Analyzing a Library of Security Protocols using Casper and FDR

Ben Donovan Paul Norris Gavin Lowe

Department of Mathematics and Computer Science, University of Leicester,
University Road, Leicester, LE1 7RH, UK.

Abstract

In this paper we describe the analysis of a library of fifty security protocols using FDR, a model checker for the process algebra CSP, and Casper*, a compiler that produces the CSP descriptions from a more concise description. We succeed in finding nearly all of the attacks previously reported upon these protocols; in addition, we identify several new attacks.*

1 Introduction

In recent years, model checking has proved to be a very successful way for analyzing security protocols. In this paper we describe the application of model checking techniques to Clark and Jacob's library of security protocols [3]. This library has been the subject of a previous study [2], with which we can compare our results.

We have used FDR, a model checker for the process algebra CSP [9], for the analysis. The CSP descriptions of the protocols were prepared using Casper [7], a compiler that produces the CSP from a more concise description. The ease of our techniques is evidenced by the fact that most of the analyses were carried out by the first two authors as part of their final year undergraduate projects; these two authors knew nothing about security protocols at the beginning of the year (and still know little about CSP) yet were able to get to grips with the technique, and to find new attacks.

Of the 50 protocols in the library, we were able to analyze all but one. We found attacks on 20 of the 25 previously known to be insecure. Further, we found attacks upon ten protocols reported as secure, and six new attacks upon protocols reported to be flawed in [3, 2].

The main contributions of this paper are:

- A tutorial on the Casper/FDR approach to analyzing security protocols;
- A study of how well these techniques can be applied to a large collection of protocols, together with an identification of a few shortcomings;
- Identification of a few (to the best of our knowledge) previously undocumented attacks.

In the next section we briefly describe the Casper/FDR approach to analyzing security protocols; in Section 3 we give an overview of our results, describe a few new attacks we found, and note a few shortcomings. In Section 4 we sum up, give some pointers to future developments of Casper, and identify some challenges faced by the protocol analysis community.

2 Analyzing security protocols using Casper and FDR

The basic approach to analyzing a security protocol using CSP and its model checker FDR is as follows.

- Each honest agent running the protocol is modelled as a CSP process;
- The most general (i.e. nondeterministic) intruder who can interact with the protocol is also modelled in CSP; this intruder can: overhear messages; intercept messages; encrypt and decrypt messages with keys it knows; and send possibly fake messages to other agents;
- A system is formed from the intruder and a finite collection of honest agents, combined together in parallel;
- Various security properties such as *secrecy* or *authentication* are formalized as CSP specification processes;
- FDR is used to test whether the system satisfies the specifications.

See [5, 8] for a fuller description of this technique.

However, this technique requires considerable expertize in CSP; and even for the CSP expert, the construction of the system is time-consuming and error-prone. For these reasons, the third author developed Casper, a compiler that produces the CSP from a more abstract and concise description. All the CSP scripts used to perform the analyses reported in this paper were prepared using Casper.

We describe below a typical input file for Casper, taking as our example the familiar and much studied Needham Schroeder Public Key Protocol:

$$\begin{array}{ll} 1. & a \to b \ : \ \{na, a\}_{PK(b)} \\ 2. & b \to a \ : \ \{na, nb\}_{PK(a)} \\ 3. & a \to b \ : \ \{nb\}_{PK(b)} \end{array}$$

A fuller description of Casper can be obtained from the manual, available via the Casper World Wide Web home page.

The file is split into two parts: defining the protocol, as a template; and defining the actual system to be analyzed by FDR, as an instantiation of the template. These two parts are further subdivided into (normally) eight sections; each section is headed by a line beginning with "#".

The `#Free variables` section defines types for variables and functions that are used in the definition of the protocol:

```
#Free variables
a, b : Agent
na, nb : Nonce
PK : Agent -> PublicKey
SK : Agent -> SecretKey
InverseKeys = (PK, SK)
```

`a` and `b` are agents' identities; `na` and `nb` are nonces; `PK` and `SK` are functions that return an agent's public and secret key; the "`InverseKeys`" line defines that these two functions return keys that are inverses of one another. Throughout this paper we use small letters for the names of free variables, and capitals for values with which these variables are instantiated.

The `#Protocol description` section defines the protocol itself, as a sequence of messages, using an ASCII representation of the standard notation for protocols (for example, writing "`{m}{k}`" for $\{m\}_k$).

```
#Protocol description
0.    -> a : b
1.  a -> b : {na, a}{PK(b)}
2.  b -> a : {na, nb}{PK(a)}
3.  a -> b : {nb}{PK(b)}
```

The message `0` above is designed to get the protocol going, to inform `a` who he should run the protocol with (i.e. the identity of `b`); it can be thought of as a message from a user or the environment.

The `#Processes` section provides information about the agents in the protocol.

```
#Processes
INITIATOR(a, na) knows PK, SK(a)
RESPONDER(b, nb) knows PK, SK(b)
```

The "`INITIATOR`" and "`RESPONDER`" give names to the roles played by the agents, and provides a mechanism for defining the actual system to be checked (in the `#System` section, below). The first variable inside the parentheses is the identity of the agent, as used in the protocol description. The rest of the variables inside the parentheses are those data items initially known by the agent, and for which the agent can choose different values in different runs. The functions and function applications following the "`knows`" are those initially known by the agent. Casper uses these parameters to perform some consistency checking, which helps to catch most user errors.

The `#Specification` section defines the specifications or requirements of the protocol.

```
#Specification
Secret(a, na, [b])
Secret(b, na, [a])
Agreement(a, b, [na,nb])
Agreement(b, a, [na,nb])
```

The first two specifications above are secrecy specifications; the first can be paraphrased as "`a` believes that `na` is a secret shared with `b`"; any situation where the intruder learns the value of `na` in a run where `b` is not the intruder will constitute an attack. The third and fourth specifications are a form of authentication specification; the first can be paraphrased as "`a` is authenticated to `b` in the sense that if `b` completes a run, apparently with some agent `a`, then the same `a` has been running the protocol with `b`, and the two agree upon the roles each took, upon the values of `na` and `nb`, and upon the number of runs".

We now move onto the definition of the actual system to be checked by FDR. The `#Actual variables` section instantiates the types from the `#Free variables` section.

```
#Actual variables
Alice, Bob, Ivor : Agent
Na, Nb, Nm : Nonce
```

The `#Functions` section similarly instantiates the functions.

```
#Functions
symbolic PK, SK
```

The "`symbolic`" keyword indicates that Casper will create its own (injective) definitions for these functions.

The `#System` section defines the actual system to be checked using FDR.

```
#System
INITIATOR(Alice, Na)
RESPONDER(Bob, Nb)
```

Here Casper will create a system where `Alice` can run the protocol once as initiator, and `Bob` can run the protocol once as responder.

Finally, the `#Intruder Information` section defines the identity of the intruder, and his initial knowledge.

```
#Intruder Information
Intruder = Ivor
IntruderKnowledge = \
  {Alice, Bob, Ivor, Nm, PK, SK(Ivor)}
```

(The "\" allows a single logical line to be split across several physical lines.)

An input script such as this can be prepared in about five to ten minutes by a reasonably experienced user. Casper can then be used to produce the corresponding CSP description, which can be checked using FDR.

3 The case studies

In this section we report upon our analysis of the protocols from Clark and Jacob's library of security protocols [3]. Our results are summarized in Figures 1 and 2; we include a comparison with Clark and Jacob's results, and with Brackin's results from [2]. In these figures, we write "Attack(?)" to denote that attacks are reported that we consider somewhat dubious, because they depend upon very particular implementation details, and would not be successful against other implementations.

We discuss below some of the differences between our results and the previous results, in particular, noting some new attacks; many of the new attacks we find are not necessarily serious, but they all show that the protocols do not satisfy as strong a specification as might be expected. It is clearly important to know the limitations of a protocol, so as to not place an over-reliance upon it.

3.1 Multiplicity attacks

Several of the attacks that we find are what we term *multiplicity attacks*: where two agents disagree about the number of times the protocol has been run. The seriousness of such attacks depends upon the use to which the protocol is put—if after a completed run some money is debited from a bank account, then such an attack is extremely serious; in any case, we should be aware of the limitations of a protocol.

A typical example of such an attack applies to the ISO Public Key Two-Pass Mutual Authentication Protocol:

$$\begin{array}{ll} 1. & a \rightarrow b \;:\; na, b, \{na, b, text1\}_{SK(a)} \\ 2. & b \rightarrow a \;:\; nb, a, \{nb, a, text2\}_{SK(b)} \end{array}$$

where $SK(a)$ is a's secret key. An intruder can watch such an exchange and then replay Message 1 to cause b to complete a second run; alternatively, the intruder can replay Message 2 to cause a to complete a second run. This attack can also be considered as a lack of freshness attack—there is nothing in either message that the recipient knows is fresh.

Careful implementation checks will prevent such attacks, but it is the duty of the protocol designer to describe such necessary checks.

Several other ISO protocols are subject to such attacks—the Symmetric Key One-Pass Unilateral Authentication, Symmetric Key Two-Pass Mutual Authentication, One-Pass Unilateral Authentication with CCFs, Two-Pass Mutual Authentication with CCFs, Public Key One-Pass Unilateral Authentication, and Public Key Two-Pass Mutual Authentication Protocols—as are the Kerberos Protocol, and the Davis Swick Private Key Certificates protocols.

3.2 Role confusion

The ISO Public Key Two-Pass Mutual Authentication Protocol can also be attacked in a way that causes the two agents to disagree upon which roles each took:

$$\begin{array}{lrcl} \alpha.1. & Alice \rightarrow I_{Bob} & : & Na, Bob, \\ & & & \{Na, Bob, Text1\}_{SK(Alice)} \\ \beta.1. & Bob \rightarrow I_{Alice} & : & Nb, Alice, \\ & & & \{Nb, Alice, Text2\}_{SK(Bob)} \\ \beta.2. & I_{Alice} \rightarrow Bob & : & Na, Bob, \\ & & & \{Na, Bob, Text1\}_{SK(Alice)} \\ \alpha.2 & I_{Bob} \rightarrow Alice & : & Nb, Alice, \\ & & & \{Nb, Alice, Text2\}_{SK(Bob)} \end{array}$$

The attack uses two runs, α and β; the notation I_{Alice} represents the intruder Ivor imitating Alice to fake a message from her, or intercepting a message intended for her. Here both agents believe that they initiated the exchange; whether this disagreement over roles is important, depends upon the use to which the protocol is put. The weakness is that the two messages in the protocol have identical forms, so a Message 1 may be replayed in the place of a Message 2.

Similar attacks work on the ISO Symmetric Key Two-Pass Mutual Authentication and Two-Pass Mutual Authentication with CCFs Protocols.

3.3 Self-authentication attacks

Some protocols can be attacked if an agent runs it with herself, using the same key as both initiator and responder; a possible scenario would be where the protocol is

Protocol	Our results	Clark & Jacob's results	Brackin's results
ISO Symmetric Key One-Pass Unilateral Authentication Protocol	Attack	No attack	No attack
ISO Symmetric Key Two-Pass Unilateral Authentication Protocol	No attack	No attack	No attack
ISO Symmetric Key Two-Pass Mutual Authentication Protocol	Attack	No attack	No attack
ISO Symmetric Key Three-Pass Mutual Authentication Protocol	No attack	No attack	No attack
Using Non-Reversible Functions	No attack	No attack	No attack
Andrew Secure RPC Protocol	Attack	Attack	Attack
ISO One-Pass Unilateral Authentication with CCFs	Attack	No attack	No attack
ISO Two-Pass Unilateral Authentication with CCFs	No attack	No attack	No attack
ISO Two-Pass Mutual Authentication with CCFs	Attack	No attack	No attack
ISO Three-Pass Mutual Authentication with CCFs	No attack	No attack	No attack
Needham Schroeder Protocol with Conventional Keys	No attack	Attack	Attack
Denning Sacco Protocol	No attack	No attack	No attack
Otway-Rees Protocol	Attack	Attack(?)	No attack
Amended Needham Schroeder Protocol	No attack	No attack	No attack
Wide-Mouthed Frog Protocol	Attack	Attack	No attack
Yahalom Protocol	No attack	Attack(?)	Attack(?)
Carlsen's Secret Key Initiator Protocol	No attack	No attack	No attack
ISO Four-Pass Authentication Protocol	No attack	No attack	No attack
ISO Five-Pass Authentication Protocol	No attack	No attack	No attack
Woo and Lam Authentication Protocol Π_f	No attack	No attack	No attack
Woo and Lam Authentication Protocol Π_1	Attack	Attack	No attack
Woo and Lam Authentication Protocol Π_2	Attack	Attack	No attack
Woo and Lam Authentication Protocol Π_3	Attack	Attack	No attack
Woo and Lam Authentication Protocol Π	Attack	Attack	No attack
Woo and Lam Mutual Authentication protocol	Attack(?)	Attack(?)	No attack
Needham-Schroeder Signature Protocol	Attack	No attack	Attack
Kerberos Version 5	Attack	No attack	No attack
Neuman Stubblebine	Attack	Attack	No attack
Kehne Langendorfer Schoenwalder	Attack	Attack	No attack
Kao Chow Repeated Authentication Protocol (1)	No attack	Attack	Attack
Kao Chow Repeated Authentication Protocol (2)	No attack	No attack	Attack
Kao Chow Repeated Authentication Protocol (3)	No attack	No attack	Attack
ISO Public Key One-Pass Unilateral Authentication Protocol	Attack	No attack	No attack
ISO Public Key Two-Pass Unilateral Authentication Protocol	No attack	No attack	No attack
ISO Public Key Two-Pass Mutual Authentication Protocol	Attack	No attack	No attack
ISO Three-Pass Mutual Authentication Protocol	No attack	No attack	No attack
ISO Two Pass Parallel Mutual Authentication Protocol	No attack	No attack	No attack
Bilateral Key Exchange with Public Key	No attack	No attack	No attack
Diffie Hellman Exchange	Unanalyzed	No attack	No attack
Needham Schroeder Public Key Protocol	Attack	Attack	No attack
SPLICE/AS Authentication Protocol	Attack	Attack	No attack
Hwang and Chen's modified SPLICE/AS Authentication Protocol	Attack	Attack	No attack
Denning Sacco Key Distribution with Public Key	Attack	Attack	No attack
CCITT X.509	Attack	Attack	No attack
Shamir Rivest Adelman Three Pass protocol	Attack	Attack	Unanalyzed
Gong Mutual Authentication Protocol	Unanalyzed	No attack	No attack

Figure 1. Summary of results

Protocol	Our results	Clark & Jacob's results	Brackin's results
Encrypted Key Exchange-EKE	Attack	Attack	No attack
Davis Swick Private Key Certificates, protocol 1	Attack	Attack	Attack
Davis Swick Private Key Certificates, protocol 2	Attack	Attack	Attack
Davis Swick Private Key Certificates, protocol 3	Attack	No attack	No attack
Davis Swick Private Key Certificates, protocol 4	Attack	No attack	No attack

Figure 2. Summary of results, continued

used to transfer files between accounts on different computers owned by the same person. We call such attacks *self-authentication attacks*. Again, the ISO Public Key Two-Pass Mutual Authentication Protocol serves as illustration:

$$\begin{array}{lrcl} 1. & Alice \rightarrow I_{Alice} & : & Na, Alice, \\ & & & \{Na, Alice, Text1\}_{SK(Alice)} \\ 2. & I_{Alice} \rightarrow Alice & : & Na, Alice, \\ & & & \{Na, Alice, Text1\}_{SK(Alice)} \end{array}$$

The problem here is that when Alice runs the protocol as both initiator and responder, the two messages have identical forms, and so the intruder is able to replay a message 1 and have it interpreted as a message 2.

Similar attacks can be launched on several other ISO protocols—the Symmetric Key Two-Pass Mutual Authentication and Two-Pass Mutual Authentication with CCFs Protocols.

3.4 Needham Schroeder Signature Protocol

The Needham Schroeder Signature Protocol is designed to authenticate a message m sent from a to b, with the aid of a server s:

$$\begin{array}{lrcl} 1. & a \rightarrow s & : & a, \{h(m)\}_{Shared(a,s)} \\ 2. & s \rightarrow a & : & \{a, h(m)\}_{Private(s)} \\ 3. & a \rightarrow b & : & m, \{a, h(m)\}_{Private(s)} \\ 4. & b \rightarrow s & : & b, \{a, h(m)\}_{Private(s)} \\ 5. & s \rightarrow b & : & \{a, h(m)\}_{Shared(b,s)} \end{array}$$

where $Private(s)$ is a secret key known only by s, $Shared(a, s)$ is a key shared between a and s, etc, and h is a cryptographic hash function. Brackin [2] reports a multiplicity attack (which may also be viewed as a lack of freshness attack), which we can also find. However, we found another attack, which shows that the protocol does not satisfy as strong a property as we expected. It turns out that when b receives the final message, he has no guarantee that a intended the message for him, as opposed to some third party; the following attack shows this:

$$\begin{array}{lrcl} \alpha.1. & Alice \rightarrow Sam & : & Alice, \{h(M)\}_{Shared(Alice,Sam)} \\ \alpha.2. & Sam \rightarrow Alice & : & \{Alice, h(M)\}_{Private(Sam)} \\ \alpha.3. & Alice \rightarrow I_{Carol} & : & M, \{Alice, h(M)\}_{Private(Sam)} \\ \beta.3. & I_{Alice} \rightarrow Bob & : & M, \{Alice, h(M)\}_{Private(Sam)} \\ \beta.4. & Bob \rightarrow Sam & : & Bob, \{Alice, h(M)\}_{Private(Sam)} \\ \beta.5. & Sam \rightarrow Bob & : & \{Alice, h(M)\}_{Shared(Bob,Sam)} \end{array}$$

Alice intended the message for Carol, but Bob ends up thinking that it was intended for him. We consider this to show a limitation of the protocol, rather than necessarily being a serious flaw; however, it acts as a reminder that we should be sure about precisely what guarantees are provided by a protocol. The flaw that allows this attack is the absence of the responder's identity from any of the encrypted messages.

3.5 Diffie Hellman key exchange

The Diffie Hellman key exchange can be described as follows:

$$\begin{array}{lrcl} 1. & a \rightarrow b & : & G^x \bmod N \\ 2. & b \rightarrow a & : & G^y \bmod N \end{array}$$

where N is a large, publicly known number, and G is a publicly known generator of the field $F \mathrel{\widehat{=}} \{0, \ldots, N-1\}$. The two agents then compute the key:

$$(G^y \bmod N)^x \bmod N \;=\; (G^x \bmod N)^y \bmod N. \quad (1)$$

a calculates the key according to the left hand side, and b calculates it according to the right hand side. The protocol depends for its security upon the difficulty of calculating discrete logarithms over the large finite field F.

We have extended Casper so as to be able to model user-defined types like F, by writing, for example:

```
datatype F = G | Exp(F,Num) unwinding 2
```

This defines F in a way that should be familiar to functional programmers; a term $Exp(f, x)$ represents $f^x \bmod N$. The "`unwinding 2`" indicates that only those values of the datatype obtainable by unwinding the equations at most twice should be considered (i.e. using at most two applications of the `Exp` constructor); we need such a restriction or else the datatype, and hence the state space, would be infinite.

The protocol uses the commutativity of exponentiation, captured by equation (1). Casper includes a mechanism for the user to be able to specify algebraic properties of the message space; the commutativity of exponentiation can be specified as follows:

```
#Equivalences
forall x, y : Num . \
  Exp(Exp(G,y), x) = Exp(Exp(G,x), y)
```

When we model and analyze the Diffie Hellman protocol using Casper and FDR, a fairly obvious attack is found, showing that the exponentials are not authenticated:

$$\begin{array}{lrcl} 1. & I_{Alice} \rightarrow Bob & : & Exp(G, Z) \\ 2. & Bob \rightarrow I_{Alice} & : & Exp(G, Y) \end{array}$$

After this exchange, the intruder knows the key

$$(G^Y \bmod N)^Z \bmod N \ = \ (G^Z \bmod N)^Y \bmod N$$

which Bob believes he shares with Alice. This attack is easily prevented by signing the exponentials, with appropriate explicitness:

$$\begin{array}{lrcl} 1. & a \rightarrow b & : & \{b, G^x \bmod N\}_{SK(a)} \\ 2. & b \rightarrow a & : & \{G^y \bmod N, a\}_{SK(b)} \end{array}$$

3.6 Type flaws

Our approach (in common with nearly all protocol analysis techniques) is not good at finding type flaws, where a field (or several fields) is interpreted as being of a different type to what was expected.

Casper has a limited feature whereby an atomic value can be defined as having two different types. We have used this feature to find an attack upon the Woo and Lam Mutual Authentication Protocol, previously reported in [6], and an attack upon the Neuman Stubblebine Protocol, previously reported in [4]. However, our Casper scripts was very much oriented towards finding particular attacks, that we already know about, and it is unlikely that we would have found these attacks otherwise.

3.7 Miscellany

Clark and Jacob report an implementation-dependent type flaw attack upon the Otway Rees protocol. We found a different attack (previously reported in [10]), where the two agents running the protocol end up with different session keys.

Clarke and Jacob report some implementation dependent attacks against four of the Woo and Lam protocols: these work by having a nonce confused for an encrypted component, and so will not succeed in all implementations. Abadi and Needham [1] report a rather pleasing man-in-the-middle attack against these protocols, not described by Clark and Jacob. We independently rediscovered these latter attacks.

3.8 Shortcomings

We were able to model the Gong Mutual Authentication Protocol using Casper; however, attempts to analyze the protocol using FDR failed, because of insufficient swap space available (this protocol has a particularly large message space: at least an order of magnitude larger than any other in the library).

We do not model key compromise, which means we cannot find the well known attack on the Needham Schroeder Protocol with Conventional Keys, or a similar attack upon the Kao Chow Repeated Authentication Protocols. This accounts for four of the five attacks that we did not find.

We do not find Clark and Jacob's type flaw attack upon the Yahalom protocol, but we consider this attack to be erroneous, anyway. Brackin reports a weakness in the protocol, namely that an agent cannot be sure that a particular key is fresh; however, he does not describe how this weakness can be exploited, and we haven't found any way to do so, either.

4 Conclusions

In this paper we have reported on the analysis of a library of security protocols. We are pleased with our results: we have found nearly all the known attacks, and several more to boot. We also believe that we have demonstrated that this is a very practical and easy to use analysis technique, within the capabilities of undergraduate students, and hence of computer professionals.

These case studies have identified a number of areas in which Casper needs to be extended. We intend to extend Casper to model key compromise in the near future; modelling this in CSP provides no great challenge—the key is simply passed to the intruder at the end of the run—and so we expect this extension to Casper to be reasonably straightforward.

A number of protocols are designed to provide repeated authentication. These are typically in two stages: (1) an initial authentication, during which a key and ticket (or key certificate) are established; (2) repeat authentication, which may itself be repeated several times, where the ticket is used to re-establish authentication. Casper currently has no way of specifying that part 2 may be repeated arbitrarily many times using the ticket established in part 1. We are able to model the two parts separately, or to model a system that runs part 1 followed by a single instance of part 2; this allowed us to find the attacks upon the Neuman Stubblebine and Kehne Langendorfer Schoenwalder protocols. We intend to extend the Casper syntax to allow the user to specify that the protocol should repeat the repeat authentication phase (possibly with fresh nonces), so as to enable a more complete analysis of such protocols.

We were able to analyze the Encrypted Key Exchange (EKE) protocol; we found a rather weak replay attack, but no attack that led to the compromise of the session key. However, this protocol is designed to operate in the presence of an intruder who is able to guess poorly chosen passwords, and then try verifying his guesses by interacting with the protocol; this is a somewhat stronger intruder than the one normally considered. We consider that modelling such an intruder presents an interesting challenge to all current protocol analysis techniques.

Acknowledgements

This work was supported by grants from the UK Engineering and Physical Sciences Research Council, the UK Defence Evaluation and Research Agency, and the US Office of Naval Research.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. Research Report 125, Digital Equipment Corporation Systems Research Center, 1994.

[2] S. H. Brackin. Evaluating and improving protocol analysis by automatic proof. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, 1998.

[3] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available via `http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[4] T. Hwang, N.-Y. Lee, C.-M. Li, M.-Y. Ko, and Y.-H. Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letters*, 53:103–107, 1995.

[5] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

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

[7] G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`.

[8] G. Lowe and B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[9] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[10] F. J. Thayer Fábrega, J. C. Herzog, and J. D. Guttman. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop*, 1998.