Prudent Engineering Practice for Cryptographic Protocols

Martín Abadi* Roger Needham†

Abstract

We present principles for the design of cryptographic protocols. The principles are neither necessary nor sufficient for correctness. They are however helpful, in that adherence to them would have avoided a considerable number of published errors.

Our principles are informal guidelines. They complement formal methods, but do not assume them. In order to demonstrate the actual applicability of these guidelines, we discuss some instructive examples from the literature.

1 Introduction

It has been evident for a number of years that cryptographic protocols, as used in distributed systems for authentication and related purposes, are prone to design errors of every kind. A considerable body of literature has come into being in which various formalisms are proposed for investigating and analyzing protocols to see whether they contain various kinds of blunders. (Liebl's bibliography [11] contains references to protocols and formalisms.) Although sometimes useful, these formalisms do not of themselves suggest design rules; they are not directly beneficial in seeing how to avoid trouble.

We present principles for the design of cryptographic protocols. The principles are not necessary for correctness, nor are they sufficient. They are however helpful, in that adherence to them would have contributed to the simplicity of protocols and avoided a considerable number of published confusions and mistakes.

We arrived at our principles by noticing some common features among protocols that are difficult to analyze. If these features are avoided, it becomes less necessary to resort to formal tools—and also easier to do so if there is good reason to. The principles themselves are informal guidelines, and useful independently of any logic.

We illustrate the principles with examples. We draw our examples from the published literature, in order to demonstrate the actual applicability of our guidelines. Some of the oddities and errors that we analyze here have been documented previously (in particular, in [4]). Other examples are new: protocols by Denning and Sacco [6], Lu and Sundareshan [12], Varadharajan, Allen, and Black [29], and Woo and Lam [32]. We believe they are all instructive.

Generally, we choose examples from the authentication literature, but the principles are applicable elsewhere, for example to electronic-cash protocols (e.g., [15]). We focus on traditional cryptography, and on protocols of the sort commonly implemented with the DES [18] and the RSA [26] algorithms. In particular, we do not consider the subtleties of interactive schemes for signatures (e.g, [7]). Moreover, we do not discuss the choice of cryptographic mechanisms with adequate protection properties, the correct implementation of cryptographic primitives, or their

*ma@src.dec.com. Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, California 94301, USA.

†rmn@cl.cam.ac.uk. University of Cambridge, Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB1 3QG, UK.

appropriate use; these subjects are discussed elsewhere (e.g., [30, 17]).

Throughout, we concentrate on the simple facts with the largest potential applicability and payoff. Admittedly, the literature is full of ingenious protocols and attacks. We do not attempt to organize the principles that underly this ingenuity, and perhaps it is not necessary. We hope that our simple principles and examples will be of help to the engineering of robust cryptographic protocols.

2 Basics

A protocol, for present purposes, is a set of rules or conventions defining an exchange of messages among a set of two or more partners. These partners are users, processes, or machines, which we will generically refer to as principals. In a cryptographic protocol the whole or part of some or all of the messages is encrypted. We interpret the term encryption fairly broadly, applying it for example to signature operations. Encryption and decryption are for present purposes defined as key-dependent transformations of a message which may only be inverted by using a definite key; the keys used for encryption and decryption are the same or different, depending on the cryptographic algorithm used.

We find two overarching principles for the design of secure cryptographic protocols. One principle is concerned with the content of a message and the other with the circumstances in which it will be acted upon:

1. Every message should say what it means—its interpretation should depend only on its content.

2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.

Next we explain these general principles. They lead to other, more specific recommendations, which we discuss in the subsequent sections.

2.1 Explicit communication

In full, our first basic principle is:

Principle 1

> Every message should say what it means: the interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content—though if there is a suitable formalism available that is good too.

For example, an authentication server S might send a message whose meaning may be expressed thus: "After receiving bit-pattern P, S sends to A a session key K intended to be good for conversation with B". All elements of this meaning should be explicitly represented in the message, so that a recipient can recover the meaning without any context. In particular, if any of P, S, A, B, or K are left to be inferred from context, it may be possible for one message to be used deceitfully in place of another.

Principle 1 is not completely original. In [4], we recommend the use of a logical notation in generating and describing protocols—essentially proposing a method to follow the principle. Establishing the correspondence between the logical protocol and its concrete implementation can be a simple matter of parsing, as for example in [31, Section 4.3.2]. Although a precise comparison of informal ideas is difficult, we also find an affinity with Boyd and Mao's proposal that protocols should be robust in the sense that "authentication of any message in the protocol depends only on information contained in the message itself or already in the possession of the recipient" [3]. An operational variant on this theme appears in the work of Woo and Lam, who call a protocol full information if "its initiator and responder always include in their outgoing encrypted messages all the information they have gathered" [33].

2.2 Appropriate action

For a message to be acted upon, it not merely has to be understood but a whole variety of other conditions have to hold too. These often consist of what may informally be regarded as statements of trust, though this anthropomorphic notion should be used with care. Statements of trust cannot be wrong though they may be considered inappropriate. For example, if someone believes that choosing session keys should be done by a suitably trusted server rather than by one of the participants in a session, then he will not wish to use a protocol such as the Wide-mouthed-frog protocol [4].

In general, we have:

Principle 2

> The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.

2.3 Secrecy

The secrecy of certain pieces of information is essential to the functioning of cryptographic protocols. Obviously, a protocol should not publicize the cryptographic keys used for communicating sensitive data.

None of the our principles makes this point explicitly. Rather, all of our principles warn against mistakes that often imply the loss of secrecy, integrity, and authenticity. Some of the examples clarify how the principles relate to the need for secrecy.

There may be more to say about secrecy guidelines for cryptographic protocols, but these are outside the scope of the present paper.

2.4 Examples and other principles

Below we discuss many concrete examples where errors would have been avoided by use of our two basic principles. We also introduce other principles. Some of these are clearly corollaries of the basic ones, others are not. In particular, we recommend:

- Be clear on how encryption is used, and on the meaning of encryption.
- Be clear on how the timeliness of messages is proved, and on the meaning of temporal information in messages.

Hopefully, the two basic principles will encourage a certain lucidity in the design of cryptographic protocols, and thereby make it easier to follow our other principles.

3 Notation

We adopt notation common in the literature. That notation is not quite uniform and, in examples, we make compromises between uniformity of this paper and faithfulness to original notation.

In this paper, the symbols A and B often represent arbitrary principals, S represents a server, T a timestamp, N a nonce (a quantity generated for the purpose of being recent), K a key, and K^{-1} its inverse. In symmetric cryptosystems such as DES, K and K^{-1} are always equal. For asymmetric cryptosystems such as RSA, we assume for simplicity that the inversion operation is an involution (so ${K^{-1}}^{-1}$ equals K); we tend to use K^{-1} for the secret part and K for the public part of a key pair (K, K^{-1}). We write $\{X\}_K$ to represent X encrypted under K; anyone who knows $\{X\}_K$ and the inverse of K can obtain X. If K is secret, we may refer to $\{X\}_K$ as a signed message, and to the encryption operation as a signature.

For example,

$$\text{Message 4} \quad B \rightarrow A: \quad \{T_a + 1\}_{K_{ab}}$$

describes the fourth message in a protocol; in this message, B sends to A the timestamp T_a incremented by 1, under the key K_{ab}. In this example, the subscript a in T_a is a hint that A first generated T_a; the subscript ab in K_{ab} is a hint that K_{ab} is intended for communication between A and B. Similarly, we may write K_a for A's public key.

4 Naming

The most immediate instance of Principle 1 prescribes being explicit about names of principals:

Principle 3

> 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.

The names relevant for a message can sometimes be deduced from other data and from what encryption keys have been applied. However, when this information cannot be deduced, its omission is a blunder with serious consequences.

The principle is obvious and simple, yet it is commonly ignored. We give several examples of fairly different natures.

Example 3.1 In [6, p. 535], Denning and Sacco propose a protocol for key exchange based on an asymmetric cryptosystem. In the first two messages of this protocol, A obtains certificates CA and CB that connect A and B with their public keys K_a and K_b, respectively. The exact form of CA and CB is not important for our purposes. The interesting part of the protocol is Message 3. There, A sends to B a key K_{ab} for subsequent encrypted communication between A and B, with a timestamp T_a. The protocol is:

$$\begin{array}{lll} \text{Message 1} & A \to S: & A, B \\ \text{Message 2} & S \to A: & CA, CB \\ \text{Message 3} & A \to B: & CA, CB, \\ & & \{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b} \end{array}$$

This third message is encrypted for both secrecy and authenticity. When A sends this message to B, it is important that no other principal obtain K_{ab}; the use of K_b provides this guarantee. Furthermore, the intent is that, when B receives the message, B should know that A sent it (because of the signature with K_a^{-1}). Finally, B should know that the message was intended for B (because of the use of K_b).

Unfortunately nothing provides this final guarantee, with dramatic consequences. Any principal B with which A opens communication can pretend to a third party C that it actually is A, for the duration of validity of the timestamp. For simplicity, we omit the exchanges which yield the public certificates CA, CB, and CC. When A opens communication with B,

$$\begin{array}{lll} \text{Message 3} & A \to B: & CA, CB, \\ & & \{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b} \end{array}$$

B removes the outer encryption, reencrypts with K_c, sends:

$$\begin{array}{lll} \text{Message 3'} & B \to C: & CA, CC, \\ & & \{\{K_{ab}, T_a\}_{K_a^{-1}}\}_{K_c} \end{array}$$

and C will believe that the message is from A. In particular, C might send sensitive information under K_{ab}, and B may see it when perhaps only A should.

The intended meaning of Message 3 is roughly "At time T_a, A says that K_{ab} is a good key for communication between A and B". Any adequate format for Message 3 should contain all of these elements expressly or by implication. The obvious one is:

$$\begin{array}{lll} \text{Message 3} & A \to B: & CA, CB, \\ & & \{\{A, B, K_{ab}, T_a\}_{K_a^{-1}}\}_{K_b} \end{array}$$

although the name A might be deducible from K_a^{-1}. It is important that this format must not be used for anything else; we return to this point in Section 7. □

Example 3.2 In [32, pp. 42–43], Woo and Lam present an authentication protocol based on symmetric-key cryptography. When B wants to check that it is in A's presence, it runs:

$$\begin{array}{lll} \text{Message 1} & A \to B: & A \\ \text{Message 2} & B \to A: & N_b \\ \text{Message 3} & A \to B: & \{N_b\}_{K_{as}} \\ \text{Message 4} & B \to S: & \{A, \{N_b\}_{K_{as}}\}_{K_{bs}} \\ \text{Message 5} & S \to B: & \{N_b\}_{K_{bs}} \end{array}$$

Here N_b is a nonce, S is a server, and K_{as} and K_{bs} are keys that A and B share with S. Basically, A claims his identity (Message 1); B provides a nonce challenge (Message 2); A returns this challenge encrypted under K_{as} (Message 3); B passes this message on to S for verification, bound with A's name under K_{bs} (Message 4); S decrypts using A's key and reencrypts under B's (Message 5). If S replies $\{N_b\}_{K_{bs}}$, then B should be convinced that A has responded to the challenge N_b.

The protocol is flawed. The connection between the messages is not sufficient. In particular, nothing connects B's query to S with S's reply. The protocol is therefore vulnerable to an attack, as follows. Suppose that B is willing to talk to A and to C roughly at the same time; A may be off-line. Then C can impersonate A:

Message 1	$C \to B:$	A
Message 1'	$C \to B:$	C
Message 2	$B \to A:$	N_b
Message 2'	$B \to C:$	N_b'
Message 3	$C \to B:$	$\{N_b\}_{K_{cs}}$
Message 3'	$C \to B:$	$\{N_b\}_{K_{cs}}$
Message 4	$B \to S:$	$\{A, \{N_b\}_{K_{cs}}\}_{K_{bs}}$
Message 4'	$B \to S:$	$\{C, \{N_b\}_{K_{cs}}\}_{K_{bs}}$
Message 5	$S \to B:$	$\{N_b''\}_{K_{bs}}$
Message 5'	$S \to B:$	$\{N_b\}_{K_{bs}}$

where N_b'' is the result of decrypting $\{N_b\}_{K_{cs}}$ using K_{as}. In Messages 1 and 1', C tells B that both A and C want to establish a connection. In Messages 2 and 2', B replies with two challenges; C receives one normally, and captures the other one, which was destined to A's address. In Messages 3 and 3', C replies to both challenges. On A's behalf, it can send anything. On its own behalf, C responds to the challenge intended for A. In Messages 4 and 4', B consults S about the two responses. Messages 5 and 5' are the replies from S. One of these replies matches nothing, while the other one contains the challenge intended for A. On the basis of these replies, then, B must believe that A is present.

The existence of this attack demonstrates that the messages in the protocol are not sufficiently explicit about the identity of the principals in question. (After we contacted them, Woo and Lam came to the same conclusion [33].) Reasoning as in Example 3.1, we may remove the flaw, by changing the last message of the protocol to

$$\text{Message 5} \quad S \to B: \quad \{A, N_b\}_{K_{bs}}$$

A further change is discussed in Example 6.2. □

Example 3.3 A more dramatic example is provided by the basic Internet protocol of Lu and Sundareshan [12, pp. 1016–1017]. This protocol is rather complicated for a detailed description. Its intent is to allow two principals A and B to obtain a session key, with the mediation of local servers and gateways.

On the other hand, the fundamental flaw of the protocol is rather simple. One immediately sees that neither A nor B ever receives a message that contains the other's name. Obviously, this opens the door for confusions between different connections. It also allows some easy attacks to defeat the protocol. After we contacted them, the authors published a correction [13], where names appear in messages explicitly. □

5 Encryption

The next group of principles and examples concern encryption. They are generally related to Principle 1, since they concern what encryption means and on what it does not mean.

5.1 The uses of encryption

As the examples below illustrate, encryption is used for a variety of purposes in the present context [19].

- Encryption is sometimes used for the preservation of confidentiality. In such case it is assumed that only intended recipients know the key needed to recover a message. When a principal knows K^{-1} and sees $\{X\}_K$, it may deduce that X was intended for a principal who knows K^{-1}; and it may even deduce that X was intended for itself, given additional information.

- Encryption is sometimes used to guarantee authenticity. In such case it is assumed that only the proper sender knew the key used to encrypt a message. The encryption clearly contributes to the overall meaning of the message. The extreme situation is that where a principal shows that a key is known by encrypting a null message or a timestamp.

- While encryption guarantees confidentiality and authenticity, it also serves in binding together the parts of a message: receiving

$\{X, Y\}_K$ is not always the same as receiving $\{X\}_K$ and $\{Y\}_K$. When encryption is used only to bind parts of a message, signature is sufficient. The meaning attached to this binding is rather protocol-dependent, and often subtle.

- Finally, encryption can serve in producing random numbers. There is a vast theory that explains the relation between one-way functions and random-number generators. At the level of abstraction that we consider, one typically assumes that random numbers are available without examining how they are constructed (but see Example 7.1).

There is considerable confusion about the uses and meanings of encryption. If the cryptography is asymmetric it may be obvious what is intended; if the cryptography is symmetric, it is generally not.

Principle 4

> Be clear as to why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security, and its improper use can lead to errors.

Example 4.1 The Kerberos protocol [16] is based on the original Needham-Schroeder protocol [20], but makes use of timestamps as nonces in order to remove flaws demonstrated by Denning and Sacco [6] and in order to reduce the total number of messages required. Like the Needham-Schroeder protocol on which it is based, the Kerberos protocol relies on symmetric-key cryptography. A slightly simplified version of the protocol goes:

Message 1	$A \to S:$	A, B
Message 2	$S \to A:$	$\{T_s, L, K_{ab}, B,$ $\{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
Message 3	$A \to B:$	$\{T_s, L, K_{ab}, A\}_{K_{bs}},$ $\{A, T_a\}_{K_{ab}}$
Message 4	$B \to A:$	$\{T_a + 1\}_{K_{ab}}$

Here, T_s and T_a are timestamps, and L is a lifetime. Initially the server S shares the keys K_{as} and K_{bs} with the principals A and B; after execution, A and B share K_{ab}. This protocol serves to illustrate different uses of encryption; we describe the protocol step by step:

- Encryption is not essential for Message 1. Without encryption, though, an attacker can flood S with requests for keys, by falsifying instances of Message 1. It is common for designers not to focus on this sort of vulnerability.

- Message 2 requires encryption: K_{ab} should remain confidential, and S should sign the message as a proof of authenticity.

- We may however question why double encryption is used in Message 2. Most probably, this use of double encryption is simply inherited from the Needham-Schroeder protocol (see Example 9.1). As in that protocol, double encryption does not add anything from the points of view of confidentiality or authenticity, and it is not entirely free of cost.

  It does provide a guarantee: when B receives the first part of Message 3, it knows that A must have extracted it from Message 2, and hence that A must have looked at Message 2. (Heintze and Tygar [9] discuss a similar use of encryption in a variant of the Otway-Rees protocol [23].) This interpretation of encryption is sound, but slightly unusual, and probably not what the protocol designers had in mind.

- In the second part of Message 3, encryption is used for an entirely different purpose: A encrypts T_a with K_{ab} in order to prove knowledge of K_{ab} near time T_a.

  In general, T_a may be a few hours newer than T_s. However, if T_a is not much different from T_s, the encryption is redundant: the use of double encryption in Message 2 gives adequate proof of knowledge of K_{ab}. In this case, the second part of Message 3 could be omitted altogether, and B could use T_s in Message 4. (Were we to propose a change in Kerberos, however, it would not be removing an encryption in Message 3 but rather eliminating the double encryption in Message 2, which would become $\{T_s, L, K_{ab}, B\}_{K_{as}}, \{T_s, L, K_{ab}, A\}_{K_{bs}}$.)

- The encryption in Message 4 serves an analogous purpose.

□

Examples 6.1 and 6.2, below, illustrate the interaction of encryption and nonces. In short, encryption is often used for binding when a nonce provides an association between a message and an implicit name. Following Principle 3, we make this missing name explicit. The use of both encryption and nonces is then much simpler and economical.

5.2 Signing encrypted data

Signature is used, as the name suggests, to indicate which principal last encrypted a message. It is frequently taken as also guaranteeing that the signing principal knew the message content. It is hard, but fortunately unnecessary to be precise about what knowing is. An informal notion is sufficient for stating the next principle:

Principle 5

> When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for privacy knows the content of the message.

Failure to follow this principle can lead to errors, as in the next example.

Example 5.1 The CCITT X.509 standard contains a set of three protocols using between one and three messages [5]. The protocols are intended for signed, secure communication between two principals, assuming that each knows the public key of the other.

The CCITT proposal has problems. We discuss one problem described in [4]; it appears already in the one-message protocol:

$$\text{Message 1} \quad A \rightarrow B: \quad A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}}$$

Here, T_a is a timestamps, N_a is a nonce (not used), and X_a and Y_a are user data. The X.509 protocol actually uses hashing to reduce the amount of encryption. We do not show this because it does not affect our argument about X.509.

The protocol is intended to ensure the integrity of X_a and Y_a, assuring the recipient of their origin, and to guarantee the privacy of Y_a. However, although Y_a is transferred in a signed message, there is no evidence to suggest that the sender is actually aware of the data sent in the private part of the message. This corresponds to a scenario where some third party intercepts a message and removes the existing signature while adding his own, blindly copying the encrypted section within the signed message. This problem can be avoided by several means, the simplest of which is to sign the secret data before it is encrypted for privacy. □

A particular case of the principle concerns hash functions:

Example 5.2 It is common to use hash functions in order to save on encryption with asymmetric-key systems (see for example [25, 10]). In particular, A can send a signed, confidential message to B as follows:

$$\text{Message 1} \quad A \rightarrow B: \quad \{X\}_{K_b}, \{H(X)\}_{K_a^{-1}}$$

where H is a one-way hash function. When A sends this message, only B discovers X, and B knows that A signed the hash of X. For example, if X is a request for an operation, B may then infer that A supports X. We should think of one-way hashing as encryption, and then Principle 5 applies. In this instance, it means that B cannot be certain that A knew X. For example, if X is a secret such as a password, B cannot be certain that A knew the secret; A may have received $H(X)$ from a friend. □

In general, we recommend careful examination of those protocols that require a principal to sign material that is already encrypted, and such that the principal cannot decrypt it. On the other hand, signing before encrypting is not a bill of health; see Example 3.1.

6 Timeliness

An important part of the meaning of a message is made up of temporal information. Further, one common precondition for acting upon a message is that there is reason to believe that the message is fresh, and hence not a replay of an old one. This has to be inferred from something in the message, and evidently whatever this is should be bound together with the rest of the message so that the magic talisman cannot be attached to a message being replayed. It is important to understand properly how the freshness component works, and what is being assumed about it.

The next group of principles and examples concern time. They all address what must be assumed about proofs of timeliness, and what they actually prove.

6.1 Timestamps, sequence numbers, and other nonces

When guarding against replay of messages from an earlier run of the same protocol it is common to use nonces as part of a challenge-response exchange. A message is sent which leads to a reply which could only have been produced in knowledge of the first message. The objective is to guarantee that the second message is made after the first was sent, and sometimes to bind the two together. There is sometimes confusion about nonces—are they guaranteed new, random, unpredictable? Whence we propose:

Principle 6

> Be clear what properties you are assuming about nonces. What may do for ensuring temporal succession may not do for ensuring association—and perhaps association is best established by other means.

Example 6.1 In [23], Otway and Rees describe the following protocol. It allows two parties A and B to establish a shared key K_{ab}, with the help of a server S with whom they share keys K_{as} and K_{bs}, respectively, using the nonces M, N_a, and N_b:

$$\begin{array}{lll}
\text{Message 1} & A \rightarrow B: & M,\ A,\ B, \\
 & & \{N_a,\ M,\ A,\ B\}_{K_{as}} \\
\text{Message 2} & B \rightarrow S: & M,\ A,\ B, \\
 & & \{N_a,\ M,\ A,\ B\}_{K_{as}}, \\
 & & \{N_b,\ M,\ A,\ B\}_{K_{bs}} \\
\text{Message 3} & S \rightarrow B: & M,\ \{N_a,\ K_{ab}\}_{K_{as}}, \\
 & & \{N_b,\ K_{ab}\}_{K_{bs}} \\
\text{Message 4} & B \rightarrow A: & M,\ \{N_a,\ K_{ab}\}_{K_{as}}
\end{array}$$

This is the first protocol analyzed in [4]. Perhaps because of our relative inexperience, we were rather bold in the idealization of this protocol—in assigning meanings to these messages. As a consequence, we suggested in passing that the encryption of N_b in Message 2 is unnecessary. As Mao and Boyd have since explained in detail [14], the encryption of N_a and N_b is essential: because N_a and N_b are bound with the names A and B by encryption in Messages 1 and 2, they can serve as secure references to A and B in Messages 3 and 4. Encryption is being used not for secrecy, but for binding; nonces are exploited not only as proofs of timeliness but as substitutes for names.

Much encryption can be avoided when names are included in S's reply:

$$\begin{array}{lll}
\text{Message 1} & A \rightarrow B: & A,\ B,\ N_a \\
\text{Message 2} & B \rightarrow S: & A,\ B,\ N_a,\ N_b \\
\text{Message 3} & S \rightarrow B: & \{N_a,\ A,\ B,\ K_{ab}\}_{K_{as}}, \\
 & & \{N_b,\ A,\ B,\ K_{ab}\}_{K_{bs}} \\
\text{Message 4} & B \rightarrow A: & \{N_a,\ A,\ B,\ K_{ab}\}_{K_{as}}
\end{array}$$

The protocol is not only more efficient but also conceptually simpler after this modification. □

Example 6.2 Example 3.2 describes a protocol due to Woo and Lam. Looking back at the use of encryption in that protocol, we notice that the purpose of encryption in Message 4 is to bind two parts of a message. Looking back at the use of nonces, we notice that N_b provides only a proof of freshness, but not an association to the name A as was intended.

As we argue in Example 3.2, Message 5 should mention the name A explicitly for the sake of security. With that change, the binding of Message 4 becomes unnecessary. Further, N_b needs to be viewed only as a proof of freshness. The protocol is now simply:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & A \\ \text{Message 2} & B \rightarrow A: & N_b \\ \text{Message 3} & A \rightarrow B: & \{N_b\}_{K_{as}} \\ \text{Message 4} & B \rightarrow S: & A, \{N_b\}_{K_{as}} \\ \text{Message 5} & S \rightarrow B: & \{A, N_b\}_{K_{bs}} \end{array}$$

□

It is not essential for nonces to be unpredictable. In fact, the value of a counter makes a proper nonce. However, predictable nonces should be used with caution:

Principle 7

> The use of a predictable quantity (such as the value of a counter) can serve in guaranteeing newness, through a challenge-response exchange. But if a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay a response.

Example 7.1 Protocols that rely on synchronized clocks must be accompanied by protocols to access time servers. These protocols cannot themselves rely on synchronized clocks, but they can rely either on random nonces or on predictable nonces.

Using random nonces, we may have:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & A, N_a \\ \text{Message 2} & S \rightarrow A: & \{T_s, N_a\}_{K_{as}} \end{array}$$

where T_s is the current time and N_a is a random nonce, used as a challenge. After this exchange, A accepts T_s as the current time if the response arrived reasonably soon after the challenge. Reiter exploits random nonces roughly in this manner [24].

This protocol would not work if N_a were predictable. An attacker C could make A set its clock back: early on, C makes a request for the current time using a future value of the nonce, saves S's response, and then forwards the response to A when A uses this value in a challenge.

When N_a is predictable, it should be protected:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & A, \{N_a\}_{K_{as}} \\ \text{Message 2} & S \rightarrow A: & \{T_s, \{N_a\}_{K_{as}}\}_{K_{as}} \end{array}$$

The attack is no longer possible. Note that it is not important for N_a to remain secret (and after all we have assumed it is predictable). The encryption in Message 1 serves to construct a quantity $\{N_a\}_{K_{as}}$ that only A and S can predict from one that anyone can predict.

A similar exchange arises in the context of Kerberos. Neuman and Stubblebine suggest using Kerberos itself to obtain the time from a time server [22]. The quantity used as a nonce is roughly predictable: it is an incorrect timestamp; since it is encrypted, we expect no difficulties. □

Freshness can also be proved by the use of timestamps. Timestamps are appealing because they seem easier to use than random numbers. However, their use is not always correct. There are number of aspects of prudent practice in the use of timestamps, and they are often misunderstood. One use of timestamps is as a kind of nonce. In this case the ultimate user of the timestamp, as part of a response, is the same as the originator of the challenge of which the timestamp was part. This style of use does not depend on clock synchronization at all, but does need care because the timestamp may be to a large extent predictable. Another style of use does depend on clock synchronization. The recipient of a message looks at a timestamp in it, and only accepts the message if the timestamp is within a reasonable interval of the recipient's local time. In this case we have:

Principle 8

> If timestamps are used as freshness guarantees by reference to absolute time, then the difference between local clocks at various machines must be much less than the allowable age of a message deemed to be valid. Furthermore, the time maintenance mechanism everywhere becomes part of the trusted computing base.

Example 8.1 Timestamps have received abundant attention in the authentication literature. Gong, in particular, has described problems arising from the use of incorrect timestamps [8]. Therefore, we keep this example brief, summarizing Gong's example for the Kerberos system.

In Kerberos, as elsewhere, a principal with a slow clock is exposed to all sorts of difficulties, since the principal may mistake expired certificates for current ones. It is more interesting that a fast clock can also be an opportunity for attackers. If a principal A signs a request at time T_0 using a timestamp T, with $T_0 < T$, an attacker C can replay this request near time T. The effect of the request at time T may benefit C, for example if C is using A's workstation at time T.

Bellovin and Merritt have discussed further problems in Kerberos, some of them in the use of timestamps. □

6.2 What is fresh: use vs. generation

Roughly, a bit-pattern is fresh if any message that contains it must be recent. Clearly, it does not suffice that the bit-pattern participate in one recent message, if it may also participate in old ones. This observation is most important for keys:

Principle 9

> A key may have been used recently, for example to encrypt a nonce, yet be quite old, and possibly compromised. Recent use does not make the key look any better than it would otherwise.

Example 9.1 The Needham-Schroeder protocol and the Kerberos protocol are similar in structure and in goal; the Needham-Schroeder protocol reads:

$$\begin{array}{lll}
\text{Message 1} & A \rightarrow S: & A,\, B,\, N_a \\
\text{Message 2} & S \rightarrow A: & \{N_a,\, B,\, K_{ab}, \\
 & & \;\{K_{ab},\, A\}_{K_{bs}}\}_{K_{as}} \\
\text{Message 3} & A \rightarrow B: & \{K_{ab},\, A\}_{K_{bs}} \\
\text{Message 4} & B \rightarrow A: & \{N_b\}_{K_{ab}} \\
\text{Message 5} & A \rightarrow B: & \{N_b + 1\}_{K_{ab}}
\end{array}$$

As in Kerberos, only A makes contact with S, who provides A with the session key, K_{ab}, and a certificate encrypted with B's key K_{bs} conveying the session key to B. Then B decrypts this certificate and carries out a nonce handshake with A to be assured that A is present currently, since the certificate might have been a replay. As explained in Section 7, Message 5 contains $N_b + 1$ rather than N_b in order to distinguish this message from Message 4.

Messages 4 and 5 of the Needham-Schroeder protocol were intended to convince B that A is present and active. They do not (and in fact were not intended to) convince B that K_{ab} is fresh, and it was pointed out by Denning and Sacco that compromise of a session key could allow an intruder to deceive B [6]. Once the importance of freshness of K_{ab} is recognized, a solution may be found by using timestamps, as suggested by Denning and Sacco. In another solution, described in [21], B send a nonce to S, and then S includes it in its certificate. □

Example 9.2 In [29], Varadharajan, Allen, and Black present several protocols for delegation in distributed systems. We take as an example the one for delegation in a Kerberos environment [29, p. 273]. In this protocol, client B shares the key K_{bt} with the authentication server; B has generated a timestamp T_b and wants a key K_{bs} to communicate with another server S. The authentication server gives K_{bs} and $\{K_{bs}\}_{K_{bt}}$ to S. Then S constructs $\{T_b + 1\}_{K_{bs}}$, and sends:

$$\begin{array}{lll}
\text{Message 5} & S \rightarrow B: & S,\, B,\, \{T_b + 1\}_{K_{bs}}, \\
 & & \{K_{bs}\}_{K_{bt}}
\end{array}$$

The authors reason:

> Having obtained K_{bs}, B is able to verify using T_b that S has replied to a fresh message, so that the session key is indeed fresh.

However, B obtains no proof that K_{bs} is fresh. All that B can deduce is that K_{bs} has been used recently—but it may be an old, compromised key. □

7 Recognizing messages and encodings

It seems important that principals recognize messages for what they are, and can associate them correctly with the current step of whatever protocol they are executing. There are two possible forms of confusion (which could in principle happen together): between the current message and a message of similar purpose form a previous run of the protocol, and between the current message and a message belonging elsewhere in the protocol, or to another protocol. Snekkenes [27] and Syverson [28] have constructed examples of protocols where these confusions can arise.

We believe that these confusions are less important when all our principles are correctly followed. If a message says what it means then we have no reason to be concerned with its context. The message is meaningful (or meaningless) on its own, whether we know that it belongs in a particular protocol instance or not.

Still, mapping a message to the appropriate protocol instance is convenient when it contributes to the compact encoding of the message. For example, Message 1 of the Wide-mouthed-frog protocol always means something of the form: "the signer (with key K_{as}) says at time T_a that K_{ab} is a good key to talk to B" (see Example 11.2). If the principal who receives a message can be certain that the message is Message 1 of an instance of the Wide-mouthed-frog protocol, then the message can be encoded compactly: $\{T_a, B, K_{ab}\}_{K_{as}}$.

We arrive at the following recommendation:

Principle 10

> If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

Mapping a message to the appropriate protocol instance is often trivial if the message obeys our other principles. If the message contains sufficient timeliness guarantees and sufficient names, then the current instance cannot be confused with an old instance, or an instance for other principals. It could be confused with a concurrent instance for the same principals.

Next we give an example where this principle is relevant, but where other more important principles apply as well.

Example 10.1 If signature or confidentiality are mediated by symmetric-key encryption then a particular form of confusion is associated with the direction in which a message is intended to pass.

In the Needham-Schroeder protocol, a participant sends a challenge N_b and receives $N_b + 1$ as a response (see Example 9.1):

Message 4	$B \rightarrow A:$	$\{N_b\}_{K_{ab}}$
Message 5	$A \rightarrow B:$	$\{N_b + 1\}_{K_{ab}}$

The purpose of incrementing N_b is to distinguish the challenge from the response. Without this increment, an attacker could send B's message back to B, who could mistake it for A's reply. The purpose of incrementing a nonce has often been misunderstood. For example, a "+1" operation appears in Kerberos, where it is unnecessary.

The messages would be clearer if they were rewritten:

Message 4	$B \rightarrow A:$	$\{$N-S Message 4: $N_b\}_{K_{ab}}$
Message 5	$A \rightarrow B:$	$\{$N-S Message 5: $N_b\}_{K_{ab}}$

though in fact any way of making the two messages different will do. (This is an instance of our suggestion to Syverson mentioned in [28].)

Guided by the principle that messages should say what they mean, however, we ought to rewrite the messages:

Message 4	$B \rightarrow A:$	N_b, $\{B$ says that K_{ab} is a good key to talk to A, sometime after receiving $K_{ab}\}_{K_{ab}}$
Message 5	$A \rightarrow B:$	$\{A$ says that K_{ab} is a good key to talk to B, sometime after receiving $N_b\}_{K_{ab}}$

Of course, shorter encodings of these meanings can be constructed. Not only there is no risk of confusion between these two messages: each of them is self-contained, and it is not important to know that they are part of a particular instance of the Needham-Schroeder exchange. □

8 Trust

The use of timestamps makes explicit for the first time a question of trust. When can a principal A rely on another principal B putting a correct timestamp in a message? The answer usually given is that this is acceptable if A trusts B in relation to timestamps.

The idea of trust is pervasive and a little elusive. A careful classification of types of trust is given in [34] and is taken further by Klein in her doctoral thesis. There are questions both of practice and philosophy to do with trust relations—for example whether they are transitive or not—which it would not be appropriate to pursue here. We may simply say that A trusts B in regard to some function if a loss of security to A could follow from B not behaving in the specified way; it is usually difficult or impossible for A to verify B's good behavior.

There is some measure of trust involved whenever one principal acts on the content of a message from another. It is essential that this trust be properly understood.

Principle 11

> The protocol designer should know which trust relations his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit though they will be founded on judgement and policy rather than on logic.

Example 11.1 Complete loss of security could follow from a Kerberos server issuing wrong timestamps. The server, and its source of time, must be trusted by all concerned. This, it may be pointed out, is a case in which clients can to some extent monitor the good behavior of the trusted server because the correct time is public and global. If a client reads GPS time it will have reason for suspicion if Kerberos' time is much at variance. □

Example 11.2 The Wide-mouthed-frog protocol uses symmetric-key cryptography and an authentication server. It transfers a key from A to B via S in only two messages:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S: & A, \{T_a, B, K_{ab}\}_{K_{as}} \\ \text{Message 2} & S \rightarrow B: & \{T_s, A, K_{ab}\}_{K_{bs}} \end{array}$$

First, A sends a session key K_{ab} to S, including a timestamp T_a. Then S checks T_a and forwards the message to B, together with its own timestamp T_s. Finally, B checks T_s and accepts the session key K_{ab} for communication with A. Thus, A is trusted to choose a session key. This kind of trust is often thought unacceptable because of the quality requirements placed on key generation such as secrecy, non-repetition, unpredictability, and doubtless more. □

Example 11.3 Principals associate public keys with other principals by consulting public-key certificates. These certificates can be issued by certification authorities (CAs). CAs are trusted to certify a key only after proper steps have been taken to identify the principal that owns it. Since there is no global source of authority, it is not surprising that this is an area where questions of transitivity of trust come up. It may happen that the only way A can find out B's public key is by accepting a certificate from CA_1 for CA_2's public key which is used to sign a certificate for CA_3's public key ... which is used to sign a certificate for B's public key, for example. In this case A knows and trusts CA_1 but has never heard of the other certification authorities—and maybe even distrusts them. □

Example 11.4 It is usually taken for granted that the two principals in an authentication dialogue are honestly working to the common end of establishing a secure communication channel for subsequent use. This is not always the case, as may be seen in communication between potential enemies or between security forces and terrorists. Possible sorts of bad behavior are disclosure of keys and forgery of messages. Therefore, if this assumption is made in a particular case then it should be explicit. □

Example 11.5 An access control list (ACL) is a statement of trust [1]: if principal A appears on the ACL for an operation then A is trusted when it says that the operation should be performed (that is, when it makes a request). It can be a complex matter to determine whether the statement of trust that the ACL represents is appropriate. Often, the question of whether it is appropriate makes little sense, particularly in the context of completely discretionary access control policies. Nonetheless, understanding ACL's and their consequences is crucial for security. □

In practice it is not very common for complicated inferences about trust to be necessary. With the exception of the chains of trust of Example 11.3, which are likely to be simpler in practice than they might be in theory, it is usually not difficult to isolate the trust relations being relied on in a particular circumstance. It is valuable to do so explicitly, because this may lead to useful debate about the appropriateness of these trust relations.

9 Conclusion

We have found the principles and examples described in this paper useful in our own work. Perhaps it is because of this that they bear a certain subjective character. We do however believe that they respond to an immediate general need, in a discipline where some basic mistakes appear in print several times.

Many of our suggestions can be embodied in development methods and in formalisms. While these are helpful, we tried to emphasize an informal understanding of some issues essential for security. We hope that our guidelines will contribute to the improvement of the practice of designing cryptographic protocols.

Acknowledgments

We have benefited from discussions with Mike Burrows and Butler Lampson. In particular, we discovered many of the examples in this paper in collaboration with Mike Burrows. The authors of the papers from which we drew our examples have also been helpful.

Raphael Yahalom, Michael Reiter, and anonymous referees all made useful comments on earlier versions of this paper.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A Calculus for Access Control in Distributed Systems. *ACM Transactions on Programming Languages and Systems* Vol. 15, No. 4, September 1993, 706–734.

[2] S.M. Bellovin and M. Merritt. Limitations of the Kerberos Authentication System. *Computer Communication Review* Vol. 20, No. 5, October 1990, pp. 119–132.

[3] C. Boyd and W. Mao. Limitations of Logical Analysis of Cryptographic Protocols. *Eurocrypt '93*, to appear.

[4] M. Burrows, M. Abadi, and R.M. Needham. A Logic of Authentication. *Proceedings of the Royal Society of London A* Vol. 426, 1989, pp. 233–271. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.

[5] CCITT. CCITT Blue Book, Recommendation X.509 and ISO 9594-8: The Directory-Authentication Framework. Geneva, March 1988.

[6] D.E. Denning and G.M. Sacco. Timestamps in Key Distribution Protocols. *CACM* Vol. 24, No. 8, August 1981, pp. 533–536.

[7] U. Feige, A. Fiat, A. Shamir. Zero Knowledge Proofs of Identity. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 1987, pp. 210–217.

[8] L. Gong. A Security Risk of Depending on Synchronized Clocks. *Operating Systems Review* Vol. 26, No. 1, January 1992, pp. 49–54.

[9] N. Heintze and J.D. Tygar. Timed Models for Protocol Security. CMU Technical Report CMU-CS-92-100, January 1992.

[10] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed

Systems: Theory and practice. *ACM Transactions on Computer Systems* Vol. 10, No. 4, November 1992, 265–310.

[11] A. Liebl. Authentication in Distributed Systems: A Bibliography. *Operating Systems Review* Vol. 27, No. 4, October 1993, pp. 31–41.

[12] W.P. Lu and M.K. Sundareshan. Secure Communication in Internet Environments: A Hierarchical Key Management Scheme for End-To-End Encryption. *IEEE Transactions on Communications* Vol. 37, No. 10, October 1989, pp. 1014–1023.

[13] W.P. Lu and M.K. Sundareshan. Enhanced Protocols for Hierarchical Encryption Key Management for Secure Communication in Internet Environments. *IEEE Transactions on Communications* Vol. 40, No. 4, April 1992, pp. 658–660.

[14] W. Mao and C. Boyd. Towards Formal Analysis of Security Protocols. *Proceedings of the Computer Security Foundations Workshop VII*, 1993, pp. 147–158.

[15] G. Medvinsky and B.C. Neuman. NetCash: A Design for Practical Electronic Currency on the Internet. *Proceedings of the 1993 ACM Conference on Computer and Communications Security*, pp. 102–106.

[16] S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer. Kerberos Authentication and Authorization System. *Project Athena Technical Plan* Section E.2.1, MIT, July 1987.

[17] J.H. Moore. Protocol Failures in Cryptosystems. *Proceedings of the IEEE* Vol. 76, No. 5, May 1988, pp. 594–602.

[18] National Bureau of Standards. Data Encryption Standard. FIPS Pub. 46, January 1977.

[19] R.M. Needham. Cryptography and Secure Channels. *Distributed Systems, 2nd Ed.*, S. Mullender (editor), ACM Press, 1993, 231-241.

[20] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *CACM* Vol. 21, No. 12, December 1978, pp. 993–999.

[21] R.M. Needham and M.D. Schroeder. Authentication Revisited. *Operating Systems Review* Vol. 21, No. 1, January 1987, p. 7.

[22] B.C. Neuman and S.G. Stubblebine. A Note on the Use of Timestamps as Nonces. *Operating Systems Review* Vol. 27, No. 2, April 1993, pp. 10–14.

[23] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review* Vol. 21, No. 1, January 1987, pp. 8–10.

[24] M.K. Reiter. A Security Architecture for Fault-Tolerant Systems. Ph.D. Thesis, Cornell University. Available as Technical Report 93-1367, Department of Computer Science, Cornell University, July 1993.

[25] R. Rivest. The MD4 Message Digest Algorithm. *Advances in Cryptology: Crypto '90*, Springer-Verlag, 1991, pp. 303–311.

[26] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM* Vol. 21, No. 2, February 1978, pp. 120-126.

[27] E. Snekkenes. Roles in Cryptographic Protocols. *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pp. 105–119.

[28] P. Syverson. On Key Distribution Protocols for Repeated Authentication. *Operating Systems Review* Vol. 27, No. 4, October 1993, pp. 24–30.

[29] V. Varadharajan, P. Allen, S. Black. An Analysis of the Proxy Problem in Distributed Systems. *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 255–275.

[30] V.L. Voydock and S.T. Kent. Security Mechanisms in High-Level Network Protocols, *Computing Surveys* Vol. 15, No. 2, 1983, pp. 135–171.

[31] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, 1993, pp. 256–269.

[32] T.Y.C. Woo and S.S. Lam. Authentication for Distributed Systems. *Computer* Vol. 25, No. 1, January 1992, pp. 39–52.

[33] T.Y.C. Woo and S.S. Lam. A Lesson on Authentication Protocol Design. Manuscript, 1993.

[34] R. Yahalom, B. Klein, T. Beth. Trust Relations in Secure Systems—A Distributed Authentication Perspective. *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, pp. 150–164.