

forward

a future of reliable wireless ad hoc networks of roaming devices

Protocol Synthesis Feasibility Report

June 30, 2003

Record of Changes

Date	Version	Comment
30-06-2003	1.0	First Issue

Authorisation

Dr. Gavin Lowe
FORWARD Steering Committee Member
PP. Dr. Sadie Creese
FORWARD Steering Committee Member

Date

Authors

Michael Auty, Oxford University, mike.auty@comlab.ox.ac.uk

Gavin Lowe, Oxford University, gavin.lowe@comlab.ox.ac.uk

Executive Summary

This report forms deliverable D2 of the FORWARD project. It is the first deliverable of Work Package 1 Authentication and Key Management. The aim of Work Package 1 is to enable key management solutions for the Next Wave, by investigating the feasibility of novel authentication and key management solutions and by enabling the design and assessment of authentication and key management systems for the pervasive paradigm.

The report provides a review of the work conducted in the first six months of research. The research is directed towards developing a single protocol which can be used in a variety of situations, each with a different set of security requirements, effectively synthesizing new protocols as situation requires.

The report opens explaining the ideas behind such a protocol, the various areas that will need investigation, and the areas that have been investigated in this report. It continues with a literature review covering the various areas examined so far. The paper then examines the problem of security requirements and concludes that the main five requirements of integrity, authentication, confidentiality, non-repudiation and anonymity are perhaps the only security requirements that matter.

The report continues with a dissection of a well known protocol, the Transport Layer Protocol (TLS) better known as Secure Socket Layers (SSL) which is used in most online personal transactions to provide confidentiality and authentication. The protocol is first simplified, then dissected and finally reconstructed from its components. Areas of concern with the TLS protocol are discussed, most notably those of the content carried over the higher data layer.

Finally the report discusses the possibilities of protocol composition and the outstanding difficulties that have yet to be tackled.

Contents

- 1 Overview** **1**
- 2 Partial Literature Review** **2**
- 3 Security Requirements** **3**
- 4 Transport Layer Security Protocol Research** **4**
 - 4.1 Simplification 4
 - 4.2 Protocol Deconstruction 5
 - 4.3 Rational Reconstruction 6
 - 4.4 Difficulties with TLS 7
- 5 Composability** **9**
 - 5.1 Granularity 9
 - 5.2 Outstanding Problems 9
- 6 Conclusion** **10**
- A Related Work** **11**
 - A.1 General security 11
 - A.2 Protocol proof 11
 - A.2.1 Belief Logics 11
 - A.2.2 Direct Proofs 12
 - A.2.3 State Exploration 13
 - A.3 Transport Layer Security Protocol 13
 - A.4 Protocol Synthesis and Composition 13
- B Simplification of TLS 1.0** **16**
 - B.1 Notation 16
 - B.2 Initial Protocol Description 16
 - B.3 Initial Modifications 17
 - B.4 Extraneous Constants 18
 - B.5 Removing Hashes 19
 - B.6 Coalescing Duplicates 20
 - B.7 Cleaning Up Hash Functions and Expanding 20

This page is intentionally blank

1 Overview

This research is aimed at producing a framework for composable security protocols by distinguishing the various security requirements of a protocol in the form of a specification and attempting to define rules for their composability. The eventual goal is to produce a single protocol whose abstract security requirements can be determined (and altered) at runtime and the protocol adapted to provide those requirements during transmission. The single protocol can then be implemented in embedded chips, in standard libraries, without need to recreate a new protocol for each program intending to make use of it. It will effectively synthesis new protocols at runtime, based directly upon the requirements of the situation (as specified by the user). Since only the abstract requirements would be specified by the protocol, implementations of the protocol could include “modules” which provide different methods of achieving the same goals. This however is currently a distant goal, and the first aim of the research is to develop a method for breaking protocols and their associated proofs down into smaller chunks which can be successfully composed without the need to reprove the entire composed protocol.

Various areas of interest in the research already include:

Requirements Have all the possible needs of the protocol been determined? What model will be used to define the interfaces between protocols?

Interaction Interaction between the various modules of the protocol is the main area for concern. Can key material be shared? Can earlier protocol data be included within payload data?

Negotiation Determining which implementations of which modules should be used for a particular run will require a great deal of thought and proof to ensure it is not a weak link in the protocol. How will negotiation work? Will negotiation be fair? What is the correct balance of allowing the user choice over protecting the user from themselves?

Granularity How big is a module? Can modules be made up of smaller modules? What are the implications of doing this?

Extensions Once the protocol is made, how flexible is it? What type of modules may be developed in the future? Will the protocol allow self-updating from trusted sources?

This first report on the research includes: a partial literature review 2; an initial examination and dissection of a commonly used protocol (TLS 1.0, Section 4), which is a prime example of two “logical” requirements, those of authentication and confidentiality; a rational reconstruction of the protocol (Section 4.3), to show that most protocols are developed by achieving one goal, then the next, and could be reworked in a modular format; and comment on various problems with the TLS protocol (Section 4.4) which also apply to layering and modularization in general, as well as some comment on the granularity and requirements of this compositional framework (Section 5). Appendix A lists the articles read in the course of the research to date and comments on the possible uses for each, whilst Appendix B includes a complete simplification of the TLS protocol.

2 Partial Literature Review

The very first requirement for ensuring that a framework for composing protocols will be viable is to examine exactly what various protocols exist and what they are designed to achieve. The books [Sch96], [And01] and [FS03] provide a number of esoteric protocols, but identify that the main requirements for security protocols in use today are confidentiality and authentication. A few other uses came to light also, including non-repudiation, and anonymity.

Once the requirements have been established, we will need a model to work in, both for creating the specifications that the composable modules must adhere to, but also for proving the correctness of the modules, and of the framework itself. Various models presented themselves, from the models involved in state space analysis ([Low98], [RSG⁺00]) through to the belief logics ([BAN89], [GNY90], [SvO96]). The model that seemed to provide the most flexibility however, was strand spaces ([THG99]). This model has already had various extensions ([GF01]) and theorems proved with it, and seems to provide the ability to give complete proofs that state spaces cannot, whilst not suffering the limitations of the logics.

Protocol composition, in the style proposed for this research, does not seem to have been examined in the past. [Sch96] seems to view protocols in a modular fashion, providing small protocols and cryptographic mechanisms for achieving certain minor goals (such as key exchange, bit commitment, fair decisions, etc.) but still requires the protocol designer to determine the specifications of the blocks, and also prove their composed security. Several small protocols ([DH76]) have been adapted and now find use in other many larger protocols (such as TLS, SSH), but each is again being individually examined for flaws ([WS96]).

One of the main difficulties with composing protocols appears to be the interaction of protocols, both different protocols ([Rus95]) and also different versions of the same protocol ([WS96]), where backwards compatible protocols are forced to run as earlier vulnerable versions of the same protocol. Proposed solutions to avoid the various attacks ([HLS00], [GS95]) are still far from being a total solution.

A full list of the papers read, with a summary regarding their possible uses, is included in Appendix A.

3 Security Requirements

Security requirements centre around data available between participants. In the general model where each entity has an identity and a stream of messages (either being received or sent), the security properties can be broken down as follows:

Integrity (a subcomponent of authentication) The messages entity A sent to entity B were not modified in transit.

Authentication Entity A sent messages to entity B and both are certain of the identity of the other participant, and aware that the messages received were those intended by the sender.

Anonymity Entity B is not aware of (or cannot prove exactly what is) entity A 's identity.

Non-repudiation Message m allows A to prove that a particular event occurred with participant B .

Confidentiality Only entity B can learn the payload of the messages sent by entity A .

These can of course be broken down into further sub-requirements (authentication alone can have several different meanings, as can non repudiation). However, there do not seem to be any further requirements (ignoring efficiency) upon a protocol. Further research must be done into the models of security and communications protocols, to form some suitably close model of actual communications protocols (including methods of identification, and addressing schemes).

4 Transport Layer Security Protocol Research

The initial research centred around a proof of the Transport Layer Security (TLS) protocol to achieve a given secrecy condition. The TLS protocol is a good choice for dissection, since it is an industry standard used by web browsers in almost every online financial transaction in the world today. The TLS protocol can be seen as a composition of components and as such a specific instance of the many possible combinations available to a truly generic composable protocol. TLS provides (or is believed to provide under certain circumstances) confidentiality and mutual entity authentication. Other protocols can then be layered on top of this, making use of the properties of the underlying protocol. The research was firstly to attempt to prove this fact, and secondly to dissect the protocol and rationally recreate it using its key elements. The protocol was too complex initially to approach, since it had been designed to cope with not only protocol attacks but also cryptographic attacks. It remains to be seen whether this is over engineering for peace of mind or for real security gains. The first step was therefore simplification.

4.1 Simplification

The idea of simplifying transformations is to remove redundant information present in the protocol whilst not losing any possible attacks. The end result should be a fault-preserved version of the original protocol, whereby any attack on the original is also an attack upon the simplified version. When the simplified version is then checked for attacks, if any are found they can be tested against the full protocol to see if they are indeed an actual attack. If the simplified protocol can be proven secure, then the original protocol is also secure. Further details on fault-preserving simplifying transformations can be found in [HL01].

The simplification managed to remove much redundancy, which obscured the logical reasoning behind the protocol design. Once all the simplifications have been applied the protocol is reduced to the following form. The complete simplification is available in Appendix A.

Encryption is as per standard notation, where m encrypted under key k is denoted $\{m\}_k$, which must be decrypted by k^{-1} (where k^{-1} is not necessarily equal to k). Concatenation is simply denoted by g, h . For clarity the abbreviation $prev_n$ is used to denote the concatenation of the messages from 1 to n inclusive. Throughout the protocol there are various functions which could not be abstracted away, including G_n and prf_x which are both based upon standard hashing functions (similar to $Hmac$ which is the standard HMAC function).

Message 1 $C \rightarrow S : R_c$
Message 2 $S \rightarrow C : R_s$
Message 3 $S \rightarrow C : \{PK(S), S\}_{SK(V)}$
Message 4 $S \rightarrow C : V'$
Message 5 $C \rightarrow S : \{PK(C), C\}_{SK(V')}$
Message 6 $C \rightarrow S : \{P\}_{PK(S)}$
Message 7 $C \rightarrow S : \{prev_6\}_{SK(C)}$
Message 8 $C \rightarrow S : [prf_{cf}(P, prev_7), 0]_{CM, CE}$
Message 9 $S \rightarrow C : [prf_{sf}(P, prev_7), 0]_{SM, SE}$

where

$$\begin{aligned}
 [M, n]_{MAC,ENC} &= \{M, Hmac(MAC, \langle n, M \rangle)\}_{ENC} \\
 CM &= G_0(P, R_c, R_s) \\
 SM &= G_1(P, R_c, R_s) \\
 CE &= G_2(P, R_c, R_s) \\
 SE &= G_3(P, R_c, R_s)
 \end{aligned}$$

The protocol starts by the client, C , sending a nonce (for session freshness) and the server, S , doing the same. Next S sends his certificate and a list, V' , of validators (certificate authorities) which it will trust. C responds with his own certificate, the premaster secret (P) encrypted under the server's certificate public key, and the previous messages encrypted under his own certificate's private key. This last message is to prove to S that the sender has knowledge of C 's private key. Finally S and C send the previous messages of the protocol to demonstrate knowledge of the various keys involved and to counter possible active attacks. After this handshake is complete, the two parties can continue sending messages using the established encryption and authentication keys. The messages are also numbered so that the message sequence can be established should there be packet loss (intentional or accidental) during transit. The protocol seems to have several complexities such as the use of V' to allow the client to choose a certificate which will be valid for the server, and the "finished" messages for active attack detection. We shall see shortly, however, that the protocol can be easily understood and decomposed into its component parts.

4.2 Protocol Deconstruction

The following steps can now be seen within the simplified protocol. They are easily extracted from the simplified protocol, although as shown below some messages have two distinct uses within the protocol (and thus are extracted and used twice later on).

Mutual Certificate Exchange Mutual certificate exchanges is designed to provide each entity with a certificate which it can associate with the other participant in the protocol. These certificates must be verified by each party, and their trust determined. If trusted the protocol can proceed; if not the protocol must terminate.

The V' term in message 4 is merely to list which certificate servers the server will accept, if the intruder attempted to modify this value, the effect would be the client might try to authenticate with a certificate signed by a CA the server didn't trust, in which case the server would abort and communications would be stopped. This message is thus not critical to the protocol. Indeed the certificate transfer messages are not critical either, since they could be pre-delivered by any PKI mechanism. The solution in SSL is literally to transmit the certificates and allow each participant to validate the certificates against the Certificate Authority (V or V') and determine their trust, which appears to adequately fulfil the requirements.

Message 3 $S \rightarrow C : \{PK(S), S\}_{SK(V)}$

Message 5 $C \rightarrow S : \{PK(C), C\}_{SK(V')}$

Client Authentication to Server The requirements for client authentication to the server are that the server can be certain it is running the protocol with the client and that the client is currently running the protocol. This is most often reduced to proving that the client possesses the private key corresponding to the certificate's public key presented earlier by encrypting some fresh nonce.

An additional requirement is often that the server can be certain the client "believes" it is running the protocol with the server. This does however stray mildly into the area of the server

making assumptions about the client's state and beliefs, which should be approached cautiously. It should be noted that this certainty can only be achieved if the client is an honest client and includes the identity it is communicating within the returned response. TLS/SSL automatically copes with this as part of its anti-active attack mechanism.

Message 2 $S \rightarrow C : R_s$

Message 7 $C \rightarrow S : \{prev_6\}_{SK(C)}$

Here the server sends out a random fresh variable (a nonce) and receives it transformed by the client who signs it with his secret key, thus proving he has access to the secret key and is running the protocol freshly. TLS in fact includes all previous messages to avoid any form of active attack (both parties can verify that their messages to date match). However, for the authentication the important elements are R_s and some elements to prove to the server who the client believes it is running with (catered for by $prev_6$).

Server Authentication to Client Similar requirements apply for server authentication to client as to the client authentication to server above.

Here the client generates a random value P which can be known only by the possessor of the server's private key (in the full certificate). Since it is fresh, if the server makes use of the value (as an encryption key, and within the authenticating MACs) it proves that the server is running in the current session and that it is the intended server (as verified by the earlier certificate and now proof of knowledge of that certificate's private key). Again $prev_7$ is used to stop any active attack (and is thus beneficial and necessary to stop attacks).

Message 6 $C \rightarrow S : \{P\}_{PK(S)}$

Message 9 $S \rightarrow C : [prf_{sf}(P, prev_7), 0]_{SM,SE}$

Confidentiality of Data Confidentiality of data requires that no one other than the two participants can determine the contents of the encrypted data passing between them and that confidential data is only passed between them once authentication has been established.

This is provided by confidentiality of a single secret, which is then used to generate and encrypt (thus keeping confidential) a larger data stream.

Message 6 $C \rightarrow S : \{P\}_{PK(S)}$

The secrecy of the initial secret is established by simple use of the public/private key mechanism inherent in certificates. Note this initial secret passed between the participants is also the client's nonce. This may suggest a conflict of interests and the possibility of an unfair protocol (whereby the client can in some way influence the protocol, possibly with an eye to an attack), but since the final data used to form the encryption and decryption keys include the server's random nonce, there should not be a problem.

4.3 Rational Reconstruction

The protocol can thus be rebuilt by using the various components, and in fact, streamlined. The reason so many of the protocol messages can be left out is that this provides only protocol security and not also cryptographic security. The protocol now also does not allow for negotiation of various crypto suites or compression suites. It should be noted that since the value P is used both as the client authentication nonce, and also as the primary secret for confidentiality message number 6 has been included twice, once as part of the authentication subcomponent and once as part of the confidentiality subcomponent. This also shows that in the original protocol, the client's nonce was completely superfluous.

The rebuilt protocol is thus (message numbers from the original protocol are in brackets):

Message 1(3) $S \rightarrow C : \{PK(S), S\}_{SK(V)}$
 Message 2(5) $C \rightarrow S : \{PK(C), C\}_{SK(V')}$
 Message 3(6) $C \rightarrow S : \{P\}_{PK(S)}$
 Message 4(2) $S \rightarrow C : R_s$
 Message 5(7) $C \rightarrow S : \{prev_4\}_{SK(C)}$
 Message 6(6) $C \rightarrow S : \{P\}_{PK(S)}$
 Message 7(9) $S \rightarrow C : [prf_{sf}(P, prev_6), 0]_{SM,SE}$

Further simplifications include order optimizations to allow the server to clump its various messages together. The message numbers from the above protocol are in brackets. We now also leave out one of messages 3 and 6 from the above protocol, since one will suffice.

Message 1(1, 4) $S \rightarrow C : \{PK(S), S\}_{SK(V)}, R_s$
 Message 2(2, 3, 5) $C \rightarrow S : \{PK(C), C\}_{SK(V')}, \{P\}_{PK(S)}, \{prev_1\}_{SK(C)}$
 Message 3(7) $S \rightarrow C : [prf_{sf}(P, prev_2), 0]_{SM,SE}$

That allows the protocol to effectively be reduce to a three message protocol in a similar order to mutual authentication protocols. It also manages to achieve confidentiality of the session data, and in the combination of authentication and confidentiality, ensures the data sent from one participant to the other is integral and unmodified.

4.4 Difficulties with TLS

We initially attempted to prove that the TLS protocol provided the following security properties in relation to the higher transport layer:

Authentication The message stream received by a participant is a prefix of the messages sent to them by the other participant.

Secrecy An intruder could not learn any data being sent over the transport layer unless he received it directly.

The Authentication property not only captures that the messages were sent by the other participant, but also that they arrive in the correct order and without gaps. This is important to authenticate not only the messages themselves, but the entire stream. The proof of these conditions is currently work in progress but we have already discovered that TLS can't be used indiscriminately.

It turns out that the TLS protocol will break unless certain sanity rules are in place. One rule is that the transport stream should not be used to transmit either long-term secret keys or session keys. Further, messages sent on the transport stream should not take the same form as the lower level TLS messages, or else the two protocols could interfere with one another, as demonstrated by the following example (as discovered by Gavin Lowe). Suppose there is a higher layer protocol which uses the following messages:

Message 1 $I \rightarrow S : \{P\}_{PK(S)}$
 Message 2 $S \rightarrow I : P$

The underlying TLS protocol is used to establish a connection from C to S , during which the intruder could capture message 6 from the TLS handshake. Later the intruder could establish his own TLS layer with S (using his own identity) and then replay $\{P\}_{PK(S)}$ as message 1 of the "secured" higher layer protocol, so as to obtain P .

Of course this attack is not limited to TLS but can include any two protocols that share message spaces or key spaces. If the two protocols are not message space disjoint (or more accurately term space disjoint) then an unintended transformation by use of a key which spans two protocols could result in security breaches for one or both of the protocols in question. In other words, elements from one message in a protocol might be put to malicious use in an all together different protocol. Protocol identifiers in each term will ensure that a different protocol's terms can be clearly identified by any other protocol. When designing composable protocols this must be considered with extreme care as there can be no telling what other protocols may be running alongside our composable protocol.

5 Composability

Composability itself has many different factors affecting it, not least of which is how large to make the composable systems. The granularity of the system, and further problems that have yet to be tackled are covered below. Module negotiation could prove the hardest task within the area, but this has not yet been explored at all, and will most likely be delayed until an adequate framework for requirements has been developed.

5.1 Granularity

The granularity of our composition framework is currently undecided. There is an argument to be made for very small building blocks, which can individually be analysed and proved fairly simply and then carefully composing these small blocks together. The downside to having such small blocks is that there will naturally be a lot of different interfaces between the components. Developing a proof framework to allow these blocks to then be composed as required will be a difficult challenge.

An alternate approach, and the original direction for this research, is to produce large abstract modules with their own interfaces, and a few large proofs to show that these few interfaces can be composed together in a particular order. This eases the burden of proving details concerning very small blocks, and allows individual module designers to develop their own modules (complete with proof that they implement the interface required). It also allows implementations to be replaced if they are deemed no longer secure. It does however involve the very large task of proving the interface connections or “glue” to stick the blocks together. Not only this, but the module negotiation protocol will be an added complication to the task. This does however seem to be extremely beneficial and more useful in practical situations than small components. For the bulk of this document, the granularity has been that of large modules unless otherwise stated.

It may be possible to have a framework encompassing both sizes, spanning small blocks of a few messages, up to complete protocol solutions (perhaps even made up of smaller blocks themselves).

5.2 Outstanding Problems

With the problem of mixing possibly unknown communication data streams, it becomes clear that every module must adhere to some rules. To combat cross-protocol attacks it has been hypothesized that each protocol should be term space disjoint with itself (i.e. no complex term within the space of possible messages to a participant can be mistaken for any other complex term of any other protocol). The simplest way to achieve this appears to be including a protocol term identifier. [GT00] suggests that this will make each term within any message unique to that protocol and thus make protocols, even with the same terms, independent.

Also sending key material for a stream within the payload of that stream is inadvisable; it may be necessary to deem certain information within the protocol state sensitive and as such never to leave the protocol state (either through the stream or to any other part of the system upon which the protocol is running). Another consideration is cross protocol key usage, whereby one protocol accidentally decrypts a message from another protocol. This shouldn't happen under an integral stream, since each message contains a protocol name and number, making messages cross protocol disjoint; however setting up the integral stream will still have to deal with this. As such, another constraint may be that keys must be used only within a single protocol (i.e. the system has a single key for use with a particular protocol block, say confidentiality by RSA, and requires another key for a different protocol block, say authentication via a DSA key). This seems both a prudent and plausible restriction but it also has not been fully researched and may possibly be unnecessary depending on the final specifications reached.

6 Conclusion

The research is proceeding well, and appears to be both new and potentially extremely useful. The dissection of the existing protocol has shown that most protocols are designed in a goal by goal strategy, and the literature review has shown that smaller protocols are almost always simpler to analyse and prove. The literature review has provided little pertaining to composability and it appears the approach of complete composability (as opposed to composition followed by additional proof) has not been explored much, if at all. Finally, the research into security requirements has turned up the fact that there are surprisingly few possible requirements, considering the number of protocols that have been developed to tackle the same problems over and over again. Currently there seems no reason to believe that protocol composition by modules will not be possible.

Research ahead includes a good model of security, which appears to be vital and is the very next step in the research. So far it appears that security protocols can be deconstructed with ease once clearly understood, and further research will attempt to build a library of decomposed protocols and abstract their commonality to provide a specification for each of the requirements. Once the requirements and proof framework have been established, the interaction problems will need addressing and this will be the next step. Granularity will probably determine itself during the course of the research. Finally advanced areas such as negotiation and modular extensions will be the final area of research in this task.

A Related Work

The following is a partial literature review concerning the area of computer security. Some of the papers are geared directly towards composability but these are few and far between; the majority of the papers concerning proof methods and models of security. The remaining literature to be read for the next deliverable is liable to include more specific papers on composability and synthesis.

A.1 General security

This section discusses various books that have provided a broad overview of computer security, and within that field focusing on protocol security.

Applied Cryptography [Sch96] This provides a useful overview of security protocols. Whilst a fair proportion of the book is dedicated to the cryptographic underlayings of security, it also provides information on many security requirements, including some more esoteric goals which are often neglected by standard protocol analysis and synthesis techniques, such as anonymity and non-repudiation. The book also attempts to provide certain cryptographic primitives which may in turn spawn interesting ways of achieving the desired goals including blinding, zero-knowledge proofs and threshold schemes.

Security Engineering [And01] This book attempts to provide much more of a mind set about security in general, and whilst introducing useful threat models (such as Bell-LaPadula), it tends to stray too far into physical security. This makes only the first few chapters relevant to the computer security area at all. That is not to imply that the book has no useful ideas, certainly ideas from the physical communications security world may well be adaptable to the world of protocols, such as frequency hopping and direct sequence spread spectrum. These make jamming difficult by diffusing the message throughout the bandwidth; perhaps they might be applied to anonymity or steganographic modules.

Practical Cryptography [FS03] The authors attempted to create a guide to good programming techniques when developing protocols. They provide solutions for cryptographic difficulties, which will come in useful to ensure that the project is not only theoretically correct but practically useful as well. An underlying theme was modularity and as such the book provides a good insight into the practical considerations of the composition of security properties. It explains how the authors would build a secure channel (which would equate to an authenticated confidential stream in the context of the current research) by composing the authenticity and encryption properties and stating what their preferred ordering would be and why. During talks with one of the authors, it became apparent that they believe certain security properties can not be entirely independent.

A.2 Protocol proof

An understanding of protocol proof methods will be critical to this research. Once a model for specifications has been developed, it will be required that layering one protocol block on top of another protocol block can be proved not to interfere with either's ability to comply with its specification. These will also be used eventually by protocol block designers to prove their blocks fulfil the specifications required.

A.2.1 Belief Logics

Belief logics reason about what each participant in a protocol believes and attempts to prove certain hypotheses based upon the logic and model of beliefs used to conduct the proof. Belief logics could be useful to this research, as we will require some model of security to specify each security goal

precisely, and then some method of proof for our compositional system. One of the most powerful, but most dangerous, tools belief logics can provide is that of intent for information; in state space based models, intent is almost never considered, which makes it on the whole much more difficult to make a mistake, but also makes it difficult to create precise and concise specifications for properties.

A Logic of Authentication [BAN89] The classic paper on belief logics, the BAN logic described within it provides one of the first belief logics developed. This method of proving security protocols provides a seemingly more complete method of proof than state space analysis, which sometimes is unable to include circumstances such as multiple protocol runs. However, as demonstrated by this logic, the proof only extends as far as the model upon which the logic is based. This logic was used to prove the Needham-Schroeder public key authentication protocol, which was later proved to have a subtle flaw, undetected in the original proof.

Reasoning About Belief in Cryptographic Protocols [GNY90] The Gong, Needham and Yahalom belief logic attempts to extend the BAN logic from the paper above to require fewer assumptions and a few extra features. Essentially this is just an improvement on the standard BAN logic and whilst fixing some flaws was still not a globally useful logic.

A Unified Cryptographic Protocol Logic [SvO96] Syverson and van Oorschot attempted to correct the previous flaws found within the BAN logic and other similar logics. This new unified logic comes closest to coping with all the concerns over previous belief logics and as such would be the best choice for a belief logic if one is needed.

A.2.2 Direct Proofs

Strand spaces provide an interesting framework within which to prove protocol correctness. This model can be used not only to specify properties but also to prove them, and as such is particularly versatile.

Strand Spaces: Proving Security Protocols Correct [THG99] This paper introduces the basic notion of strand spaces, by modelling protocols as a directed graph with both send and receive nodes. Two types of edges, temporal (between nodes of the same participant strand) and causal (nodes of transmission, one send and one receive) are then introduced. From this basis, statements can be constructed to determine uniqueness and origination of values (i.e. which nodes of the graph causally occur before any others). This form of analysis is almost a halfway house between model checking (since large bundles can be considered and multiple runs can be taken into account) and belief logics (where assertions can be made about what happens to a particular value, rather than having to test the value in all locations and see if something breaks). As such this may provide the best framework to prove statements concerning the composability of these protocol layers/modules.

Authentication Tests and the Structure of Bundles [GF01] This paper extended the idea of strands providing several primitives for use within the strands world, without having to develop terminology or proofs for trivial but repeatedly used features. It also introduced the idea of authentication tests which provide one of the most easily understood explanations of what is required for authentication that I've read so far. It suggests that authentication of another entity occurs when that entity knowingly transforms a piece of (fresh) data in some way that only it could do. The paper then further breaks this down into effectively encryption and decryption transformations. The difficulty arises in proving the other entity transformed the data knowingly. This indicates that context and intent will feature strongly in the final research and must be thought about and incorporated at an early level.

A.2.3 State Exploration

State space analysis overcomes one of the main short comings of proof techniques by being mostly automatic. As long as the correct space is defined (which can be done using tools), the space itself is analysed by programs. There are no hand proofs, inventions or other tricks required to give a proof. Sadly state space analysis is also limited to simple systems and protocols with few variables due to the computational power required to examine the entire space. FDR and Casper are one such pair of tools, Casper takes a simple protocol description and develops the full specification (defining the space to be searched, and what to search for) in the language of Concurrent System Processes (CSP) and FDR performs the actual analysis.

Casper: A Compiler for the Analysis of Security Protocols [Low98] Casper is fairly usable (and widely available) unlike several other protocol analysis tools, which makes it useful in this research as a good verification that composed protocols are indeed secure. It may also serve a purpose in the intermediate term, containing a large amount of parsing code for standard protocol notation. This may be used to create CSP models of various protocol aspects. It may well turn out that CSP is a good language to conduct the modelling, although the existing model used by Casper lacks useful extra information such as addressing information, so this is still under investigation.

Modelling and Analysis of Security Protocols [RSG⁺00] This book elaborates upon the details of the CSP model used by Casper and includes interesting background material relating to other state space analysis tools as well. If CSP is used as the final model upon which to base the specifications, this book will prove extremely useful. Lastly it contains a good description of the various security requirements known in general (although not formally). These provide a good basis for developing a formal specification of the requirements.

The NRL Protocol Analyzer: An Overview [Mea96] The NRL protocol analyzer, whilst primarily a state space analysis tool, has elements of direct proof. It was designed in prolog and as such can make use of logic language and can thus make inductive proofs. This is merely another example of state exploration techniques and will probably have no bearing on the research of this project directly.

A.3 Transport Layer Security Protocol

The TLS 1.0 protocol has featured prominently in the research to date for several reasons. The protocol achieves multiple security requirements (such as confidentiality and authentication), it is a de facto standard for secure communications within the industry, and it demonstrates some of the streaming/layering ideas this research intends to examine. The TLS protocol will provide a multi-goal protocol for dissection into the relevant components.

Analysis of SSL 3.0 Protocol [WS96] Whilst TLS 1.0 had minor changes from SSL 3.0, this analysis is still extremely relevant to the protocol, pointing out flaws in such a versatile protocol. The attacks suggested included version rollback attacks to previously compromised versions of the protocol, and a flag message deletion attack, where a message being used as a flag to change state was dropped by the attacker. The first of these attacks must be considered carefully when developing the module identification system, such that no attacker can force a legitimate connection to use a compromised module.

A.4 Protocol Synthesis and Composition

Composing protocols is not an easy task, and trying to compose any protocols is not necessarily possible. Protocol synthesis approaches the question from a different angle and says, if we take

extremely restricted and predictable analysed protocols, can we compose them to produce secure protocols? This method forces a protocol to be not only secure, but also have rigidly defined properties so that when composed no subtle attacks can sneak in. The research is really directed towards developing a protocol that can be composed at whim, such that when new requirements appear, they can be added during transmission. As such the composition of protocols is the essence of the research.

How to Prevent Type Flaw Attacks on Security Protocols [HLS00] This paper introduces the idea of tagging data items, such that any message received can be unambiguously parsed back into its component parts. This again will require careful consideration when developing the modular framework, to ensure that no message received by a module can be misinterpreted, no matter its origins. Note this does not stop an attacker who can create messages which validate correctly, but should stop attacks where the message cannot be entirely or accurately modified, for example where the contents of the message are not known.

Fail-Stop protocols: An approach to designing Secure Protocols [GS95] This paper divides a method of detecting an active attack against a protocol immediately and ensuring that no extra information is leaked to the attacker once the attack has been detected. The authors provide particular fields which must be encoded into every message in an attempt to detect an active attack, and these may prove the basis for a generic envelope that will be required of each protocol during transmission.

New Directions in Cryptography [DH76] This paper by Diffie and Hellman, introduces the Diffie-Hellman Key Exchange protocol, which can provide a session key to two parties. This is most commonly used to establish an encryption key before determining whether the other party is in fact that party they say they are. This led to the idea of splitting an integrity property from the main authentication property. Further thinking in this area has not yet revealed a reason to provide integrity without some knowledge of with whom the communication is taking. It still seems however to be an important enough step within the authentication to demark it as a subcomponent and reason about it separately from authentication.

Combining System Properties: A Cautionary Example and Formal Examination [Rus95] This provides a seemingly pessimistic message that composing two protocols which both fulfil specifications, does not ensure that the composition of the two achieves the composition of the results. The examples used are an Alternating Bit Protocol and a Checksum Protocol, one of which converts a lossy stream into a reliable stream, and the other a corrupt stream into an error free stream. The caution mentioned in the title is that when composed, in either order, the protocols do not convert a lossy corrupt stream into an error free reliable stream. On closer examination this is obvious, since the precondition of the new scenario (a lossy corrupt stream) into which these protocols have been placed, is not as strong as the preconditions required by either of the subprotocols individually. This paper highlights the necessity for proper module specification and a good underlying model of the system in which the modules are expected to act. It also illustrates that the layering of protocols is most straight forward when sequential rather than combinatorial.

A Compositional Logic for proving properties of security protocols [DMP01] This paper initially sounded promising as a possibility for proving compositional protocols; however the paper focuses more on the creation of yet another logic style. Whilst the first half of the paper discusses an interesting style of protocol description (referred to as cord notation, presumably for its similarity to strand spaces) which seemed designed from the outset with composition in mind, the second half of the paper fleshes out the proof logic and eventually only conducts the standard proofs of

the Needham-Schroeder Public Key Authentication protocol. Research mentioned in this paper is probably not mature enough to have any bearing upon this research.

B Simplification of TLS 1.0

B.1 Notation

“Standard” protocol notation will be used (m encrypted under key k is denoted as $\{m\}_k$ and must be decrypted by k^{-1} which is not necessarily equal to k , and where concatenation denoted by g, h), with the following additions:

For multi-input functions, concatenated terms (normally a, b) will be delimited as $\langle a, b \rangle$. For example, a two input function, f may take a and the concatenation of b and c as $f(a, \langle b, c \rangle)$.

All literal values will be noted using **bold face**.

B.2 Initial Protocol Description

The initial protocol description has the following components. All messages are wrapped within a record layer. This is made up of three fields: the first field specifies a sub-protocol type (22 being a handshake message, 20 being a ChangeCipherSpec message); the second specifies the protocol version (for TLS 1.0, this is always 3.1); and the third specifies the size of the packet length (specified for message x by a function $L(x)$).

Within the record layer, there are four different sub-protocols. The messages of the handshake sub-protocol (messages 1–8, 10 and 12) start with a message type identifier, then the relation L2 and then the payload in the form of the particular message.

Certain named components featured within the protocol include *CypS* and *CmpS* (the cypher-suites and compression-suites available), *SID* (the session identifier) and *Cyp* and *Cmp* the chosen cypher-suite and compression-suite.

The TLS protocol uses a pseudo random function (PRF), which takes as inputs a secret, a seed and an identifying label and produces an pseudo-random data stream. We will assume that the PRF is a hash function. Proving or disproving this is left as an exercise for cryptographers. Once this has been established, the hash functions can be composed and rewritten to provide other hash functions to reason about.

Another hashing method used commonly in the TLS protocol is first hashing with MD5, then hashing with SHA and taking a concatenation of initial portions of the two hashes. This will be replaced with a single verification hash function, *vh*.

Certificates initially will be unspecified, abstractly defined as a function taking two identities (a principal and a validator) and providing a public key, and giving the assurance that the corresponding private key is possessed only by the principal. The assurance can be trusted to the same degree as the validator, which may be different for each participant.

Two messages include hashes of the concatenation of all previous messages. To model this we will use the term $prev_x$ for all previous message bodies up to message x (and the term M_x for the x^{th} message).

Also of note is the quarter function $quarter(s, n)$ which takes an arbitrary length input and an identifier, such that knowledge of $quarter(s, n)$ reveals no information about $quarter(s, m)$ where $m \neq n$ and also reveals no extra information about s . This function is designed to cut the pseudorandom stream generated by the prf into segments for use as key material later in the protocol.

- Message 1 $C \rightarrow S$: **22, 3.1**, $L(1)$, **1**, $L2(1)$, **3.1**, R_c , $CypS$, $CmpS$
 Message 2 $S \rightarrow C$: **22, 3.1**, $L(2)$, **2**, $L2(2)$, **3.1**, R_s , SID , Cyp , Cmp
 Message 3 $S \rightarrow C$: **22, 3.1**, $L(3)$, **11**, $L2(3)$, $Cert(S, V)$
 Message 4 $S \rightarrow C$: **22, 3.1**, $L(4)$, **13**, $L2(4)$, **1**, V'
 Message 5 $S \rightarrow C$: **22, 3.1**, $L(5)$, **14**, $L2(5)$
 Message 6 $C \rightarrow S$: **22, 3.1**, $L(6)$, **11**, $L2(6)$, $Cert(C, V')$
 Message 7 $C \rightarrow S$: **22, 3.1**, $L(7)$, **16**, $L2(7)$, $\{3.1, P\}_{PK(S)}$
 Message 8 $C \rightarrow S$: **22, 3.1**, $L(8)$, **15**, $L2(8)$, $\{vh(prev_7)\}_{SK(C)}$
 Message 9 $C \rightarrow S$: **20, 3.1**, $L(9)$, **1**
 Message 10 $C \rightarrow S$: **22, 3.1**, $L(10)$, $[payload(\text{"client finished"}), 0]_{CM, CE}$
 Message 11 $S \rightarrow C$: **20, 3.1**, $L(11)$, **1**
 Message 12 $S \rightarrow C$: **22, 3.1**, $L(12)$, $[payload(\text{"server finished"}), 0]_{SM, SE}$

where in message x

$$\begin{aligned}
 [M, n]_{MAC, ENC} &= \{M, hmac(MAC, \langle n, \mathbf{22}, \mathbf{3.1}, L(x), M \rangle)\}_{ENC} \\
 payload(s) &= \langle \mathbf{20}, L2(x), prf(MK, s, vh(prev_8)) \rangle \\
 MK &= prf(P, \text{"master secret"}, \langle R_c, R_s \rangle) \\
 KE &= prf(MK, \text{"key expansion"}, \langle R_c, R_s \rangle) \\
 CM &= quarter(KE, 0) \\
 SM &= quarter(KE, 1) \\
 CE &= prf(quarter(KE, 2), \text{"client write key"}, \langle R_c, R_s \rangle) \\
 SE &= prf(quarter(KE, 3), \text{"server write key"}, \langle R_c, R_s \rangle)
 \end{aligned}$$

Within the record layer the protocol works as follows: the client sends out a nonce and a set of choices it is willing to accept concerning encryption and authentication mechanisms (message 1); the server then responds with its own nonce, a session identifier and the final decision on encryption and authentication mechanisms that will be used in the protocol (message 2); the server also sends its certificate and a list of validators (certificate authorities) it is willing to trust (messages 4 and 5); the client then sends its own certificate, a premaster secret encrypted by the server's public key, and the previous messages encrypted using its private key as proof of identity (messages 6–8). Finally both sides validate the choices made for the protocol by sending finished messages using the chosen encryption and authentication mechanisms and keys (messages 10 and 12). They include the previous messages to avoid active attacks. There are also various messages (5, 9 and 11) for indicating a change of state to the other party.

B.3 Initial Modifications

I intend to remove all of the information provided by the record layer and headings provided by the handshake sub-protocol. I will use the "Remove Atoms" method to delete all related fields throughout the protocol. This is a fault preserving transformation if the removed atoms are not required for agreement (and all items to be agreed upon are atoms).

All fields of type ContentType, ProtocolVersion and also ContentLength will be removed. The reasoning is as follows:

ContentType This merely signifies the type of message and provides parsing information. It has been proved that all type flaw attacks can be thwarted by tagging, see [HLS00]. This means that were there a type flaw attack on TLS, a new protocol (implementing tagging)

could be developed very quickly and cure the flaw. We will therefore ignore type flaw attacks for the rest of this document.

ProtocolVersion This information is included in many (some unusual) places in the protocol to avoid protocol rollback attacks. This simplification may introduce a rollback attack, however since this paper discusses only TLS 1.0 and does not allow for dynamic protocol decisions, this simplification should have no side effects.

ContentLength The length fields are provided to avoid truncation attacks and to provide proper parsing information. Again it will be assumed that all messages are parsed correctly and that truncation attacks can be detected (which can be made true by the type of encryption used for encrypted terms and would be signified by the absence of an atom for non encrypted terms).

Renaming of string literals For clarity the string literal terms will now all be renamed to atoms with shorter names: “server finished” with *sf*, “client finished” with *cf*, “master secret” with *ms*, “client write key” with *ck*, “server write key” with *sk* and “key expansion” with *ke*.

B.4 Extraneous Constants

From this protocol we can now remove “marker” atoms. These are string literal values such as “1”, which are used mainly to signify a particular type of message has been sent (such as change cipher spec). There was a weakness involving the ChangeCipherSpec marker message ([WS96]), but this was resolved in TLS 1.0 by requiring a valid Finished message directly following it. Since this only now signals to the other participant to send the next message (which we have fixed for this model of protocol), we can safely remove these. We will also remove all messages without content: since nothing is sent, they may as well not be present at all. Similarly since the cypher suite *CypS*, and compression suite *CmpS* and the server chosen values *Cyp* and *Cmp* will be removed since they are set (RSA certificate with RSA key exchange, no compression) for this description of the protocol. They would be required for agreement in the *prev* statements, however removing these should only remove a protocol negotiation attack, which we have stated have been ignored. Similarly since session resumption is not considered for this document (and the *SID* is only generated never used) it will be removed.

This gives the simplified protocol below:

Message 1 $C \rightarrow S : R_c, CypS, CmpS$
 Message 2 $S \rightarrow C : R_s, SID, Cyp, Cmp$
 Message 3 $S \rightarrow C : Cert(S, V)$
 Message 4 $S \rightarrow C : V'$
 Message 5 $C \rightarrow S : Cert(C, V')$
 Message 6 $C \rightarrow S : \{P\}_{PK(S)}$
 Message 7 $C \rightarrow S : \{vh(prev_7)\}_{SK(C)}$
 Message 8 $C \rightarrow S : [prf(MK, cf, vh(prev_8)), 0]_{CM,CE}$
 Message 9 $S \rightarrow C : [prf(MK, sf, vh(prev_8)), 0]_{SM,SE}$

where now

$$\begin{aligned}
 [M, n]_{MAC,ENC} &= \{M, Hmac(MAC, \langle n, M \rangle)\}_{ENC} \\
 MK &= prf(P, ms, \langle R_c, R_s \rangle) \\
 KE &= prf(MK, ke, \langle R_c, R_s \rangle) \\
 CM &= quarter(KE, 0) \\
 SM &= quarter(KE, 1) \\
 CE &= prf(quarter(KE, 2), ck, \langle R_c, R_s \rangle) \\
 SE &= prf(quarter(KE, 3), sk, \langle R_c, R_s \rangle)
 \end{aligned}$$

since the various constants were removed.

B.5 Removing Hashes

We conjecture that if prf is a hash function then $prf_c(\langle x, y \rangle) = prf(\langle x, c, y \rangle)$ is also a hash function where c is a constant or string literal. In other words that each keyed labeled prf , is in fact a distinct hash of the key concatenated with the data to be hashed. I can now remove prf_{ms} , prf_{ke} , prf_{ck} and prf_{sk} . prf_{ck} and prf_{sf} will not be removed since they allow for much simpler reasoning concerning the leaking of the mastersecret/premastersecret. Similarly where c is a constant, $quarter(a, c)$ can be replaced by some function $H_c(a) = quarter(a, c)$. Since there are no longer any multi-input functions, we will revert to standard notation for concatenation within functions. At this point we will also remove the verification hash, vh . This may open the simplified protocol to more attacks, but since each copy is encrypted, this should not cause too many problems.

$$\begin{aligned}
 \text{Message 1 } C \rightarrow S &: R_c \\
 \text{Message 2 } S \rightarrow C &: R_s \\
 \text{Message 3 } S \rightarrow C &: Cert(S, V) \\
 \text{Message 4 } S \rightarrow C &: V' \\
 \text{Message 5 } C \rightarrow S &: Cert(C, V') \\
 \text{Message 6 } C \rightarrow S &: \{P\}_{PK(S)} \\
 \text{Message 7 } C \rightarrow S &: \{prev_6\}_{SK(C)} \\
 \text{Message 8 } C \rightarrow S &: [prf_{cf}(MK, prev_7), 0]_{CM,CE} \\
 \text{Message 9 } S \rightarrow C &: [prf_{sf}(MK, prev_7), 0]_{SM,SE}
 \end{aligned}$$

where

$$\begin{aligned}
 [M, n]_{MAC,ENC} &= \{M, Hmac(MAC, \langle n, M \rangle)\}_{ENC} \\
 MK &= P, R_c, R_s \\
 KE &= MK, R_c, R_s \\
 CM &= H_0(KE) \\
 SM &= H_1(KE) \\
 CE &= H_2(KE), R_c, R_s \\
 SE &= H_3(KE), R_c, R_s
 \end{aligned}$$

B.6 Coalescing Duplicates

Firstly by swapping pairs (and substituting in MK), the KE assignment can be rewritten as $\langle KE := P, R_c, R_s, R_c, R_s \rangle$. The pair swapped is R_s, R_c which occurs only in the assignment to KE and now in messages 8 and 9 (at the start of $prev_7$). By the coalescing duplicates simplification (now that several layers of hashing have been removed) the duplicate values in R_c, R_c and R_s, R_s can be removed.

B.7 Cleaning Up Hash Functions and Expanding

The protocol has now been extensively simplified and the only remaining complexity is expressed in the assignments. To remove this complexity we will use the following functional mapping:

$$\begin{aligned} G_N(P, R_c, R_s) &= H_N(\text{prf}_{ke}(P, R_c, R_s)) && \text{if } N = 0 \vee N = 1 \\ G_N(P, R_c, R_s) &= H_N(\text{prf}_{ke}(P, R_c, R_s), R_c, R_s) && \text{if } N = 2 \vee N = 3 \end{aligned}$$

Finally, once all this is complete, we will define the necessary requirements for a Certificate. A certificate $Cert(A, B) = \{PK(A), A\}_{SK(B)}$, that is a certificate for A from B must be a key $PK(A)$, such that A and only A possesses $SK(A)$, and this and the identity of the owner is signed using $SK(B)$.

The final version of the protocol is as follows:

- Message 1 $C \rightarrow S : R_c$
- Message 2 $S \rightarrow C : R_s$
- Message 3 $S \rightarrow C : \{PK(S), S\}_{SK(V)}$
- Message 4 $S \rightarrow C : V'$
- Message 5 $C \rightarrow S : \{PK(C), C\}_{SK(V')}$
- Message 6 $C \rightarrow S : \{P\}_{PK(S)}$
- Message 7 $C \rightarrow S : \{prev_6\}_{SK(C)}$
- Message 8 $C \rightarrow S : [\text{prf}_{cf}(P, prev_7), 0]_{CM, CE}$
- Message 9 $S \rightarrow C : [\text{prf}_{sf}(P, prev_7), 0]_{SM, SE}$

where

$$\begin{aligned} [M, n]_{MAC, ENC} &= \{M, \text{Hmac}(MAC, \langle n, M \rangle)\}_{ENC} \\ CM &= G_0(P, R_c, R_s) \\ SM &= G_1(P, R_c, R_s) \\ CE &= G_2(P, R_c, R_s) \\ SE &= G_3(P, R_c, R_s) \end{aligned}$$

References

- [And01] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.
- [BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.
- [DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
- [DMP01] Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 241–255, 2001.
- [FS03] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. Wiley Publishing, Inc., 2003.
- [GF01] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 2001.
- [GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning About Belief in Cryptographic Protocols. In Deborah Cooper and Teresa Lunt, editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society, 1990.
- [GS95] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.
- [GT00] Guttman and Thayer. Protocol independence through disjoint encryption. In *PCSFW: Proceedings of The 13th Computer Security Foundations Workshop*. IEEE Computer Society Press, 2000.
- [HL01] Mei Lin Hui and Gavin Lowe. Fault-preserving simplifying transformations for security protocols. *Journal of Computer Security*, 9(1, 2):3–46, 2001.
- [HLS00] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 255–268, 2000.
- [Low98] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998.
- [Mea96] Catherine Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
- [RSG⁺00] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Pearson Education, 2000.
- [Rus95] John Rushby. Combining system properties: A cautionary example and formal examination. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, June 1995. Unpublished project report; Available at <http://www.csl.sri.com/rushby/combined.html>.

- [Sch96] Bruce Schneier. *Applied Cryptography (Second Edition)*. John Wiley & Sons, 1996.
- [SvO96] P. Syverson and P. van Oorschot. A unified cryptographic protocol logic. Technical Report 5540-227, NRL Center for High Assurance Computer Systems, 1996.
- [THG99] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2, 3):191–230, 1999.
- [WS96] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of The Second USENIX Workshop on Electronic Commerce*, pages 29–40. USENIX Press, 1996.