Security Protocols and Specifications

Martín Abadi
ma@pa.dec.com

Systems Research Center
Compaq

Abstract. Specifications for security protocols range from informal narrations of message flows to formal assertions of protocol properties. This paper (intended to accompany a lecture at ETAPS '99) discusses those specifications and suggests some gaps and some opportunities for further work. Some of them pertain to the traditional core of the field; others appear when we examine the context in which protocols operate.

1 Introduction

The method of "security by obscurity" dictates that potential attackers to a system should be kept from knowing not only passwords and cryptographic keys but also basic information about how the system works, such as the specifications of cryptographic algorithms, communication protocols, and access-control mechanisms. It has long been argued that "security by obscurity" is usually inferior to open design [55, 28]. Of course, the value of writing and publishing specifications is greater when the specifications are clear, complete, and at an appropriate level of abstraction.

Current specifications of security mechanisms and properties vary greatly in quality, scope, purpose, and vocabulary. Some specifications are informal narrations that mix natural language and ad hoc notations. For example, the documents that describe the functioning of security protocols such as SSL [27], SSH [63], and IKE [32] often have this style. Other specifications are precise mathematical statements, sometimes expressed in formal calculi. These specifications have played a particularly significant role in cryptography and cryptographic protocols, but also appear in other areas, for example in information-flow analysis (e.g., [28, 22, 43, 48]).

Many of these specifications serve as the basis for reasoning, with various degrees of rigor and effectiveness, during system design, implementation, and analysis. In recent years, there has been much progress in the development of techniques for stating and proving properties about small but critical security components. For example, a substantial and successful body of work treats the core messages of security protocols and the underlying cryptographic functions. In this area, theory has been relevant to practice, even in cases where the theory is simplistic or incomplete. There seems to have been less progress in treating more complex systems [56], even those parts in the vicinity of familiar security mechanisms. For example, we still have only a limited understanding of many of the interfaces, prologues, and epilogues of practical security protocols.

In this paper, we discuss specifications in the field of security, focusing on protocol specifications. We examine specifications of several sorts:

- In section 2, we consider specifications that concern the step-by-step behavior of a protocol. Such specifications can be largely independent of any assumptions or intended effects of the protocol.
- In section 3, we consider properties of protocols, in particular authenticity and secrecy properties, but also more exotic properties. We emphasize secrecy properties.
- In section 4, we view protocols in context by discussing their boundaries. These boundaries include programming interfaces, protocol negotiation, and error handling.

This paper is an informal, partial overview, and does not advocate any particular methods for specification and verification. Occasionally, however, the spi calculus [6] serves in explanations of formal points. In addition, the paper suggests some gaps and some opportunities for further work. The subject of this paper seems to be reaching maturity, but also expanding. There is still much scope for applying known techniques to important protocols, for developing simpler techniques, for exploring the foundations of those techniques, and also for studying protocols in context, as parts of systems.

2 Protocol narrations

The most common specifications are mere narrations of protocol executions. These narrations focus on the "bits on the wire": they say what data the various participants in a protocol should send in order to communicate. They are sometimes simple, high-level descriptions of sequences of messages, sometimes more detailed documents that permit the construction of interoperable implementations.

Following Needham and Schroeder [52], we may write a typical pair of messages of a protocol thus:

$$\begin{array}{ll} \text{Message 1} & A \rightarrow B : \{N_A\}_{K_{AB}} \\ \text{Message 2} & B \rightarrow A : \{N_A, N_B\}_{K_{AB}} \end{array}$$

Here A and B represent principals (users or computers). In Message 1, A sends to B an encrypted message, with key K_{AB} and cleartext N_A. In Message 2, B responds with a similar message, including N_B in the cleartext. The braces represent the encryption operation, in this case using a symmetric cryptosystem such as DES [48]. The subscripts on K_{AB}, N_A, and N_B are merely hints. It may be understood that A and B both know the key K_{AB} in advance and that A and B freshly generate N_A and N_B respectively, so N_A and N_B serve as nonces.

As Bob Morris has pointed out [7], the notation "Message n $X \rightarrow Y : M$" needs to be interpreted with care, because security protocols are not intended to operate in benign environments. The network between X and Y may be unreliable and even hostile; X and Y themselves may not deserve total trust. So we may interpret "Message n $X \to Y : M$" only as "the protocol designer intended that X send M as the nth message in the protocol, and for it to be received by Y". One may want additional properties of this message, for example that only Y receive it or that Y should know that this message is part of a particular protocol execution; however, such properties cannot be taken for granted.

A sequence of messages is not a complete description of a protocol; it must be complemented with explanations of other forms. Protocol narrations often give some but not all of these explanations.

- As done above, a specification should say which pieces of data are known to principals in advance and which are freshly generated.
- A specification should also say how principals check the messages that they receive. For example, after receipt of Message 2, principal A may be expected to check that it is encrypted under K_{AB} and that the first component of its cleartext is the nonce N_A sent in Message 1. If this check fails, A may ignore the message or report an error. (Section 4 discusses errors further.) Checks are an essential part of protocols. For example, the absence of a check in the CCITT X.509 protocol [18] allowed an attack [16]; other attacks arise when principals assume that the messages that they receive have particular forms [9].
- The emission of Message n+1 follows the reception of Message n only in the simplest protocols. In general, a protocol may allow multiple messages belonging to the same session to be in flight simultaneously. The constraints on the order of messages in SSL have often been misunderstood [60]. Other complex protocols may be similarly confusing.
- As a convention, it is generally assumed that many protocol executions may happen simultaneously, and that the same principal may participate in several such executions, possibly playing different roles in each of them. This convention has exceptions, however. For example, some protocols may restrict concurrency in order to thwart attacks that exploit messages from two simultaneous executions. In addition, some roles are often reserved for fixed principals—for example, the name S may be used for a fixed authentication server. A complete specification should not rely on unclear, implicit conventions about concurrency and roles.

These limitations are widely recognized. They have been addressed in approaches based on process calculi (e.g., [41, 6, 47, 38, 57]) and other formal descriptions of processes (e.g., [53, 58]). The process calculi include established process calculi, such as CSP, and others specifically tailored for security protocols. Here we sketch how protocols are described in the spi calculus [6]; descriptions in other process calculi would have similar features.

The spi calculus is an extension of the pi calculus [50] with primitives for cryptographic operations. Spi-calculus processes can represent principals and sets of principals. For example, the process:

$$(\nu K_{AB})(P_A \mid P_B)$$

may represent a system consisting of two principals, playing the roles of A and B as described above, in a single execution of the protocol. The construct ν is the standard restriction binder of the pi calculus; here it binds a key K_{AB}, which will occur in P_A and P_B. The construct $\mid$ is the standard parallel-composition operation of the pi calculus. Finally, P_A and P_B are two processes. The process P_B may be:

$$c(x).\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ (\nu N_B)\overline{c}\langle\{y, N_B\}_{K_{AB}}\rangle$$

Informally, the components of this process have the following meanings:

- c is the name of a channel, which we use to represent the network on which the principals communicate.
- $c(x)$ awaits a message on c. When a message is received, the bound variable x is instantiated to this message. The expected message in this example is $\{N_A\}_{K_{AB}}$.
- $\mathit{case}\ x\ \mathit{of}\ \{y\}_{K_{AB}}\ \mathit{in}\ (\nu N_B)\overline{c}\langle\{y, N_B\}_{K_{AB}}\rangle$ attempts to decrypt x using the key K_{AB}. If x is a term of the form $\{M\}_{K_{AB}}$, then the bound variable y is instantiated to the contents M, and the remainder of the process $((\nu N_B)\overline{c}\langle\{y, N_B\}_{K_{AB}}\rangle)$ is executed.
- (νN_B) generates N_B.
- $\overline{c}\langle\{y, N_B\}_{K_{AB}}\rangle$ sends $\{M, N_B\}_{K_{AB}}$ on c, where M is the term to which y has been instantiated.

The syntax of the spi calculus distinguishes names (such as c, K_{AB}, and N_B) from variables (x and y), and processes (active entities) from terms (data that can be sent in messages). We refer to previous papers for the details of this syntax. We also omit a definition of P_A; it is similar in style to that of P_B.

Since the spi calculus is essentially a programming language, it is a matter of programming to specify the generation of data, checks on messages, concurrency, and replication. For these purposes, we can usually employ standard constructs from the pi calculus, but we may also add constructs when those seem inadequate (for example, for representing number-theoretic checks). In particular, we can use the ν construct for expressing the generation of keys, nonces, and other data. For example, the name N_B bound with ν in P_B represents the piece of data that B generates. On the other hand, the free names of P_B (namely c and K_{AB}) represent the data that B has before the protocol execution.

Thus, specifications in the spi calculus and other formal notations do not suffer from some of the ambiguities common in informal protocol narrations. Moreover, precise specifications need not be hard to construct: in recent work, Lowe, Millen, and others have studied how to turn sequences of messages into formal specifications [47]. To date, however, formal specifications do not seem to have played a significant role for protocol implementations. Their main use has been for reasoning about the properties of protocols; those properties are the subject of the next section.

3 Protocol properties

Although the execution of a protocol may consist in sending bits on wires, the bits have intended meanings and goals. These meanings and goals are not always explicit or evident in protocol narrations (cf. [7]).

There is no universal interpretation for protocols. Two usual objectives are to guarantee authenticity and secrecy of communications: only the intended principals can send and receive certain pieces of data. Other objectives include forward secrecy [24], non-repudiation, and availability. Some objectives contradict others. For example, some protocols aim to guarantee anonymity rather than authenticity, or plausible deniability [54] rather than non-repudiation. Moreover, many definitions have been proposed even for such basic concepts as authenticity (e.g., [11, 30, 42, 3]).

Nevertheless, there are some common themes in the treatment of protocol properties.

- The participants in security protocols do not operate in a closed world, but in communication with other principals. Some of those principals may be hostile, and even the participants may not be fully trusted. Thus, interaction with an uncertain environment is crucial.
- Security properties are relative to the resources of attackers. Moreover, it is common to attempt to guarantee some properties even if the attackers can accomplish some unlikely feats. For example, although precautions may be taken to avoid the compromise of session keys, an attacker might obtain one of those keys. A good protocol design will minimize the effect of such events. In particular, certificates for keys should expire [23]; and when one key is expiring, it should not be used for encrypting the new key that will replace it.
- It is common to separate essential security properties from other properties such as functional correctness and performance. For example, one may wish to establish that messages between a client and a server are authentic, even if one cannot prove that the server's responses contain the result of applying a certain function to the client's requests.

Protocol properties have been expressed and proved in a variety of frameworks. Some of these frameworks are simple and specialized [16], others powerful and general. A frequent, effective approach consists in formulating properties as predicates on the behaviors (sequences of states or events) of the system consisting of a protocol and its environment (e.g., [62, 11, 31, 41, 51, 53, 14, 57]). For example, in the simple dialogue between A and B shown in section 2, the authenticity of the second message may be expressed thus:

> If A receives a message encrypted under K_{AB}, and the message contains a pair N_A, N_B where N_A is a nonce that A generated, then B has sent the message sometime after the generation of N_A.

Once properly formalized, this statement is either true or false for any particular behavior. Such predicates on behaviors have been studied extensively in the literature on concurrency (e.g., [8, 36]).

A richer view of authenticity also takes into account concepts such as authority and delegation [29, 37]. Those concepts appear, for example, when we weaken the authenticity statement by allowing B to delegate the task of communicating with A and the necessary authority for this task. However, it is still unclear how to integrate those concepts with predicates on behaviors.

Furthermore, some security properties—such as noninterference—are not predicates on behaviors [44, 45]. For instance, suppose that we wish to require that a protocol preserve the secrecy of one of its parameters, x. The protocol should not leak any information about x—in other words, the value of x should not interfere with the behavior of the protocol that the environment can observe. The parameter x may denote the identity of one of the participants or the sensitive data that is sent encrypted after a key exchange. In general, we cannot express this secrecy property as a predicate on behaviors. On the other hand, representing the protocol as a process $P(x)$, we may express the secrecy property by saying that $P(M)$ and $P(N)$ are equivalent (or indistinguishable), for all possible values M and N for x (cf. [59, 33]). Here we say that two processes P_1 and P_2 are equivalent when no third process Q can distinguish running in parallel with P_1 from running in parallel in P_2. This notion of process equivalence (testing equivalence) has been applied to several classes of processes and with several concepts of distinguishability, sometimes allowing complexity-theoretic arguments (e.g., [21, 15, 6, 38]). Now focusing on the spi calculus, we obtain one definition of secrecy:

Definition 1 (One definition of secrecy). *Suppose that the process $P(x)$ has at most x as free variable. Then P* preserves the secrecy *of x if $P(M)$ and $P(N)$ are equivalent for all terms M and N without free variables.*

For example, the process $(\nu K)\overline{c}\langle\{x\}_K\rangle$, which sends x encrypted under a fresh key K on a channel c, preserves the secrecy of x. Previous papers on the spi calculus [6, 1] contain more substantial examples to which this concept of secrecy applies.

Approaches based on predicates on behaviors rely on a rather different definition of secrecy, which can be traced back to the influential work of Dolev and Yao [26] and other early work in this area [35, 49, 46]. According to that definition, a process preserves the secrecy of a piece of data M if the process never sends M in clear on the network, or anything that would permit the computation of M, even in interaction with an attacker.

Next we show one instantiation of this general definition, again resorting to the spi calculus. For this purpose, we introduce the following notation from the operational semantics of the spi calculus; throughout, P and Q are processes, M is a term, m, m_1, …, m_k are names, and x is a variable.

– $P \xrightarrow{\tau} Q$ means that P becomes Q in one silent step (a τ step).
– $P \xrightarrow{m} (x)Q$ means that, in one step, P is ready to receive an input x on m and then to become Q.
– $P \xrightarrow{\overline{m}} (\nu m_1, \ldots, m_k)\langle M\rangle Q$ means that, in one step, P is ready to create the new names m_1, …, m_k, to send M on m, and then to become Q.

We represent the state of knowledge of the environment of a process by a set of terms S with no free variables (intuitively, a set of terms that the environment has). Given a set S, we define $C(S)$ to be the set of all terms computable from S, with the properties that $S \subseteq C(S)$ and $C(C(S)) = C(S)$; thus, C is a closure operator. The main rules for computing $C(S)$ concern encryption and decryption:

- if $M \in C(S)$ and $N \in C(S)$ then $\{M\}_N \in C(S)$;
- if $\{M\}_N \in C(S)$ and $N \in C(S)$ then $M \in C(S)$.

Straightforward rules concern terms of other forms, for example pairs:

- if $M \in C(S)$ and $N \in C(S)$ then $(M, N) \in C(S)$;
- if $(M, N) \in C(S)$ then $M \in C(S)$ and $N \in C(S)$.

Given a set of terms S_0 and a process P_0, we let R be the least relation such that:

- $R(P_0, S_0)$.
- If $R(P, S)$ and $P \xrightarrow{\tau} Q$ then $R(Q, S)$.
- If $R(P, S)$ and $P \xrightarrow{m} (x)Q$ and $m \in C(S)$ and $M \in C(S)$ then $R(Q[M/x], S)$.
- If $R(P, S)$ and $P \xrightarrow{\overline{m}} (\nu m_1, \ldots, m_k)\langle M\rangle Q$ and $m \in C(S)$ and $m_1, \ldots, m_k$ do not occur in S then $R(Q, S \cup \{M\})$.

Intuitively, $R(P, S)$ means that, if the environment starts interacting with process P_0 knowing S_0, then the environment may know S (and all terms computable from it, $C(S)$) when P_0 evolves to P. The environment may know some names initially, but it does not create more names along the way. The first clause in this definition sets the initial state of the interaction. The second one is for silent steps. The third one deals with a message from the environment to the process; the environment must know the message's channel name m and contents M. The fourth one deals with a message in the opposite direction; assuming that the environment knows the message's channel name m, it learns the message's contents M; some new names $m_1, \ldots, m_k$ may occur in M.

We arrive at the following alternative view of secrecy:

Definition 2 (Another definition of secrecy). *Suppose that S is a set of terms with no free variables, and P a process with no free variables. Suppose that the free names of M are not bound in P or any process into which P evolves. Let R be the relation associated with P and S. Then P* may reveal *M from S if there exist P' and S' such that $R(P', S')$ and $M \in C(S')$; and P* preserves the secrecy *of M from S otherwise.*

We do not have much experience with this definition of secrecy for the spi calculus. It is a somewhat speculative translation of definitions proposed in other settings.

By presenting both definitions of secrecy in the same framework, we are in a position to compare them and understand them better. We can immediately see that, unfortunately, neither definition of secrecy implies the other: the first one concerns a process with a free variable x, while the second one concerns a process plus a set of terms with no free variables. There are also deeper differences between them: in particular, the first definition rules out implicit information flows [22], while the second one does not. We leave for further work explaining when one definition is appropriate and when the other, and finding useful relations between them.

Both of these definitions of secrecy rely on a simple, abstract representation of cryptographic functions. More detailed accounts of cryptography may include complexity-theoretic assumptions about those functions (e.g., [43]). Another, challenging subject for further work is bridging the gap between those treatments of cryptography. For instance, we may wonder whether the complexity-theoretic assumptions justify our definitions of secrecy. Analogous questions arise for definitions of authenticity.

4 Protocol boundaries

Often the specification of a protocol and its verification focus on the core of the protocol and neglect its boundaries. However, these boundaries are far from trivial; making them explicit and analyzing them is an important part of understanding the protocol in context. These boundaries include:

(1) interfaces and rules for proper use of the protocol,
(2) interfaces and assumptions for auxiliary functions and participants, such as cryptographic algorithms and network services,
(3) traversals of machine and network boundaries,
(4) preliminary protocol negotiations,
(5) error handling.

We discuss these points in more detail next.

(1) Whereas narrations may say what data the various principals in a protocol should send, they seldom explain how the principals may generate and use that data. On the other hand, the good functioning of the protocol may require that some pieces of data be unrelated (for example, a cleartext and the key used to encrypt it). Other pieces of data (typically session keys, but sometimes also nonces) may need to remain secret for some period of time. Furthermore, as a result of an execution of the protocol, the participants may obtain some data with useful properties. For instance, the protocol may yield a key that can be used for signing application messages. Application program interfaces (or even programming languages) should allow applications to exploit those useful properties, with clear, modular semantics, and without revealing tricky low-level cryptographic details (e.g., [12, 40, 39, 61, 2, 5, 10]).

(2) Some protocols rely on fixed suites of cryptosystems. In other cases, assumptions about the properties of cryptographic operations are needed. For example, in the messages of section 2, it may be important to say whether B

can tell that A encrypted N_A using K_{AB}. This property may hold because of redundancy in N_A or in the encryption function, and would not hold if any message of the appropriate size is the result of encrypting some valid nonce with K_{AB}. It may also be important to say that B is not capable of making $\{N_A, N_B\}_{K_{AB}}$ from $\{N_A\}_{K_{AB}}$ and N_B without K_{AB}. This property is a form of non-malleability [25]. In recent years, the literature on protocols has shown an increasing awareness of subtle cryptographic issues; it may be time for some principled simplification.

Similarly, protocols often rely on network time servers, trusted third parties, and other auxiliary participants. Detailed assumptions about these servers are sometimes absent from protocol narrations, but they are essential in reasoning about protocols.

(3) Protocol messages commonly go across network interfaces, firewalls with tunnels, and administrative frontiers (e.g., [12, 61, 20, 19, 4]). In some contexts (e.g., [17]), even the protocol participants may be mobile. These traversals often require message translations (for example, marshaling and rewriting of URLs). They are subject to filtering and auditing. Furthermore, they may trigger auxiliary protocols. Some of these traversals seem to be a growing concern in protocol design.

(4) Systems often include multiple protocols, each of them with multiple versions and options. Interactions between protocols can lead to flaws; they can be avoided by distinguishing the messages that correspond to each protocol (e.g., [7, 34]). Before executing a protocol (in a particular version, with particular options) the participants sometimes agree to do so by a process of negotiation in which they may consider alternatives. The alternatives can vary in their levels of security and efficiency. In protocols such as SSL, this process of negotiation is rather elaborate and error-prone [60]. Despite clear narrations, it offers unclear guarantees.

(5) As discussed in section 2, protocol specifications often do not explain how principals react when they perceive errors. Yet proper handling of errors can be crucial to system security. For example, in describing attacks on protocols based on RSA's PKCS #1 standard [13], Bleichenbacher reported that the SSL documentation does not clearly specify error conditions and the resulting alert messages, and that SSL implementations vary in their handling of errors. He concluded that even sending out an error message may sometimes be risky and that the timing of the checks within the protocol is crucial.

The intrinsic properties of a protocol, such as the secrecy of session keys, are worthy of study. However, these intrinsic properties should eventually be translated into properties meaningful for the clients of the protocol. These clients may want security, but they may not be aware of internal protocol details (such as session keys) and may not distinguish the protocol from the sophisticated mechanisms that support it and complement it. Therefore, specification and reasoning should concern not only the core of the protocol in isolation but also its boundaries, viewing the protocol as part of a system.

Acknowledgments

Discussions with Mike Burrows, Paul Kocher, John Mitchell, Roger Needham, and Phil Rogaway contributed to the writing of this paper.

References

1. Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–638. Springer-Verlag, 1997.
2. Martín Abadi. Protection in programming-language translations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 868–883, July 1998. Also Digital Equipment Corporation Systems Research Center report No. 154, April 1998.
3. Martín Abadi. Two facets of authentication. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 27–32, 1998.
4. Martín Abadi, Andrew Birrell, Raymie Stata, and Edward Wobber. Secure web tunneling. *Computer Networks and ISDN Systems*, 30(1–7):531–539, April 1998. Proceedings of the 7th International World Wide Web Conference.
5. Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, June 1998.
6. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. Technical Report 414, University of Cambridge Computer Laboratory, January 1997. A revised version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998, and an abridged version will appear in *Information and Computation*.
7. Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.
8. Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
9. Ross Anderson and Roger Needham. Robustness principles for public key protocols. In *Proceedings of Crypto '95*, pages 236–247, 1995.
10. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 419–428, May 1998.
11. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer Verlag, August 1993.
12. Andrew D. Birrell. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems*, 3(1):1–14, February 1985.
13. Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1998.
14. Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the π-calculus. In *CONCUR'98: Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 84–98. Springer Verlag, September 1998.

15. Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.
16. 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, February 1989.
17. L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures, First International Conference (FoSSaCS '98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.
18. CCITT. *Blue Book (Recommendation X.509 and ISO 9594-8: The directory-authentication framework)*. CCITT, 1988.
19. Pau-Chen Cheng, Juan A. Garay, Amir Herzberg, and Hugo Krawczyk. Design and implementation of modular key management protocol and IP secure tunnel on AIX. In *Proceedings of the 5th USENIX UNIX Security Symposium*, pages 41–54, June 1995.
20. William Cheswick and Steven Bellovin. *Firewalls and Internet Security*. Addison-Wesley, 1994.
21. Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
22. Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.
23. Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(7):533–535, August 1981.
24. Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.
25. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In ACM, editor, *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.
26. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.
27. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol: Version 3.0. Available at `http://home.netscape.com/eng/ssl3/ssl-toc.html`, March 1996.
28. Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Company Inc., New York, 1988.
29. Morrie Gasser and Ellen McDermott. An architecture for practical delegation in a distributed system. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, pages 20–30, May 1990.
30. Dieter Gollman. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 46–54, May 1996.
31. James W. Gray III and John McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 108–116, 1995.
32. D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). Available at `ftp://ftp.isi.edu/in-notes/rfc2409.txt`, November 1998.
33. Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In *Proceedings of the 25th ACM Symposium on Principles of Programming Languages*, pages 365–377, 1998.

34. John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In *Security Protocols: 5th International Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer Verlag, 1997.
35. Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.
36. Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.
37. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
38. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
39. John Linn. Generic interface to security services. *Computer Communications*, 17(7):476–482, July 1994.
40. Jonn Linn. RFC 1508: Generic security service application program interface. Web page at `ftp://ds.internic.net/rfc/rfc1508.txt`, September 1993.
41. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.
42. Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 31–43, 1997.
43. Michael Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.
44. John McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.
45. John McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–66, January 1996.
46. Catherine Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 182–195, 1991.
47. Catherine Meadows. Panel on languages for formal specification of security protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, page 96, 1997.
48. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
49. Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.
50. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–40 and 41–77, September 1992.
51. John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
52. Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

53. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
54. Michael Roe. *Cryptography and Evidence.* PhD thesis, University of Cambridge Computer Laboratory, 1997. Available as a technical report of the Centre for Communications Systems Research at `http://www.ccsr.cam.ac.uk/techreports/`.
55. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer system. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.
56. Fred B. Schneider, editor. *Trust in Cyberspace.* National Academy Press, prepublication copy edition, 1998. Report of the Committee on Information Systems Trustworthiness, Computer Science and Telecommunications Board, National Research Council.
57. Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.
58. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings 1998 IEEE Symposium on Security and Privacy*, pages 160–171, May 1998.
59. Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:167–187, 1996.
60. David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second USENIX Workshop on Electronic Commerce Proceedings*, pages 29–40, November 1996. A revised version is available at `http://www.cs.berkeley.edu/~daw/me.html`.
61. Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.
62. Thomas Y. C. Woo and Simon S. Lam. A semantic model for authentication protocols. In *Proceedings of the 1993 IEEE Symposium on Research on Security and Privacy*, pages 178–194, 1993.
63. Tatu Ylönen. SSH—Secure login connections over the Internet. In *Proceedings of the Sixth USENIX Security Symposium*, pages 37–42, July 1996.