Security Protocols and their Properties*

Martín Abadi
Bell Labs Research
Lucent Technologies

Abstract. Specifications for security protocols range from informal narrations of message flows to formal assertions of protocol properties. This paper discusses those specifications, emphasizing authenticity and secrecy properties. It also 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 [64, 34]. 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 [33], SSH [74], and IKE [38] 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., [24, 34, 51, 56]).

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 [65], 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.

*This paper is largely based on two previous papers, "Two facets of authentication" [3] (©1998 IEEE) and "Security protocols and specifications" [4] (©1999 Springer-Verlag), which were written at Compaq's Systems Research Center.

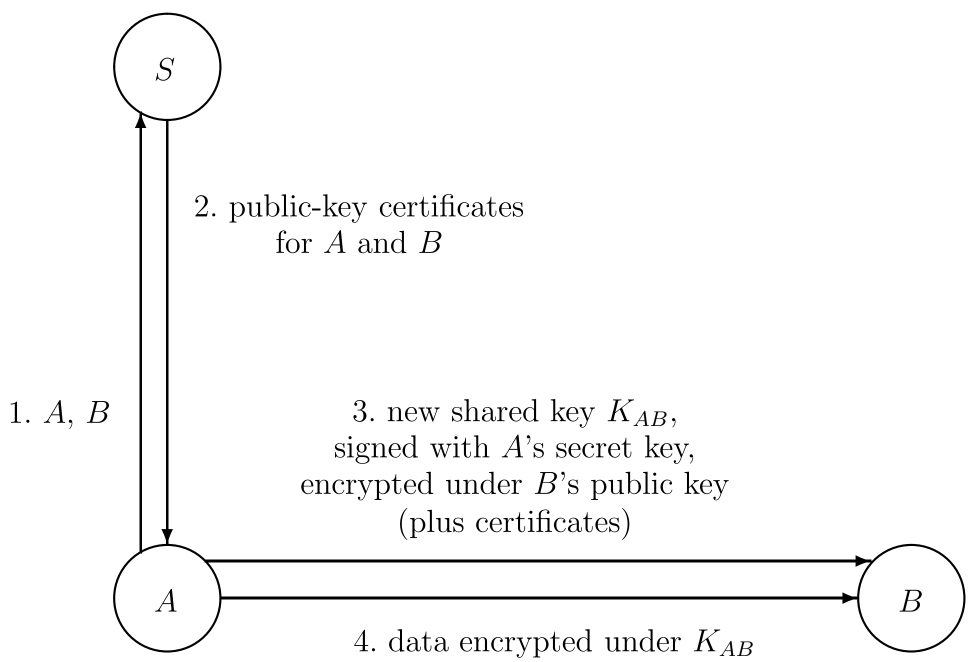
In this paper, we discuss specifications in the field of security, focusing on protocol specifications. We examine specifications of several sorts:

- In section 3, 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 sections 4, 5, and 6, we consider properties of protocols, in particular authenticity and secrecy properties.
- In section 7, 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, a calculus for access control [45, 6] and the spi calculus [8] serve in explanations. 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 An example

This section sketches a first example of a security protocol, as background for the rest of the paper. Although there are many kinds of security protocols, examples like this one are both common and useful.



The protocol relies on a symmetric cryptosystem, such as DES, and on an asymmetric one, such as RSA [26, 62, 56]. With DES, a message encrypted under a key can be decrypted using the same key. Therefore, keys are typically known to several protocol participants, and are called shared keys; they are also called session keys when they protect the messages of a session. In RSA, on the other hand, keys come in pairs. In each pair, one key is used for decryptions and for signatures, and the other is used for encryptions and for signature verifications. The former key may be known to only one participant, while the latter may be widely known. They are therefore called the secret key and the public key, respectively. In addition to symmetric and asymmetric cryptosystems, many protocols rely on one-way hash functions (roughly, functions that are hard to invert and collision-free) [56]. In this protocol, for example, applying a one-way hash function may reduce the size of the data to be signed. Despite these distinctions, we need not be concerned with the details of cryptosystems; we treat them rather abstractly (as black boxes) as in much of the literature.

In the figure, A and B represent principals (users or computers); S is a server. The objective of the protocol is the secure communication of some data between A and B. First, A contacts the server by sending its name and B's. The server replies with certificates that associate the names to public keys, which A can forward to B. In addition, A invents a shared key K_{AB} and sends it to B, signed with A's secret key and encrypted under B's public key; B can decrypt this message and check its signature, obtaining K_{AB}. Finally, A sends encrypted data to B under K_{AB}. Although the figure does not show it, B can send encrypted data to A under K_{AB} as well, in the opposite direction.

For instance, the data that A and B exchange may represent file-system operations or electronic-commerce transactions. Thus, this protocol (and similar ones) can serve a variety of purposes. In contrast, some other protocols are intended for a narrower range of applications, sometimes with application-specific techniques and objectives. They seem to be less important at present. At any rate, this paper does not discuss particular applications in depth.

This first protocol description is somewhat vague and incomplete. The rest of the paper explains protocols in greater depth and detail.

3 Protocol narrations

The most common protocol 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 message flows (like the figure of section 2), sometimes more detailed documents that permit the construction of interoperable implementations.

3.1 Informal protocol narrations

Following Needham and Schroeder [60], we may write a typical pair of messages of a protocol thus:

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

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. 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, as nonces (quantities generated for the purpose of being fresh).

As Bob Morris has pointed out [9], 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. Needham and Schroeder's model is standard:

> We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material. [...]
>
> We also assume that each principal has a secure environment in which to compute, such as is provided by a personal computer or would be by a secure shared operating system.

Furthermore, the principals X and Y themselves may not deserve total trust. So we may interpret "Message n $X \rightarrow Y$: 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 7 discusses errors further.) Checks are an essential part of protocols. For example, the absence of a check in the CCITT X.509 protocol [20] allowed an attack [18]; other attacks arise when principals assume that the messages that they receive have particular forms [11].

- 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 [71]. 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., [49, 55, 8, 46, 66]) and other formal descriptions of processes (e.g., [61, 68]). The process calculi include established process calculi, such as CSP, and others specifically tailored for security protocols. Next we sketch how protocols are described in the spi calculus [8]; descriptions in other process calculi would have similar features.

3.2 Formal protocol narrations

The spi calculus is an extension of the pi calculus [58] 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:

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

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 $\{N_A\}_{K_{AB}}$.
- $case\ x\ of\ \{y\}_{K_{AB}}\ in\ (\nu N_B)\overline{c}\langle\{y, N_B\}_{K_{AB}}\rangle$ 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 details of this syntax and its semantics. 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 [55]. 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.

4 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 [9].

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 [28], non-repudiation, and availability. Some objectives contradict others. For example, some protocols aim to guarantee anonymity rather than authenticity, or plausible deniability [63] rather than non-repudiation. Moreover, many definitions have been proposed even for such basic concepts as authenticity (e.g., [13, 36, 50]).

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, following Needham and Schroeder, it is common to assume that an attacker can interpose a computer in all communication paths. In addition, 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 [25]; 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 (e.g., [18]), 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., [73, 13, 37, 49, 59, 61, 16, 66]). For example, in the simple dialogue between A and B shown in section 3, 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., [10, 44]).

A richer view of authenticity also takes into account concepts such as authority and delegation [35, 45]. 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. The next section discusses some of those concepts informally. (Papers by Gollmann and Lowe also elaborate on the definition of authenticity [36, 50], in other directions.)

The following section, section 6, presents two definitions of secrecy. Only one of them is based on predicates on behaviors.

5 Two facets of authentication

Authentication can serve both for assigning responsibility and for giving credit. An "authenticated" message M from a principal A to a principal B may be used in at least two distinct ways:

- B may believe that the message M is being supported by A's authority. For example, suppose that B is a file server, A a client, and M a request to delete a file f. Then B may use A's identity as an argument to the access control decision of whether to delete f.

- B may attribute credit for the message M to A. For example, suppose that B is running a contest, offering a prize to the principal that mails the factors for a large number. When B receives the message M as an entry, B may give credit for the entry to A.

These two uses are sharply different. Furthermore, some natural protocols are adequate for assigning responsibility but not for giving credit, and vice versa. When a new authentication protocol is designed, it is therefore prudent to state whether the protocol is intended to establish responsibility, credit, or both. Similarly, when an authentication protocol is analyzed, it is prudent to clarify whether its properties suffice for establishing responsibility or credit.

This section explains the distinction between responsibility and credit, through several examples, and discusses the role of this distinction in the design and analysis of protocols. The two facets of authentication are most clearly separate in protocols that rely on asymmetric cryptosystems. We therefore take such protocols as examples. Although our list of examples is not meant to be comprehensive, we illustrate the distinction between responsibility and credit through several protocols of different styles. Our emphasis is not on classifying cryptographic techniques. (See [56, Chapter 12] for a helpful classification.) Rather, we focus on the higher-level guarantees that protocols provide to the applications that may rely on them.

The direct author of a message need not always be held responsible for the message or be given credit for it. In particular, a principal may delegate some part of its authority to another principal, taking responsibility for messages sent by the delegate (much like one can let a friend withdraw from one's bank account). Similarly, a principal may legitimately divert credit for its messages to another principal (much like one can deposit into a friend's bank account). Therefore, even when it is proved beyond a reasonable doubt that a principal sent a message, responsibility and credit may not follow. Responsibility and credit are thus distinct from non-repudiation of origin and other related concepts [75, 63].

5.1 Four examples

Next we consider four examples. They all concern techniques that appear in published, useful protocols. In all four examples, responsibility and credit for the messages between two parties is assigned by the parties themselves. For simplicity, we do not consider scenarios where a third party (for example, a judge) may assign responsibility or credit, but such scenarios could be interesting too.

Signing a public key As a first example, we consider a simple protocol where a principal A creates a short-term key pair, sends the short-term public key to a principal B, signing it with its long-term secret key, then uses the short-term secret key for signing further messages:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A,\, B,\, \{K,\, A,\, B,\, T\}_{K_A^{-1}} \\ \text{Message 2} & A \rightarrow B: & A,\, B,\, \{\{M\}_{K^{-1}}\}_{K_B} \end{array}$$

Here M is an arbitrary message, T is a timestamp, K_A is A's long-term public key, K_A^{-1} is the corresponding secret key, K is A's short-term public key, and K^{-1} is the corresponding secret key. We use braces for signatures, as in $\{M\}_{K^{-1}}$, and for encryptions, as in $\{\{M\}_{K^{-1}}\}_{K_B}$. Message 1 is the core of the protocol; Message 2 appears only as an example of the use of K.

In one interpretation of this protocol, Message 1 conveys that A takes responsibility for the key K, so B can blame A for any message signed with K^{-1}. Thus, the protocol fits applications that require responsibility for a message (or for a connection). For example, the protocol seems adequate for the situation where B is a file server, A a client, and M a request to delete a particular file. It seems adequate even if A gives K^{-1} to a delegate, allowing the delegate to make requests on A's behalf (although delegation mechanisms where B knows the identity of the delegate may be preferable [45]).

In a second interpretation of this protocol, B would assign credit to A for every message that is signed with K^{-1}. This interpretation is not justified, as the following would constitute an attack:

$$\begin{array}{llll} \text{Message 1} & A \rightarrow B: & A,\, B,\, \{K,\, A,\, B,\, T\}_{K_A^{-1}} & \text{(intercepted by } C\text{)} \\ \text{Message 1'} & C \rightarrow B: & C,\, B,\, \{K,\, C,\, B,\, T\}_{K_C^{-1}} & \\ \text{Message 2} & A \rightarrow B: & A,\, B,\, \{\{M\}_{K^{-1}}\}_{K_B} & \text{(intercepted by } C\text{)} \\ \text{Message 2'} & C \rightarrow B: & C,\, B,\, \{\{M\}_{K^{-1}}\}_{K_B} & \end{array}$$

Here C is an attacker who signs the public key generated by A. On receipt of M, we would have B give credit for M to C rather than to A. This sequence of messages constitutes an attack with the second interpretation, since it may result in erroneous credit to C. On the other hand, with the first interpretation, this sequence of messages may result in erroneous blame to C; it may also result in denial of service to A and B, but has no other direct negative impact on them.

The protocol can be strengthened in a variety of ways. In particular, A may sign its own name with the secret key K^{-1}, for example as in:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A,\ B,\ \{K,\ A,\ B,\ T\}_{K_A^{-1}},\ \{A\}_{K^{-1}} \\ \text{Message 2} & A \rightarrow B: & A,\ B,\ \{\{M\}_{K^{-1}}\}_{K_B} \end{array}$$

or

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A,\ B,\ \{K,\ A,\ B,\ T\}_{K_A^{-1}} \\ \text{Message 2} & A \rightarrow B: & A,\ B,\ \{\{A,\ M\}_{K^{-1}}\}_{K_B} \end{array}$$

These modification do not prevent an attacker C from signing the key K with its key K_C^{-1}; however, C cannot sign its own name with K^{-1}, since C does not have K^{-1}. Therefore, the protocol becomes adequate for giving credit for M to A.

We may say that, with these modifications, A proves possession of the secret key K^{-1} [56, Definition 12.7]. However, this statement should not be taken literally. For example, suppose that A is a user with a smart-card and that B is a server. The smart-card may sign a short-term key K generated by the workstation to which the smart-card is connected, and the workstation may sign the name A, while the user and the smart-card may never have access to the secret key K^{-1}.

Protocols similar to this one may be used for registering public keys. In this application, B is a certification authority and A is a client that enters a new public key K into B's registry. It has been argued that, whenever a client is associated with a public key K, the client should prove possession of the corresponding secret key K^{-1} [56, Remark 13.23]. However, this view is not universal—it is not shared, in particular, in some recent public-key-infrastructure designs [31].

Encrypting a session key While the first example shows that a signature may lead to responsibility, the second example shows that an encryption may lead to credit, and that a decryption may lead to responsibility.

We consider the situation where a principal A transmits a session key K (say, a DES key) to another principal B, encrypting K under B's public key K_B, and including the name A along with K:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{A,\ K\}_{K_B} \\ \text{Message 2} & A \rightarrow B: & \{M\}_K \\ \text{Message 3} & B \rightarrow A: & \{M'\}_K \end{array}$$

Messages 2 and 3 simply illustrate the use of the session key K for sending some messages, M and M'. We assume that both principals have means for recognizing and ignoring their own messages, so for example A will not mistake a replay of $\{M\}_K$ for a message from B.

This protocol is adequate for applications that require responsibility for B, and perhaps for applications that require credit for A:

- Only B can extract K from Message 1. Therefore, B can take responsibility for the use of K. Whenever A receives a message encrypted under K, if A did not generate the message itself, then A can be certain that B generated the message, or that some principal to which B gave K generated the message. So A can hold B responsible for the message.

- Since Message 1 yields no proof of A's identity, B does not know who else has K. Therefore, B should not send any secrets under K. We do not consider the issue of credit for B for lack of a compelling example where B would claim credit for public data. (Perhaps only secrets are worth claiming credit for.)

- Since any principal could have produced Message 1, B cannot hold anyone responsible for messages from A encrypted under K.

- Nevertheless, by including its name in Message 1, A claims credit for messages encrypted under K. Of course, if A chooses K incompetently or maliciously, then another principal C might have K as well, and might also produce messages encrypted under K. Still, since A has K, and assuming that C sends those messages to A or that A is capable of intercepting them, A can read and produce those same messages. Therefore, (some) credit to A is justified.

The last of these points indicates that this protocol may not be a solid basis for unqualified credit to A. The following alternative protocol provides a clearer case for credit to A and preserves the responsibility of B:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{J_A\}_{K_B} \\ \text{Message 2} & B \rightarrow A: & J_B \\ \text{Message 3} & A \rightarrow B: & \{M\}_K \\ \text{Message 4} & B \rightarrow A: & \{M'\}_K \\ & (K = H(J_A, J_B, A, B)) & \end{array}$$

where J_A and J_B are random quantities invented by A and B, respectively, and K is computed by applying a one-way hash function H to the concatenation of J_A, J_B, and the names A and B. With this alternative protocol, A can still forward a message M received from another principal, of course, but A cannot pick a session key K in use by other principals.

Making a session key from encrypted shares As a third example, we consider a protocol for obtaining a shared key from encrypted shares. The protocol has been applied in several contexts. It appears in the work of Lampson et al. [45], and it also appears as a component of Krawczyk's SKEME protocol [42]. We adopt Krawczyk's name for this protocol: SHARE.

SHARE enables two principals A and B to obtain a shared key, assuming that initially each knows the public key of the other (K_A or K_B). Much as in the previous example, each of the principals invents a random quantity (J_A or J_B), as a "half key". Then the following exchange takes place:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & \{J_A\}_{K_B} \\ \text{Message 2} & B \rightarrow A: & \{J_B\}_{K_A} \end{array}$$

After this exchange, the two principals A and B compute a shared key K by applying a one-way hash function to the concatenation of the half keys J_A and J_B.

In his explanation of SHARE, Krawczyk says:

> If A follows the protocol then she is assured that the shared key [...] is not know to anyone except B (though A does not have the assurance that B knows the key). And analogously for B.

These properties should not be interpreted literally (and it is not Krawczyk intention that they be interpreted literally [43]). In fact, in their literal interpretation, these properties do not hold. Consider, for example, the following message sequence:

$$\begin{array}{lll} \text{Message 1} & A \to B: & \{J_A\}_{K_B} \\ \text{Message 1'} & B \to C: & \{J_A\}_{K_C} \\ \text{Message 2} & C \to A: & \{J_C\}_{K_A} \end{array}$$

In this sequence, A sends a half key to B, who then sends the same half key to C. By whatever means, B makes C believe that Message 1' comes from A. Therefore, C replies to A with another half key. As a result of this exchange, both A and C compute a shared key; C believes that it is shared with A, while A believes that it is shared with B.

Whether this scenario constitutes an attack depends on the use of the protocol. It constitutes an attack in applications where it can result in harm to the principals A and C, which follow the protocol.

- In some situations, this scenario may cause some confusion, but no clear harm. As a result of B's behavior, A may attribute C's requests to B, and lend them the weight of B's authority. Moreover, A may inadvertently reveal some of B's secrets to C. Thus, B's behavior may harm B, but would not directly harm A or C.

  Note that B does not obtain the key that A and C compute. Therefore, B cannot learn C's secrets by decrypting the subsequent messages from C to A. (In this respect, this scenario is quite different from Lowe's attack on the Needham-Schroeder public-key protocol [49], although its structure is reminiscent of that attack.)

- More concretely, A may be a file server, and B and C two of its clients. In this case, the confusion may result in some immediate damage to B's files as a result of C's requests. Moreover, C may be able to read some of B's files.

  Indirectly, however, there can also be some harm to C if C's requests are not sufficiently explicit. Because of the confusion, A may write some of C's data into B's files. In a later session B can read these files, obtaining C's data.

- Finally, if A is running a contest, then it may think that a winning entry that arrives under K came from B when in fact it came from C.

In short, SHARE is adequate for some applications that require responsibility but not necessarily for applications that give credit.

With some additional precautions, SHARE can be used in applications that give credit. For example, each message that an application encrypts under the shared key could include (bound in the encryption) the name of its source, who expects credit for this message if any credit is to be had at all. The inclusion of this name is a prudent measure; it agrees with Principle 3 of [9]:

> If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

Thus, the issue of credit can be separated from the protocol for agreement on a key, and shifted to the use of the key.

Alternatively, we can easily strengthen SHARE, preventing the scenario described above. One possible modification is to let K be the result of applying a one-way hash function to the concatenation of J_A, J_B, and the names A and B, as in the previous example. Another one is to add messages where the two parties A and B prove knowledge of K, as in SKEME.

Krawczyk does not view SHARE as a complete protocol, but only as a phase of SKEME:

> To be really meaningful this phase [SHARE] needs to be combined with the other phases of the protocol [SKEME].

One may instead argue that SHARE is meaningful on its own, although precautions are needed for its use, and elucidating its meaning requires separating credit from responsibility.

The Station-to-Station protocol As a last example, we consider the Station-to-Station protocol [28]. This protocol is fairly well known, so we do not reproduce it or discuss it in detail.

Basically, the Station-to-Station protocol consists of a signed Diffie-Hellman exchange [27], where both parties sign the Diffie-Hellman shares, and where they confirm possession of the resulting session key by using it to encrypt the signed Diffie-Hellman shares. Without the key confirmation, the protocol suffices for establishing responsibility: each party may hold the other responsible for messages under the session key. With the key confirmation, the protocol is also suitable for establishing credit.

Although the Station-to-Station protocol may be rather attractive, it is not without alternatives. Other techniques for key confirmation could replace the encryption of the signed Diffie-Hellman shares. Furthermore, the key confirmation could be postponed or avoided altogether if, whenever credit is wanted, each message encrypted under the session key would contain the name of its source.

5.2 Views on responsibility and credit

We close this section with a discussion of views on responsibility and credit in the design and analysis of protocols.

Design In the literature on protocol design, there seems to be a consensus that an authentication protocol should at least establish responsibility, probably because responsibility is crucial for access control. In the context of access control, the fundamental property of a secure channel from a principal is that the channel "speaks for" the principal [45, 6]. In essence, a channel C speaks for a principal A if A's authority backs every message on C. This semantics justifies the following axioms:

- If A says that C speaks for A, then C does speak for A.
- If C speaks for A and C says s, then A says s.

These axioms are useful and sensible when "A says s" entails that A takes responsibility for s; but they would be unreasonable if "A says s" was meant to imply that A deserves credit for s. In access control, responsibility is largely separate from credit.

There does not seem to be a consensus that an authentication protocol should also establish credit. Although the literature does not seem to contain an explicit, articulate debate about this issue, we can propose likely arguments for both sides.

- We may argue that, once a protocol has set up a channel that speaks for a principal, it is easy to use the channel for establishing credit whenever the need arises. (For example, the principal may send its name on the channel; see section 5.1.) So it is not essential that an authentication protocol establish credit.

- On the other hand, we may say that stronger guarantees are better than weaker guarantees, particularly when protocols may be applied carelessly or in ways not fully anticipated by their designers. Establishing credit is a matter of prudence.

The latter argument may be prevailing. Popular Internet protocols (such as the SSL protocol) probably establish credit, although one may conjecture that applications rarely take advantage of this feature.

Analysis Formal analyses seldom highlight the distinction between responsibility and credit, and the effect of a proof of possession of a secret key. There are some exceptions, for example in the work of van Oorschot and Syverson [69, 67]. Elsewhere, the distinction between responsibility and credit seems to be made through decisions in the modelling of protocol participants.

- Honest protocol participants are expected to follow the rules of the protocol faithfully, and not to try to obtain credit for messages that they did not generate themselves. A proof about honest protocol participants may show that a protocol establishes responsibility, but not credit.

- When an attacker is included as protocol participant, the attacker is not forced to follow the rules of the protocol, and may attempt to get undue credit. A proof that concerns such an attacker can show that a protocol establishes credit.

The protocols described in this section should make good examples for future work on protocol analysis. While these protocols do not present any challenges of scale, they exhibit common subtleties.

6 Two notions of secrecy

Continuing the explanation of protocol properties, this section presents and compares two definitions of secrecy. One relies on equivalences between processes. The other one is based on an inductive characterization of the knowledge of the environment in the behaviors of protocols.

6.1 Secrecy and equivalences

Some security properties—such as noninterference—are not predicates on behaviors [52, 53]. 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 [8, 46] (see also [32, 70, 39]). 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., [23, 17, 8, 46]).

Now focusing on the spi calculus, we obtain one definition of secrecy:

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

Previous papers on the spi calculus [8, 1] contain some substantial examples to which this concept of secrecy applies, with proofs. Here we discuss examples only briefly.

- $\overline{c}\langle\{x\}_K\rangle$, which sends x encrypted under a key K on a channel c, does not preserve the secrecy of x. An environment that has c and K can distinguish $\overline{c}\langle\{M\}_K\rangle$ and $\overline{c}\langle\{N\}_K\rangle$ when M and N are different terms.

- In contrast, $(\nu K)\overline{c}\langle\{x\}_K\rangle$, which sends x encrypted under a fresh key K on a channel c, preserves the secrecy of x. However, $(\nu K)(\overline{c}\langle\{x\}_K\rangle \mid \overline{c}\langle\{n\}_K\rangle)$, which sends both x and n encrypted under a fresh key K on a channel c, does not preserve the secrecy of x, because the environment can detect when x is n by comparing two ciphertexts.

- *case* x *of* $\{y\}_K$ *in* $\overline{c}\langle m\rangle$, which sends a message on a channel c only if x is of a certain form ($\{y\}_K$), does not preserve the secrecy of x, because it leaks some information about x. This leak is an implicit information flow [24].

- In a typical example, $P(x)$ describes a protocol for establishing a session key K and for sending x encrypted under K. Proving that $P(x)$ preserves the secrecy of x may require lemmas about the auxiliary key K. The final result, though, concerns x rather than K.

6.2 Secrecy and inductive definitions

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 [30] and other early work in this area [41, 57, 54]. 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, with a symmetric cryptosystem. 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 channel 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 $m_1, \ldots, m_k$, to send M on channel 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. For a symmetric cryptosystem, where the same key is used for encryption and decryption, we have:

- 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 $M \in C(S)$.

More generally, for an asymmetric cryptosystem, the second rule should instead say that if $\{M\}_N \in C(S)$ and $N^{-1} \in C(S)$ then $M \in C(S)$, where N is an encryption key and N^{-1} is the corresponding decryption key. 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 free in P_0 and 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. (The names are new with respect to P_0 and S, and a fortiori with respect to S_0 since $S_0 \subseteq S$.)

We arrive at the following alternative view of secrecy:

Definition 2 (Another definition of secrecy) *Suppose that P is a process, S a set of terms, and M a term, all without free variables, and that the free names of M are among those of P and S. 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.*

This definition is a translation of definitions proposed in other settings. We do not have much experience with this definition in the spi calculus. However, it is easy to apply to examples:

- $\overline{c}\langle\{m\}_K\rangle$ may reveal m from $\{c, K\}$.
- In contrast, $\overline{c}\langle\{m\}_K\rangle$ and $(\nu K)\overline{c}\langle\{m\}_K\rangle$ both preserve the secrecy of m from $\{c\}$. So does $(\nu K)(\overline{c}\langle\{m\}_K\rangle \mid \overline{c}\langle\{n\}_K\rangle)$.
- $(\nu K)(\overline{c}\langle K\rangle \mid c(x).case\ x\ of\ \{y\}_K\ in\ \overline{c}\langle\{m\}_K\rangle)$ may reveal m from $\{c\}$: after the environment learns K from the first message, it can send a ciphertext under K, triggering the emission of $\{m\}_K$, which it can decrypt by using K again.
- Although we could say, informally, that $(\nu K)\overline{c}\langle K\rangle$ may reveal K from $\{c\}$, the definition does not justify it, because of the requirement on the free names of M. This requirement is sensible: omitting it, we could say (without saying much!) that $(\nu K)\overline{c}\langle K\rangle$ may reveal m from $\{c\}$ for every name m different from c, since bound names are subject to renaming.
- Suppose that $P(x)$ describes a protocol for establishing a session key K and for sending x encrypted under K, using a public channel c. If the protocol is sound and n is a fresh name, we should obtain that $P(n)$ preserves the secrecy of n from $\{c\}$.

  On the other hand, we cannot obtain that $P(x)$ preserves the secrecy of x from $\{c\}$ for every x, because the environment may know some possible values of x. In particular, the value of x might be c; it might also be the name of one of the protocol participants that is transmitted in clear on c.

6.3 Comparisons

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 and some terms with no free variables. There are also deeper differences between them. In particular, the first definition rules out implicit information flows, while the second one does not. The first definition can express that a piece of data x is protected even if some of the possible values of x are known to the environment; the second definition cannot deal with such general statements, and seems to work best when the piece of data is a fresh name, such as a fresh key.

We leave for further work explaining when one definition is appropriate and when the other, and finding useful relations between them. In practice, however, they may be roughly equivalent in many examples: the secrecy of a piece of data x and the secrecy of a fresh key that protects x are often closely related.

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., [51]). 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.

7 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., [14, 48, 47, 72, 2, 7, 12]).

(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 3, 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 [29]. 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., [14, 72, 22, 21, 5]). In some contexts (e.g., [19]), 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., [9, 40]). 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 [71]. Despite clear narrations, it offers unclear guarantees.

(5) As discussed in section 3, 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 [15], 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.

Acknowledgements

Discussions with Mike Burrows, Hugo Krawczyk, and Butler Lampson led to most of the material of section 5. I believe that the arguments "for credit" and "against credit" of section 5.2 approximately reflect the views of Krawczyk and Lampson, respectively. (So they deserve some credit for these arguments but are not responsible for them.) Comments by referees of [3] suggested the mentions of non-repudiation and of third-party judgement. Comments by Andy Gordon and Marcelo Fiore led to improvements in the second definition of secrecy, which differs slightly from that of [4]. Also helpful were discussions with Ross Anderson, Dieter Gollmann, Paul Kocher, Gavin Lowe, John Mitchell, Roger Needham, Ron Rivest, Mike Roe, Phil Rogaway, and Adi Shamir.

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*, volume 1443 of *Lecture Notes in Computer Science*, pages 868–883. Springer-Verlag, 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. Security protocols and specifications. In *Foundations of Software Science and Computation Structures: Second International Conference, FOSSACS '99*, volume 1578 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, March 1999.

[5] 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.

[6] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, October 1993.

[7] 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.

[8] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.

[9] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[10] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[11] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In *Proceedings of Crypto '95*, pages 236–247, 1995.

[12] 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.

[13] 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.

[14] Andrew D. Birrell. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems*, 3(1):1–14, February 1985.

[15] 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.

[16] 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.

[17] Michele Boreale and Rocco De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.

[18] 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.

[19] 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.

[20] CCITT. *Blue Book (Recommendation X.509 and ISO 9594-8: The directory-authentication framework)*. CCITT, 1988.

[21] 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.

[22] William Cheswick and Steven Bellovin. *Firewalls and Internet Security*. Addison-Wesley, 1994.

[23] Rocco De Nicola and Matthew C. B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[24] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.

[25] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(7):533–535, August 1981.

[26] Data encryption standard. Fed. Inform. Processing Standards Pub. 46, National Bureau of Standards, Washington DC, January 1977.

[27] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[28] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[29] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 542–552. ACM, 1991.

[30] 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.

[31] Carl M. Ellison. SPKI certificate documentation. Web pages at `http://www.clark.net/pub/cme/html/spki.html`, 1997.

[32] Riccardo Focardi and Roberto Gorrieri. A classification of security properties. *Journal of Computer Security*, 3:5–33, 1994/1995.

[33] 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.

[34] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Company Inc., New York, 1988.

[35] 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.

[36] Dieter Gollmann. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 46–54, May 1996.

[37] 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.

[38] D. Harkins and D. Carrel. RFC 2409: The Internet Key Exchange (IKE). Available at `ftp://ftp.isi.edu/in-notes/rfc2409.txt`, November 1998.

[39] 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.

[40] 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.

[41] Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.

[42] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security*, February 1996. Available at `http://bilbo.isu.edu/sndss/sndss96.html`.

[43] Hugo Krawczyk. Private communication. 1997.

[44] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.

[45] 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.

[46] 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.

[47] John Linn. Generic interface to security services. *Computer Communications*, 17(7):476–482, July 1994.

[48] Jonn Linn. RFC 1508: Generic security service application program interface. Web page at `ftp://ds.internic.net/rfc/rfc1508.txt`, September 1993.

[49] 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.

[50] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 31–43, 1997.

[51] Michael Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.

[52] John McLean. Security models. In John Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.

[53] John McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–66, January 1996.

[54] 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.

[55] Catherine Meadows. Panel on languages for formal specification of security protocols. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, page 96, 1997.

[56] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[62] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[63] 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/`.

[64] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer system. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[65] 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.

[66] Steve Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.

[67] Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28, 1994.

[68] 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.

[69] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the first ACM Conference on Computer and Communications Security*, pages 232–243, 1993.

[70] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:167–187, 1996.

[71] 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`.

[72] 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.

[73] 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.

[74] Tatu Ylönen. SSH—Secure login connections over the Internet. In *Proceedings of the Sixth USENIX Security Symposium*, pages 37–42, July 1996.

[75] Jianying Zhou and Dieter Gollmann. A fair non-repudiation protocol. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 55–61, May 1996.