where a server issues other node’s public keys on request (suitably signed and perhaps with a timestamped ‘life’ for security), you are likely to find ‘attacks’ where the spy copies previously issued certificates of this sort, but if forced by the signatures to do so reliably and therefore creates no mischief. The way to allow this, should you want to, would be to drop the insistence that the server has to make a separate output of this message for each run.

- The second is to stiffen the protocol up to avoid the unexpected behaviour.

The obvious way to do this is to add nonces or timestamps. For example, we can add a nonce challenge into the second and third messages of the TMN protocols to avoid the attack described above.

If the first of these approaches (i.e., checking against the weakened intensional specification) is taken with the improved TMN protocol, an attack is found (provided the intensional specification for at least two sessions is used) under which the spy can convince *A* that a second session has been opened with *B* using the same key as an earlier one, when *B* has no inkling of the second session. The attack is described by the following series of messages (seen from the perspective of the spy).

```
1: comm.A.S.Encrypt.pks.m1.ra1.B.sa.EndEncrypt,
2: comm.S.B.A.req,
3: comm.B.S.Encrypt.pks.m3.rb1.A.sb.EndEncrypt,
4: comm.S.A.Encrypt.ra1.rb1.EndEncrypt,
5: comm.A.S.Encrypt.pks.m1.ra2.B.sa.EndEncrypt,
6: grab.S.B.A.req,
7: fake.B.S.Encrypt.pks.m3.rb1.A.sb.EndEncrypt,
8: comm.S.A.Encrypt.ra2.rb1.EndEncrypt
```

The essential point about this is that there is no guarantee that the message sent by *B* containing the key is fresh, so the spy can re-use (message 7) an old one. The fact that *A* can be persuaded to re-use an old key makes this attack more dangerous since an old key might have been compromised, and even if not *B*'s messages from the first session can be replayed. Subjectively, it seems more dangerous than the attack on *B* described in the last section.

This type of behaviour – which intensional specifications are very good at identifying – generally would be regarded as a genuine attack representing a flaw in the protocol. It is interesting that exactly the same strengthening suggested in the second approach above will avoid this replay attack. It is possible to argue in this case that, even though the first unexpected behaviour was not something we should regard as an attack, it was a symptom of the same weakness that led to the attack. Subject again to the caveat that checks are performed on systems with finitely bounded capabilities, we could find no intensional attack on *A* in the system with a nonce challenge between the second and third messages.

### 3.2 When an intensional specification succeeds

It is both the strength and the weakness of intensional approach that all it does is test the implemented protocol's behaviour against the designer's expectations. The strength comes from the likelihood that the designer probably has a pretty good idea of what the protocol achieves when it runs as anticipated: quite probably capturing more subtleties than any given extensional specification.

The main weakness is that it tells us nothing in any abstract sense about what the protocol does, so some method must be used to gain confidence about

this. This could either be establishing some extensional specification such as confidentiality or performing abstract (i.e., using some logic such as BAN) analysis.

You should also bear in mind that canonical intensional specifications will not catch any attack which takes the form of the spy replaying the contents of a completed session through the server (or another node), perhaps modified, to learn things about it. A good example of this is provided by the original TMN protocol. While the attack described in which the spy learns a current session key between  $A$  and  $B$  *would* have counted for various reasons as an intensional attack, there is another one with the same effect that does not. In this one  $A$  and  $B$  are allowed to set up a session normally, and then the spy (posing either as  $A$  or a third party  $C$  but in either case using a key it knows) requests a session to  $B$ , intercepts the outgoing request from the server and replays the third message of the first session. The spy is then told the key of the current session by the server encrypted under a key it knows. All this has happened without  $A$  or  $B$  being involved in the second ‘session’ at all. Usually, as in this case, there seem to be other attacks on the same protocols which do trip the canonical intensional specifications, but it would be unwise to rely on this.

## 4 The behaviour of NSSK

We used this protocol earlier to illustrate how a canonical intensional specification is created. It, and weakened versions of it, provide another useful insight into how this specification works. Consider first the original protocol, defined by the series of messages set out as a simple CSP process earlier. On feeding this into FDR we discovered no attack (i.e., the size of check we were able to perform succeeded) on the assumption that the spy cannot guess nonces in advance. But if it can guess them – as would be the case if the nonces were, for example, generated by a counter – then the specification fails because the spy can predict the nonce to be used by  $A$  in message 1 and thereby get a reply from the server to  $A$ ’s initial request before it is made. When  $A$  does make the request, this is intercepted and the message recorded from the server is replayed. While this does not directly breach security, it creates cryptanalytic possibilities. In other words, we have to distinguish two separate levels of nonce here: uniqueness and unpredictability. It appears that the two nonces used in this protocol are at different levels here, since no attack is found on the assumption that the second nonce is predictable (but unique).

An interesting weakening of NSSK is obtained by dropping the name  $B$  from the contents of message 2. This at first seems reasonable since  $A$  ought to be able to know the message was for a session with  $B$  because of the nonce. But dropping this name leads to the attack where the spy intercepts message 1 and replaces  $B$ ’s name with its own.  $A$  then effectively opens a session with the spy, in the mistaken belief it is talking to  $B$ .

If we allow the second nonce to be repeated, then we find a replay attack in which the spy can persuade  $B$  that a second session (on the same key) is open with  $A$  when it is not. It is found by the canonical intensional specification since there is not activity in  $A$  to justify two sessions.

For comparative purposes, the extensional safety specification seen earlier finds only the ‘no name’ attack, while our coding of NSSK to achieve no loss of service finds the second and third, but not the distinction between predictable and non-predictable first nonces. It does, however, find an attack when we allow the first nonce to be repeated (which the extensional safety specification does not).

## 5 Utility relative to known attacks

Except for attacks which depend on some cryptanalytic power, or similar, on the part of the spy, almost all the attacks on protocols known to the author would be detected by the canonical intensional specifications for the given protocols. Classic patterns of attack that will always be found are

- *Reflection* attacks, where the node  $A$  (say) makes some sort of challenge to  $B$ , and the spy is able to answer this challenge by replaying the challenge to  $A$  (using it as an *oracle*).
- *Man-in-the-middle* where the spy conducts simultaneous sessions with  $A$  and  $B$  in such a way that one of  $A$  and  $B$  is deceived into thinking that a session has been opened with the other (which generally is unaware that this has been attempted). An example of this is provided by Lowe’s new attack on the Needham-Schroeder Public Key protocol [5, 6].
- *Parallel Session* attacks are a generalisation of this and simply represent any situation where the spy manipulates interleaved sessions to deceive one or more of the legitimate participants. A large number of these are set out in [3].

It has often been observed that no protocol could be proof against the spy acting as a faithful messenger between  $A$ ,  $B$  and (where needed) servers. Our specifications will allow this sort of behaviour, but will forbid man-in-the-middle style attacks where the messages delivered are not quite the same as those sent so that each of  $A$  and  $B$  believe they have run the protocol with each other but disagree on some aspect of the contents (perhaps the key, or amount of money exchanged!).

- *Duplicate session* attacks, where the spy can convince one node, say  $A$ , that the other is participating in a second protocol run, though  $B$  has not heard of the second run. A good example of this is the one found on  $A$  in the improved TMN protocol earlier. Some authors classify these as

*freshness* attacks, since they generally involve persuading a node to re-accept an object it has accepted before. This is a slightly dangerous term, since authors frequently seek to guarantee freshness with timestamps and it is important to ensure that a message containing a timestamp  $T$  cannot be re-accepted within the validity of the  $T$ .

Freshness attacks are detected simply unless they (like the Denning-Sacco and similar attacks on NSSK) depend on a key being broken. See the next section for discussion of this.

## 6 Timing and intensional specifications

Time appears in security protocols and their analysis in at least three ways. The most obvious is the frequent use of timestamps in protocol messages to help verify freshness of messages and perhaps signify a message's lifetime as a 'certificate'. The second is that one would expect any reasonable protocol implementation, even where no time is mentioned explicitly in the abstract protocol description, to involve *timeouts*: any node sending a message to which it expects a reply will set a time limit on the response.<sup>3</sup> And finally, there is the rather abstract concept of freshness as captured, for example, in BAN logic [2] where the only distinction is between things that are fresh and things that are not.

It is possible to model all of these in CSP. The first two are rather similar except that with timestamps we require a new class of symbolic object in our programs as well as a representation of the passage of time. In the final one we will typically include an event that can happen at any time, separating the old from the new, and which resets all nodes to their base states (with no runs in progress). One use of this latter model would be to make one or more keys used *before* this dividing line available to the spy after it (representing the assumption that old keys can become compromised) and seeing what effect this has on whether the protocol satisfies its specification.

While the choice of whether or not to use time is not directly related to the question of whether intensional specifications are used, if it is used in any of these forms the presence of time in the model can be used to tighten the specifications to include our expectations about how a protocol run happens in time.

- With timeouts one will immediately gain an expectation of the length of time a protocol run will take. In a timed model (whether in CSP or any other suitable notation) it would be straightforward to include this and similar expectations in intensional specifications since one could place bounds on the times at which the series of events required occur.

---

<sup>3</sup>In the absence of this the spy can deadlock the node by persuading it to send out a message to which no response will ever come.



- With timestamps one would have the same possibility (in suitable cases) and also include the relationships between timestamps, and between timestamps and event times, in the specifications.
- With the model with a reset to detect freshness we would probably wish to assert that every session comes completely before or after the reset action, and could of course test intensional specifications like any other in the context of the compromised old keys.

We have found (using FDR) a number of variants of the Denning-Sacco attack on the NSSK protocol (the basis of all of which are that there is nothing to convince  $B$  that the third message is fresh). This, of course, comes into the last category above.

It would be interesting to investigate the phenomena described in the entertainingly named section ‘freshening the breath of the wide-mouthed frog’ of [1]. There are variations on an attack on this protocol using the server as an oracle to gradually increase the timestamp inside an encrypted message. I believe (though have not yet implemented this) that these would be an excellent example what could be found by an intensional specification of the second type above. It would be necessary here to use a somewhat weakened specification.

## 7 Conclusions and comparisons

What I am advocating in this paper is the use of specifications which take a global view of the communications in the network resulting from the implementation of a protocol and assert that they (the communications) only follow anticipated patterns. Particularly for protocols involving an element of authentication, the *canonical* intensional specifications appear to provide a readily accessible way of seeking and judging attacks. They are very strong specifications, but it seems obvious that it is better to start with a specification that is too strong and be forced to weaken it after careful thought than to make one that is too weak.

I re-iterate that they provide a separate criterion for judging a protocol to the more abstract extensional specifications one may be able to invent and should not be thought of as a substitute, except perhaps in the area of authentication.

The power of this style of specification in classifying authentication errors is shown in Lowe’s recent work [7]. There he points out a number of subtle mis-behaviours of protocols all of which demonstrate failures of the canonical intensional specification. Whether or not one would wish to assign as strong a term as *attack* to all of these (and I believe there is good reason to particularly when one imagines attempting to achieve no-loss-of-service on top of them), they are all behaviours of which anyone considering using them should be aware.

It is interesting that one of these is the STS protocol defined in [4], since, as stated earlier, that paper’s definition of authentication is the the one closest to

our own and these authors claim that the STS protocol is secure. In fact, Lowe's attack (in which the spy persuades Alice that she has a connection with Bob even though Bob has never heard of Alice, believing it has had an incomplete run with a third party) would seem close to violating the definition of 'secure' given in [4], the relevant part of which is (paraphrased)

A run is insecure if Alice accepts the other party's identity when the the other party's record of the run does not match Alice's.

which is clearly an intensional specification, though as we will see not quite the canonical one. The sense in which Lowe's attack might not be said to violate this is because of the definition given of 'matching', where 'only those parts of the messages relevant to authentication' must match. Obviously, in this case, the identity of the person with whom Bob believes he is communicating is not considered relevant. In other words, at the point where Alice accepts Bob's identity, their two runs do 'match', even though they believe the messages that match were transacted between different parties! This is not as bizarre as it sounds, since the address and sender fields of messages are not generally considered secure. Nevertheless it does point to a clear way in which our specification is stronger than theirs.

There are others: firstly, there is nothing in their specification which would forbid a message arriving before it was sent (due to the spy predicting a message, as in the TMN protocol example where the spy forecast the unprotected third message), and they do not explicitly consider multiple sessions between a given pair of participants, so that it is not clear that messages corresponding to distinct sessions must be disjoint. Finally, their definition only considers protocols with communications only between a given pair of nodes, and thus does not include the server.

Intensional specifications are, in a sense, verifications of authentication. But what we are authenticating is actually the beliefs of the designer about how the protocol runs. This style of specification can be used with equal effect over a much wider range of protocol than authentication, though it must be recognised that the further away from this area one gets the more likely it is that one will have to use something a little weaker than the canonical one.

## **Acknowledgements**

The work reported in this paper has benefitted from discussions with numerous people, in particular Paul Gardiner, Michael Goldsmith, Gavin Lowe, Paul Gardiner, Gavin Lowe and Peter Ryan.

It was supported by funds from DRA Malvern (contracts CSM 3/038 and CSM 106), the US Office of Naval Research and a grant from EPSRC under the ROPA programme.

## References

- [1] R. Anderson and R. Needham, *Programming satan's computer* Cambridge University Technical Report
- [2] M. Burrows, M. Abadi and R.M. Needham, *A logic of authentication*, ACM transactions on Computer Systems, **8**, 1 (1990), pp18-36.
- [3] J. Clark and J. Jacob. *A Survey of Authentication Protocol Literature* York University Technical Report
- [4] W. Diffie, P.C. van Oorschot and M.J. Wiener. *Authentication and key exchanges*. Design, Codes and Cryptography **2**, pp107-125 (1992).
- [5] G. Lowe, *An Attack on the Needham-Schroeder Public-Key Authentication Protocol*, Information Processing Letters 56 (1995) pp131-133.
- [6] G. Lowe *Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR*, Proceedings of TACAS 1996 (LNCS).
- [7] G. Lowe, *Some New Attacks upon Security Protocols*, In this volume.
- [8] A.W. Roscoe, *Modelling and verifying key-exchange protocols using CSP and FDR* Proceedings of CSFW8 IEEE Press 1995.
- [9] A.W. Roscoe, *Model-checking CSP*, in A Classical Mind: Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed.) Prentice-Hall 1994
- [10] A.W. Roscoe, J.C.P. Woodcock and L. Wulf, *Non-interference through determinism*, Proc. ESORICS 94, Springer LNCS 875, pp 33-53.
- [11] Gustavus J. Simmons. Cryptanalysis and protocol failures. *Communications of the ACM*, 37(11):56-65, 1994.
- [12] Makoto Tatebayashi, Natsume Matsuzaki, and David B. Newman, Jr. Key distribution protocol for digital mobile communication systems. In *Advances in Cryptology: Proceedings of Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 324-333. Springer-Verlag, 1990.