Strategies against Replay Attacks

Tuomas Aura*
Digital Systems Laboratory, Helsinki University of Technology
P.O.Box 1100, FIN-02015 HUT, Finland
Tuomas.Aura@hut.fi

Abstract

The goal of this paper is to present a set of design principles for avoiding replay attacks in cryptographic protocols. The principles are easily applied to real protocols and they do not consume excessive computing power or communications bandwidth. In particular, we describe how to type-tag messages with unique cryptographic functions, how to inexpensively implement the full information principle with hashes, and how to produce unique session keys without assuming mutual trust between the principals. The techniques do not guarantee security of protocols, but they are concrete ways for improving the robustness of the protocol design with relatively low cost.

1 Introduction

Most attacks against authentication and key distribution protocols are based on recording messages or their parts and replaying them in another context. The messages can be redirected to other recipients than originally intended, or they can be repeated in different protocols, protocol runs, or transmission steps. (Syverson [12] gives an exhaustive taxonomy of the replay attacks.)

It is commonly suggested in the literature that successful attacks against cryptographic protocols are a result of bad or lacking design principles, and that good principles can lead to protocols that are significantly more robust [1, 4, 10, 13, 2]. The principles are, however, usually presented in form of warning examples of how protocols should not be built. The warnings are of importance to anyone who wants to inspect protocols for the most common structural flaws but, unfortunately, there are few specific instructions for constructing protocols that avoid the pitfalls and fulfill the good principles.

In this paper, we present some strategies for protocol design that systematically lead to properties described as desirable in the literature. Above all, we stress that the techniques should be implementable in real protocols at a reasonable cost in computation and bandwidth. This is essential, because designers of concrete protocols often struggle with tight performance constraints. We achieve the low resource consumption by using inexpensive hash functions and by utilizing the cryptographic functions and redundancy that would in any case exist in the protocols.

The ideas presented in this paper are neither sufficient not necessary to make all protocols secure. However, following these guidelines in protocol design results in conceptually simple protocols whose security is easier to reason about.

In Sec. 2 we tackle the problem of data types. Type tagging can be used to bind messages to their specific places in the protocol so that they cannot be replayed elsewhere. We show how the data can be inexpensively tagged with static type information by creating unique cryptographic functions for all purposes. Sec. 3 continues on methods for binding data to its intended use. By including in all messages a hash of all information from the earlier messages in the same protocol run, the protocol designer can make maximal information available for checking consistency of protocol runs. Sec. 4 gives more details on comparing the principals' observations of the protocol run. In Sec. 5 we point out that the user of a session key should be familiar with the trust assumptions made in the key exchange process. Sec. 6 describes techniques for generating unique session keys without making assumptions about mutual trust between the principals, and for binding keys to their intended uses. The mechanisms protect against replay of connection data. Sec. 7 concludes the discussion and outlines some goals for future research.

2 Implicit typing by unique functions

In the replay attacks, a message or a fragment of a message is taken out of its original context and replayed as a part of another message, in another protocol run, or even in a run of another protocol. Logically, replay attacks are

*This work was funded by Helsinki Graduate School in Computer Science and Engineering (HeCSE).

prevented by binding the messages and components of the messages to their correct context. This can be done by including enough information in the messages so that they are recognized to belong to a certain state of a certain protocol run. A straightforward way to bind the context information to the messages is explicit typing. Carlsen [4] gives a comprehensive list of type information that can be attached to messages and data items:

- protocol identifier
- transmission step identifier
- message subcomponent identifier
- primitive type of data items.
- protocol run identifier

All messages and their components down to individual indecomposable data items can be recursively tagged with the above type information. Except for the protocol run identifier, the tags are static type information that can be represented with static labels on the message data structures.

In practice, however, full explicit typing is avoided for performance reasons. Instead, the types of most data items are assumed to be implicit from their context. For example, it is usually sufficient to tag the outmost signed or encrypted submessages with their composite types. Submessages can be left untagged. Full recursive tagging is also needless from security viewpoint, because the type-tags are meaning full only when bound to the data with some kind of cryptographic authentication. Sometimes the type can be deduced from the structure of the message. In that case, it is necessary for the message to contain sufficient redundancy so that the receiver can recognize it.

Example 1 (X.509) Messages created from ASN.1 specifications usually have full explicit typing including all parts of the type information listed above, expect for the protocol run identifier. The X.509 standard [5] has ASN.1 definitions for public key certificates, but not for the authentication protocol messages. This is a clear indication that explicit typing is considered too expensive in an authentication protocol. Instead, the receiver has to check the message lengths and structures in order to distinguish between the messages. (The full protocol is listed in Example 5.) ∎

Implicit typing should always be designed with utmost care. *Ad hoc* solutions such as the differing message lengths in the X.509 protocol, can lead to serious mistakes if messages with resembling structure are used for other purposes. Therefore, a consistent and complete scheme for implicit typing is needed. We now construct such a scheme by generalizing an idea that has been used in several existing protocols.

We observed above that the messages and submessages that need to have type tagged on them are always composed with some cryptographic function that provides a level of integrity protection. Otherwise, the tagging the data would not increase the security of the protocol. The most common integrity-protecting functions are encryption or secure hash. Since we only need to consider messages composed with cryptographic functions, it is possible to give the messages implicit types by using a different cryptographic function for every different submessage. That is, using unique, protocol-specific encryption functions and hashes for every application makes it impossible to copy messages from one protocol, step or submessage to another. Such unique functions can actually be found in many existing protocols.

Example 2 (GSM) In the GSM authentication between a mobile station (MS) and a fixed network (NET) [7], two application specific keyed one-way functions are used. One function, $A3$, gives the response in a challenge–response scheme and another one, $A8$, computes the session key. If Ki is the shared master key between MS and NET, and $RAND$ is a random challenge, the response value is given by $A3(Ki, RAND)$ and the new session key by $A8(Ki, RAND)$. The two functions are logically similar and can have almost identical implementations. It is only important that they compute different, unrelated functions.

1.	$MS \longrightarrow NET$	$IMSI$	(MS's identity)
2.	$NET \longrightarrow MS$	$RAND$	(challenge)
3.	$MS \longrightarrow NET$	$A3(Ki, RAND)$	(response)

The function $A3$ is used exclusively for computing the response values in the last step of the GSM key exchange protocol and nowhere else in the world. Therefore, the values of $A3$ cannot be copied to another protocol, or to another place in the same protocol. Neither can messages from any other protocol or any other information in this protocol be used in place of Message 3. Notably, it is impossible to mix the session keys and response values. ∎

The keyed one-way functions in the GSM authentication have been designed for the particular use. This guarantees that the functions are not used anywhere else, but it is, of course, not advisable that every protocol designed would create new cryptographic algorithms. Instead, well known algorithms can be instantiated with application specific parameters that contain enough redundancy so that the same parameters will not be accidentally used elsewhere. In particular, one-way hash functions, block-chaining encryption functions and stream ciphers can be initialized with a constant type value. The initialization constant should be randomly chosen and fixed at protocol design time for each use of the cryptographic functions in the protocol. It should be long enough so that no two protocols are likely to use the same value for the same cryptographic function. An alternative approach would be to register or widely publicize the used constants and to hope that designers of related protocols check for the previously reserved values.

Example 3 (GSM) The keyed one-way functions in the GSM authentication could be replaced with the following logically equivalent (although computationally more expensive) functions:

$$A3(Ki, RAND) = HASH(R_1, Ki, HASH(Ki, RAND))$$
$$A8(Ki, RAND) = HASH(R_2, Ki, HASH(Ki, RAND))$$

where $HASH$ is a one-way hash function (for example SHA) and R_1 and R_2 are two randomly chosen 64-bit constants. ■

Example 4 (Wide-mouth-frog) The Wide-mouth-frog key distribution protocol has two messages with similar format. The consequence is that an attacker can keep a protocol run alive by repeatedly playing Message 2 in place of Message 1 [2]. (Note that the encryption functions here are assumed to also protect the integrity of the messages.)

1.	$A \longrightarrow KDC$	$A, E_A(T_A, B, K)$
2.	$KDC \longrightarrow B$	$E_B(T_B, A, K)$

Normally, two properties of the protocol are blamed for the replay attack. First, the time stamp is renewed from Message 1 to Message 2. Second, the identical message contents make it impossible to differentiate between them. It is, however, common practice to stamp all messages with their actual sending time. Furthermore, the message contents are determined by the need to transfer information and it not desirable to artificially change the message length or structure because of typing. Instead, the problem can be solved by using two different encryption functions.

1.	$A \longrightarrow KDC$	$A, E_A^{\text{WMF1}}(T_A, B, K)$
2.	$KDC \longrightarrow B$	$E_B^{\text{WMF2}}(T_B, A, K)$

These (integrity-protecting) encryption functions can be implemented, for example, by encrypting the message (and a message authentication code) with an algorithm that is initialized with two different randomly chosen type identifiers. ■

Creating unique functions for all submessages costs significantly less than explicit typing of all messages. Normally, the lengths of the transmitted messages do not change. For example, signatures and message authentication codes can be made unique by initializing the message hash function with different values. (In our notation, the values of the functions S and MAC, i.e. signatures and message authentication codes, do not contain the message itself but only the signature part.)

$$S_A^{\text{unique}}(M) = S_A(HASH(TypeTag, M))$$
$$MAC_K^{\text{unique}}(M) = HASH(TypeTag, Ki, HASH(Ki, M))$$

The initialization constants are known by everyone and are not transmitted. Therefore, the cost of making the functions unique is negligible. If implemented with care, also unique encryption functions are only slightly more expensive to compute than standard ones. For example, the encryption key can be hashed with the message identifier.

$$E_K(M) = E_{HASH(K, TypeTag)}(M)$$

If a block cipher is used in CBC-mode, the message identifier should rather be hashed into the initialization vector. If non-standard hash algorithms are used, this may be more robust than hashing the message identifier with the encryption key, because a flaw in the hash algorithm would not endanger the secrecy of the encrypted message.

$$IV' = HASH(IV, TypeTag)$$
$$C_1 = E_K^{\text{block}}(P_1 \oplus IV')$$

The advantage in comparison to explicit tags is that the type identifier is merged into a message digest or initialization vector, but the redundant identifier itself is not transmitted. The idea is that a certain amount of redundancy must be present in a message so that the receiver can perform necessary cryptographic checks. The amount of necessary redundancy is constant (usually 64 bits, or 128 bits where conflict-freeness is required). If new redundant information needs to be added, the total amount can be reduced to the same constant by combining the redundancy from the two sources, e.g. by hashing the redundant data items together.

As we have seen in the above examples, unique cryptographic functions can serve as implicit identifier for protocols, transmission steps, submessages and primitive data items. We summarize the idea of this section in the following rule:

> **Strategy 1** Use a unique cryptographic function for each submessage in order to tag the submessages with their static data types. Create the unique functions by parameterizing standard cryptographic functions.

It still remains an open question, how to tag the messages with a protocol run identifier. We will not do so explicitly but the techniques presented in the next section can be thought of as tagging messages with hashes of the entire protocol run.

3 Hashed full information

Many protocol flaws are caused by excessive removal of redundancy from the protocol messages. The protocol designer is tempted to minimize the amount of data transfered, but it is often difficult to see which data items are necessary for the correctness of the protocol and which are not. Therefore, many authors stress explicitness as the foremost design principle for cryptographic protocols [2, 4]. All information particular to the protocol run, such as names of the principals, nonce values and session keys, may make it more difficult to replay messages in the wrong context if included in the protocol messages as redundant data items. For example, several classic protocol failures would have been prevented by explicitly stating the name of the intended recipient in the messages (Denning–Sacco [1], Needham–Schröder public-key protocol [9], original X.509 standard [8]. In Sec. 2 we saw how static data types can be implicitly included in messages at low cost. The goal of this section is to present a systematic technique with which also dynamic, run-specific, information can be inexpensively attached to every message.

Woo and Lam [13] take the idea of explicitness to the extremes by advocating the *principle of full information*: the principals should include in all messages all information that is in their possession and relevant to the protocol run. Although this is a helpful principle to keep in mind, it is unfortunately too expensive one to follow literally. Many techniques such as the implicit typing of Sec. 2 can be used to implicitly include information in the messages without actually transferring the full data. This kind of saving of bandwidth may seem unnecessary from a theoretical viewpoint, but it is essential before the protocols can be implemented in actual communications systems. In mobile communications and smartcard interfaces, for example, the use of security technology is severely limited by the high cost of data transfer.

Thus, we would like to enjoy the security added assurance of full information but are reluctant to transfer all the redundant data. Here we should remember that in most cases the redundant data is already known by the receiver and it is used only for comparison with the expected values. It is not necessary to transfer all the data but only a conflict-free hash of it with enough redundancy so that the receiver can reliably compare the received value with a hash of the expected data. By cryptographic hashing, any amount of redundant data can be compressed to a constant length hash value. This means that all data whose value the receiver already knows can be included in a message at a constant cost. Instead of full information (i.e. all information known by the sender), the messages can contain

1. all information known by the sender but not known by the receiver
2. a cryptographic hash of all information known by both the sender and the receiver.

This significantly reduces the cost of transmitting the redundant information. Naturally, like in full information protocols, all data has to be signed by the sender or by some other authority. Otherwise, the receiver could not rely on the authenticity of the redundant data in checking the consistency of the protocol run.

The reason why the redundant data items can be hashed to a constant length value is that the conflict-free hash functions can be assumed to preserve all redundancy of their arguments even when reducing the size of the data. Of course, the hashing does not really preserve all information but it is computationally infeasible to see the difference. If any two data items or hash values are combined by hashing, sufficient information from them is preserved for purposes of comparison with the original data. Thus, the full information principle can be implemented with only a constant increased length for each message of the protocol.

Furthermore, the cryptographic hash value on the redundant data does not need to be signed separately. The hash can be merged into the signed hash that would in any case exist on other data items in the message. Since most messages in cryptographic protocols include a signed or otherwise authenticated hash of the contents, appending the redundant data items in these hashes does not increase the length of the messages at all. The only cost is the computation required for digesting some additional data with the hash function. This is inexpensive since the standard hash functions are designed for bulk data. In fact, there is usually no variation in cost of hashing messages shorter than 512 bits.

Example 5 (three-way X.509) The reasoning behind the standard three-way X.509 protocol is somewhat complicated. The freshness of the first message is confirmed by the third message but there is no direct link between the messages. The connection between the first and third message is formed by the second message where the two nonces R_A and R_B appear together. (The obscure structure actually resulted in a weakness in the original version of he protocol. This was later corrected by adding the name of the recipient in the last message.)

1.	$A \longrightarrow B$	$R_A, B, E_B(K_{AB}),$
		$S_A(R_A, B, E_B(K_{AB}))$
2.	$B \longrightarrow A$	$R_B, A, R_A, E_A(K_{BA}),$
		$S_B(R_B, A, R_A, E_A(K_{BA}))$
3.	$A \longrightarrow B$	$R_B, B,\ S_A(R_B, B)$

It is, however, conceptually much simpler to include in all messages a signature on all information relevant to the protocol run. As we have described, it is not necessary to send all the redundant information but only a signed hash. Surprisingly, the messages of the protocol become shorter than in the standard protocol. (K_A^+ and K_B^+ are the public keys of A and B. The keys are often more precise principal identifiers than the mere names.)

1.	$A \longrightarrow B$	$R_A, E_B(K_{AB})$, S_A("Msg 1 of my protocol", $A, B, K_A^+, K_B^+, R_A, K_{AB}$)
2.	$B \longrightarrow A$	$R_B, R_A, E'_A(K_{BA})$, S_B("Msg 2 of my protocol", $A, B, K_A^+, K_B^+, R_A, R_B, K_{AB}, K_{BA}$)
3.	$A \longrightarrow B$	S_A("Msg 3 of my protocol", $A, B, K_A^+, K_B^+, R_A, R_B, K_{AB}, K_{BA}$)

In the above variant of the X.509 authentication, the link between the third and the first message is explicit. Since the cost of hashing is negligible and the messages do not grow in length, there is no reason to resort to the kind of implicit relations between messages that the standard protocol has. Understanding and security analysis of the modified protocol is much more straightforward. ■

When implemented with hashes in the way suggested above, the full information principle resembles closely the idea that all data should be tagged with a protocol run identifier. We do not select a random protocol identifier but rather use the entire contents of the protocol run as the identifier. The main purpose of a separate identifier would be to link together the messages of the protocol run, and this task is performed by the full information principle.

> **Strategy 2** Include in all authenticated messages a hash of all information that both the sender and the receiver should agree on at that stage of the protocol run.

Since the run identifier is the one piece of type information that we were not able to handle with unique cryptographic functions in Sec. 2, it makes sense to combine unique functions with hashed full information for complete typing of message data. This is exactly what we have done in Example 5.

4 Defining full information

In the previous section, we described a technique for implementing the full information principle efficiently. Having seen that the implementation is feasible, we will now take a closer look at the information that needs to be included in the messages and how it is used for security checks.

In a full information protocol, the messages always contain the sender's view of the current protocol run. This largely redundant data is used by the receiver for consistency checks with its own observations and previously received information. The data is also stored for future comparisons. All observations on the selected execution paths and data values by different principals should match. By checking this, the receiver can gain some assurance of the consistency of the entire protocol run. *Matching protocol runs* were first mentioned by Bird & al. [3] and the concept has been been formalized by Diffie, Oorschot and Wiener [6] who define records of protocol runs by two principals to be matching if all messages sent by one principal to the other are received by that principal in the same order. They then use matching as a requirement in their definition of a secure protocol run. The principals need to check that the runs match before accepting the protocol results (session keys etc.). For the checks, the principals need to receive redundant data from each other and the full information principle naturally provides them with maximal redundant information.

Although several authors have written about about "full information", it remains somewhat vague what is actually meant by that. The discussion in the previous section also avoided the precise definition of the term. There are at least two possible interpretations of what it means for a protocol principal to send all the information related to a protocol run to another principal. The full information can consist only of values of data items, or it can also include beliefs and trusts.

Thus, the simpler interpretation is that the variables in the protocol schema (principal names, nonces, timestamps, keys, hashes) are assigned values during the protocol run, and the full information of a principal comprises the values of the variables known to the principal. In addition, the data known to a principal can include the acknowledgement that certain messages of the protocol have been sent or received if this fact cannot be deduced from the values of the variables. In order to determine exactly which data items can be included, we need some understanding of which variables have been assigned values at the time when a message is sent.

In order to determine which events in the system are earlier than others, we define a reflexive and transitive order on the messages of a protocol. Intuitively, two messages compare if and only if one of them always can (possibly must) be sent before or at the same time as the other one. That is, in every protocol run, the smaller message either has to be sent earlier or it can be sent at the same time as the greater one. More formally, $M_1 \preceq M_2$ if one of the following holds:

1. The message M_1 is received and message M_2 is sent by the same principal P, and P does not have sufficient information for composing message M_2 before having access to M_1.

2. The message M_1 is received and message M_2 is sent by the same principal P, and the protocol rules sometimes (possibly always) require P to wait for M_1 and to perform some security-related comparisons before sending M_2.

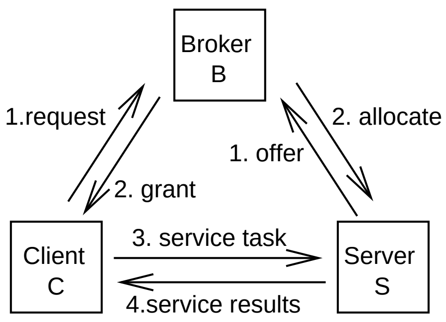

Figure 1. The service broker protocol

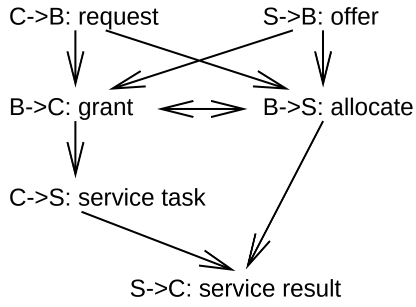

Figure 2. Inclusion order of full-information messages

3. The messages M_1 and M_2 are sent by the same principal P, and for all messages M for which $M \preceq M_1$, $M \neq M_1$ and $M \neq M_2$, also $M \preceq M_2$.

4. M_1 and M_2 are in the reflexive and transitive closure of the order given by Rules 1 through 3.

The intuitive idea of Rules 1 and 2 is that the message order obeys the flow of information. Although our order is defined on messages instead of events, these two rules correspond to the causal order of sending and receiving events in asynchronous communication. Rule 3 expresses the fact that if the prerequisites of message M_1 are a subset of the prerequisites of message M_2, then the principal always knows or is able to decide the contents of M_1 before sending M_2, and therefore, it is possible to send M_1 earlier or at the same time with M_2.

With the above defined order, we can find the maximal set of data items that are always known to a principal at the time of sending a message. For a message M, this *full information set* contains all the values of variables that have been assigned in messages smaller than or equal to M, $\{M' \mid M' \preceq M\}$. This is the maximal information that the principal sending a message can be required to always include in it.

Reiter and Gong [11] present a reminiscent idea of piggybacking on each message hashes of its causal predecessors. Since their goal is to prevent forgery of causal relationships, they concentrate on preserving the history of events while we also want to include information on the future messages when possible.

Example 6 (service broker) We illustrate the inclusion order of full-information messages with a simple system where the trusted party is a broker at a service exchange. The servers advertise their service to the broker and the clients send their requests to the same place. The broker then matches the supply to the demand, sends the server instructions to allocate resources to the client and informs the client of the allocated resources. Finally, the client contacts the server and obtains the service.

Fig. 1 shows a traditional diagram of the protocol where the messages with equal numbers can be transmitted in parallel. Fig. 2 shows the order of the messages (for clarity, without the reflexive and transitive arrows). From this order we can see, for example, that the message *grant* should contain not only all information from the messages *request* and *offer*, but also from the message *allocate*. The reason is that *grant* depends on the same earlier messages as *allocate* and, hence, its contents can be decided at the same time as the contents of *allocate*. The conclusion is natural, because a decision to grant a service always implies a decision to allocate it, and vice versa. ■

There are some points in out construction of the full information set that need further consideration. First, the principals are required to make decisions on future messages early. For example, if a principal sends two messages without receiving anything between them, it has to make up its mind on all the necessary variable assignments for the later message before sending the earlier one. This is mandatory because the above definition of full information requires inclusion of the data also in the earlier message, and it is always possible because the principal will not receive any new information before sending the later message.

Second, the principal sometimes has more information than will be included in the messages. If a data item is not always in the principal's possession by the time of sending the message, its inclusion is not required even when it is known. This kind of situation commonly occurs when the principal is handling two messages with independent contents concurrently so that the order of their sending can vary. The reason for omitting such data from the full information set is that there remain protocol runs whose security would not benefit from adding the data in the other message.

Third, the definition does not make any clear choice on which parts of the messages should be considered data items or variable assignments and included in the full information set, and which should be regarded as composite messages that need not be forwarded in the future messages. The problematic items are encrypted submessages whose contents the principal does not yet know, and values of cryptographic hashes or other functions whose arguments are unknown. It is beneficial to sign such submessages when they are originally included in a message. On the other hand, it is usually too costly to include them as redundant data in all future messages. We leave it to protocol designer to decide how far the full information principle should be extended.

It is also possible to choose a more comprehensive interpretation of the term full information. Namely, the principals do not only know the values of data items but they also have knowledge and trust that can sometimes be delivered to others. In a two-party protocol, or if all the principals trust each other, this coincides with the simpler interpretation of the full information, because in that case all authenticated information represents the common will of all the principals. In a multi-party protocol the principals, on the other hand, might not trust information passed through the hands of another principal even if they trust the original sender. In that case, one could include in the full information set not only data values but also an assurance of their authenticity in the form of the authenticated full information from previous messages. It seems, however, impossible in practice to require inclusion of all earlier authenticated messages, because the accumulating set of signed messages cannot be compressed by hashing them together like most of the other redundant data items. Nevertheless, it may at times be desirable from security viewpoint to pass along not only the data values but also the assurance of their authenticity even in situations where the assurance is redundant.

Finally, it is necessary to note that the full information principle cannot always be followed literally. In some protocols, added redundancy in messages would reveal confidential information that is to be kept secret from a protocol principal or from the outsiders. The information in danger includes secret data items that contain little entropy so that they can be recovered by comparing the hash with guessed values, e.g. 1-bit choices and passwords. Fortunately, sending hashed information instead of the full plaintext in most cases minimizes the danger of information leaks. Hashes of keys and other secrets with reasonable entropy can usually be published. Furthermore, if two or more independent secrets are hashed together, the attacker would have to guess them simultaneously in order to verify the correctness of the guess by comparison with the hash value.

5 Trust assumptions behind a session key

A key distribution protocol guarantees certain properties for the key and relies on certain assumptions. Not only the properties but also the assumptions affect the kinds of uses that the key can be put to. Replay attacks against session data can arise arise from confusion about the assumptions behind a key. The best known example is the X.509 key distribution protocol, where keys can be copied from one session to another. We point out in this section that if the key distribution process assumes trust between principals, the resulting key cannot fully protect communication on matters where the principals do not trust each other. Sec. 6 will then discuss the mechanisms by which one property of session keys, uniqueness, can efficiently be guaranteed without the mutual trust assumption.

The analysis of key distribution protocols usually ends when a session key has been delivered to the principals. The session key is believed to have some universal properties that make it good for protecting the consequent communication. The concept of *good session key* is is, however, dangerously vague in the literature. This can result in proponents of different key distribution schemes listing weaknesses of other protocols and labeling them "insecure" while they, in fact, are looking for different kinds of keys. We believe it to be more fruitful to accept that there are different types of session keys. After that, the properties of the keys, assumptions behind these properties, and the goodness of the keys for particular uses can be discussed. In the following, we do not attempt to cover all possible aspects of session keys, but rather look into the consequences of one assumption: complete trust between the principals.

The trust relations between principals of a cryptographic protocol can be arbitrarily complicated. Nevertheless, the two most common assumptions are full mutual trust and complete distrust. The former view is usually taken when the principals communicate over insecure channels and outside threats are their biggest concern. The latter situation occurs most often in commerce where a principal could gain advantage by sending false information to another. (The two extreme situations are pictured in Fig. 3 In practice, however, the there is usually the need to protect the communication against both internal and external threats. The danger here is that the cryptographic mechanisms for defending the system against external threats often rely on the assumption of trust between the principals and, therefore, can be ineffective against internal threats. In particular, if the goodness of a session key depends on the trustworthiness of a principal, the key should not be relied on in further increasing trust to that principal.

Example 7 (one-way X.509) Let us first look at the well known vulnerability of the ISO X.509 key distribution [1]. The one-way protocol is sufficient to illustrate our point.

$$\boxed{\begin{array}{lll} 1. & A \longrightarrow B & T_A, R_A, B, E_B(K_{AB}), \\ & & S_A(T_A, R_A, B, E_B(K_{AB})) \end{array}}$$

The problem is that the encrypted session key can be copied to another session where the responding party is also B. After having blindly copied the encrypted session key, the attacker could still not decrypt the messages, but he could

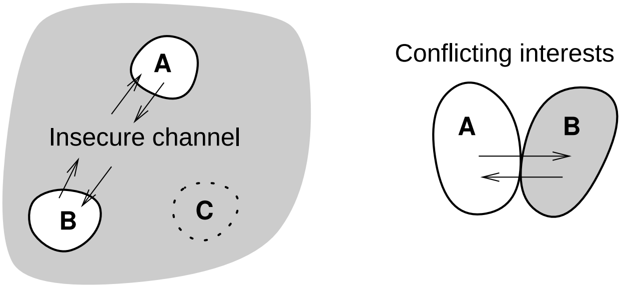

Figure 3. Settings with mutual trust and distrust

replay them between the two session. This leads to an oracle attack, where the attacker uses one session to obtain messages for the other session.

Abadi and Needham [1] present this attack as an example of the dangers of signing encrypted messages. The problem can also be seen from a more general viewpoint. It is that the key exchange protocol assumes complete trust between the principals in matters that will be protected with this key, but the key is mistakenly used to protect communication where the principals do not fully trusts each other.

For instance, if B mistakenly trusts others to show their possession of a secret piece of information by sending that information over a channel protected with an X.509 session key, an attacker C could copy the session key from A, replay the request from B to A, and pass the response from A to B. Sending the secret information in encrypted or authenticated form to B does not guarantee that it really came from C, if B cannot trust C and the session key hides the trust assumption.

The attack does not yield the X.509 protocol useless. It can be used by principals who really have full confidence in each other with respect to the things protected by the key. Such principals will not betray each other by copying keys from other sessions. For example, any two principals who want to conceal the contents of their mutual communication from the outside world and to ensure the integrity of their communication against external threats must trust each other not to reveal the contents and not to send manipulated messages. In that case, the principals trust each other with respect to the thing protected by the key, and the X.509 protocol suffices well for the key exchange. Furthermore, the protocol can be used by mutually suspicious principals to distribute a key for protecting a session against outside threats if other cryptographic mechanisms are used inside the session to counter mutual dishonesty and these mechanisms are independent of the session key. ■

Example 8 (Diffie-Hellman variant) It is important to note that the described vulnerability of the X.509 protocol is not simply a question of the order of encryption and signature although that is the technical detail causing the property. Consider a version of the Diffie-Hellman key exchange where the the public keys are stored in public directories. If an attacker copies another person's public key and sends it into the directory as his own key, a situation similar to the X.509 key-copying attack occurs. The attacker cannot recover the session key but he can replay messages between two sessions because they they both have the same session key. The attack can be countered by requiring the principals demonstrate the possession of the corresponding private D–H key before accepting a public key to the directory. This may, however, be inconvenient if a general-purpose directory if used and the public D–H keys are certified by the principals themselves.

■

The protocols in the above examples have two properties in common. Firstly, their advantage is that they make it possible for one principal to decide the session key in advance and do preparations such as encryption off-line before establishing the session. Secondly, the goodness of the session key depends entirely on the good will of the principal selecting the key. In the attacks, this principal decides to copy the key from another session.

This kind of vulnerability is usually blamed on technical details of the protocols but it can also be attributed to a poor understanding of the assumptions of trust between the principals. If the principals want the protocol to guarantee some quality of the key regardless of the trustworthiness of the principals, the protocol should utilize some specific mechanism for accomplishing this goal.

> **Strategy 3** Understand the trust assumptions made in the key-agreement protocol and limit the use of the session key accordingly, or change the protocol to eliminate the assumptions.

The specific property that is missing in the above protocols is that one often wants all session keys to be unique regardless of the will of individual principals. Protocols that always produce unique session keys are discussed in detail in the next section.

6 Producing unique session keys and binding the keys to their intended use

As we have seen in the previous section, duplicated session keys often lead to replay attacks, either to replays of entire deterministic sessions or to oracle attacks. Several techniques have been applied in protocols to produce unique keys for each session. We discuss some aspects of the most popular techniques and suggest a new, simpler one.

In the basic Diffie-Hellman key exchange, both principals can be assured that a different key is produced every time. This is because they both pick a new secret key and the session key depends on both of these secret keys. Also, any other protocol that allows both principals to choose their own key derivation parameters and combines these parameters to the session key will accomplish the same. The reason why the directory variant of Diffie-Hellman in Example 8 fails is that the keys are not new, but can be reused.

Although the above schemes cannot simultaneously guarantee unique keys and allow one principal to choose the key in advance, other protocols can. In the message below, the order of encryption and signature has been reversed, as Abadi and Needham [1] suggest.

$$E_B(\ldots, K_{AB},\ S_A(\ldots, K_{AB}))$$

The added redundancy inside the encryption now makes it impossible to cut and paste the plain session key to other sessions. A disadvantage is that the encrypted messages contains redundancy, making it thus easier to cryptanalyze. Mao and Boyd [10] go as far as stating that messages encrypted with the master key should never contain any redundancy. The master keys should be used only for encrypting random data strings, such as session keys, so that plaintext-only attack cannot recover the master key or the contents of the encrypted session keys from key distribution messages alone. The standard X.509 protocol conforms to this principle.

One way to bind the session key to a particular sender without adding redundancy inside the encryption is to add a redundant hash of the session key and the sender id into the signed message. This connects the names of the receiver

$$\ldots, E_B(K_{AB}),\ S_A(\ldots, A, K_{AB})$$

Surprisingly, [10] suggests this approach for the data origin check. The advantage is limited, however, because the hash value provides cryptanalysts with the same redundancy that is avoided inside the encryption.

Thus far, all the techniques we have seen for producing unique session keys have been somewhat unsatisfactory. They either contain redundant information of the session key, or require fresh parameters from both principals. Let us turn back to the basic problem: how to make sure that an agent cannot not have the same session key as other agents. The simplest solution seems to be to make the session key dependent on the identities of the agents. That is, use the X.509 protocol or any equivalent one, but instead of a a session key, send a key generation parameter.

$$\boxed{\begin{array}{lll} 1. & A \longrightarrow B & T_A, R_A, B, E_B(X), \\ & & S_A(T_A, R_A, B, E_B(X)) \end{array}}$$

Then compute the session key with a one-way function from the parameter and the identities of the protocol principals.

$$K_{AB} = HASH(X, A, B)$$

This method completely avoids sending of any redundant information that could be used in cryptanalyzing the session key. Also, it allows A to decide the session key in advance.

> **Strategy 4** To prevent copying of session keys and data between sessions with different sets of principals, distribute instead a key derivation parameter and generate the session key by hashing the names of the principals with the parameter.

The same strategy can actually be generalized to bind any information to the session key. When information on the intended use of the key is hashed into the key, it becomes more difficult to use key for other purposes.

Example 9 (SSH) The key exchange protocol in the SSH transport layer [14] produces a session identifier by hashing together the three first messages of the key exchange including all algorithm negotiation parameters (messages SSH_MSG_KEXINIT from the client and the server, and message SSH_MSG_KEXRSA_HOSTKEY). The encryption and integrity keys are computed with a one-way hash from the session identifier and a freshly exchanged shared secret. Thus, the key values depend on all the data items in the three first messages. This provides a strong guarantee of integrity for the algorithm negotiation process. ■

We rephrase Strategy 4:

> **Strategy 5** To bind the session key to its intended use, distribute instead a key derivation parameter and generate the session key by hashing with the parameter all information related to the key distribution process and the intended uses of the key.

7 Conclusion

In this paper, we have described several design principles for cryptographic protocols. By following the descried techniques, it is possible to build protocols that are robust against whole classes of replay attacks. Moreover, the systematically designed protocols are often more understandable and and easier to analyze than ones using highly specialized mechanisms.

Many design principles have been discussed earlier in the literature, but they are rarely presented in the form of concrete techniques that can immediately be applied in protocol design. Our goal has been to develop feasible techniques with which the principles can be efficiently implemented in real protocols. In particular, we are able to realize theoretical designs like full information protocols with small computational cost and minimal communications bandwidth. Also, we have generalized some of the mechanisms used in existing protocols.

The new techniques presented in this paper include efficient type tagging of messages with unique functions and implementation of the full information principle by hashing the redundant information. We also discuss the trust assumptions made in session key distribution and their relation to the properties of the keys. A simple technique is proposed for guaranteeing the uniqueness of session keys without making assumptions about trust between the protocol principals.

It is likely that many more design techniques will be developed and the ones presented here will be improved in the near future. In addition to building a toolbox for protocol engineers, it would be important to also analyze the correctness and completeness of such tools from a theoretical viewpoint. The extent of the security assurance given by different design principle could be assessed with the help of the logical models of cryptographic protocols. The ultimate goal of future research could be a complete set of design principles from which reliable, or even provable, protocols could be constructed systematically.

References

[1] M. Abadi and R. M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, Jan. 1996. Appeared first in Proc. 1994 IEEE CS Symposium on Research in Security and Privacy.

[2] R. Anderson and R. Needham. *Programming Satan's Computer*, volume 1000 of *LNCS*, pages 426–440. Springer Verlag, 1995.

[3] R. Bird, I. Gopal, A. Herzenberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Advances in Cryptography, Proceedings of CRYPTO'91*, volume 576 of *LNCS*, pages 44–61. Springer Verlag, 1991.

[4] U. Carlsen. Cryptographic protocol flaws. In *Proc. IEEE Computer Security Foundations Workshop VII*, pages 192–200. IEEE Computer Society Press, June 1994.

[5] *Recommendation X.509, The Directory - Authentication Framework*, volume VIII of *CCITT Blue Book*, pages 48–81. CCITT, 1988.

[6] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[7] ETSI TC-SMG. *GSM 03.20 (ETS 300 534): European digital cellular telecommunications system (Phase2); Security related network functions*, Sept. 1994.

[8] Recommendation x.509 (11/93) - the directory: Authentication framework. ITU, Nov. 1993.

[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer Verlag, Mar. 1996.

[10] W. Mao and C. A. Boyd. Development of authentication protocols: Some misconceptions and a new approach. In *Proc. IEEE Computer Security Foundations Workshop VII*, pages 178–186. IEEE Computer Society Press, June 1994.

[11] M. Reiter and L. Gong. Preventing denial and forgery of causal relationships in distributed systems. In *Proc. 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 30–40. IEEE Computer Society Press, May 1993.

[12] P. Syverson. A taxonomy of replay attacks. In *Proc. IEEE Computer Security Foundations Workshop VII*, pages 131–136. IEEE Computer Society Press, June 1994.

[13] T. Y. C. Woo and S. S. Lam. A lession on authentication protocol design. *Operating Systems Review*, 28(3):24–37, July 1994.

[14] T. Ylönen. SSH transport layer protocol. Technical report, IETF Network Working Group, Sept. 1996.