# Variations on the Themes of Message Freshness and Replay

— or the Difficulty in Devising Formal Methods to Analyze Cryptographic Protocols

Li Gong

SRI International
Computer Science Laboratory
333 Ravenswood Avenue
Menlo Park, California 94025 U.S.A.

## Abstract

*We survey the variations on the choice and the usage of freshness identifiers, and the various forms of replay attack. Besides helping to clarify the important concepts of freshness and replay, this exercise demonstrates the potential complexity in devising formal methods to analyze cryptographic protocols.*

## 1 Introduction

A common and important element in many algorithms for distributed systems is the desirable ability for a message recipient to reliably determine that the message is originated for a special occasion, e.g., the execution of a particular algorithm at a particular instant. This ability is all the more important in security-related algorithms such as authentication and authorization.

The association between a message and an occasion is usually through a temporal or causal relationship. With a temporal relationship, the occasion is usually distinguished by its start-time (and possibly its duration), and the message carries a timestamp to indicate its origination time. A nonce identifier – a number invented for a special occasion only [Needham 78] – is often used in a challenge-response scheme to prove that the response is associated with the challenge. In all cases, the originator of a message embeds a freshness identifier (a timestamp or a nonce) in the message to prove to the recipient that it is fresh and therefore not a replayed message.

After examining many real and toy protocols, it becomes clear that: (1) whether a certain value can be used as a suitable freshness identifier depends not only on its property such as being random or unpredictable but also on how it is to be used and who is to be trusted on what matters; (2) message replay can take place in various forms, most of which cannot be adequately defended by merely embedding freshness identifiers.

In the rest of the paper, we provide examples and discussions in support of the two propositions. Many arguments collected in this paper are not entirely new – they might have been said in existing literature or circulated in folklore in some form. Moreover, some of the security mechanisms have been suggested before, notably in [Voydock 83], though often in different contexts and for different purposes. We expect this note to be useful in clarifying the two important concepts in the design and verification of cryptographic protocols, and in illustrating the potential complexity and difficulty in devising logics or other tools to analyze cryptographic protocols.

Throughout the paper, we use notation $\{x\}_k$ (or $\{x\}_k^{-1}$) to denote $x$ encrypted (or decrypted) with key $k$, and $A \rightarrow B : x$ to denote that party $A$ sends message $x$ to $B$. Although our discussion applies to cryptographic protocols in general, we sometimes conveniently choose authentication protocols as examples.

## 2 Using Freshness Identifiers

The use of freshness identifiers involves three parties. A *supplier* chooses an identifier. A *prover* embeds it in some message. A *verifier* establishes the freshness of the message by examining the message composition, especially the use of the freshness identifier.

Given such a cast, we can now examine the most common types of freshness identifiers, how they are chosen and used, and how they differ in usage patterns. Note that a freshness identifier is also often called a nonce. All subsequently suggested freshness identifiers are nonces in that the supplier is not meant to repeatly use one identifier. However, it will become clear later that nonces come in different leagues.

**Timestamps.** To use timestamps, some form of loosely synchronized clocks must exist [Denning 81, Mills 90]. A typical usage is:

**Usage 0.**
$A \rightarrow B : \{Ta\}_k$

where $A$ embeds a timestamp $Ta$ in a message encrypted with $k$, which is shared between $A$ and $B$. When $B$ receives the message, $B$ decrypts it and compares $Ta$ with its current reading, say $Tb$, from the local clock. If $\mid Ta - Tb \mid$ is sufficiently small, then the message is regarded as fresh. In some situations, it is important to detect repetition of the same message while the message is still fresh (e.g., [Steiner 88]).

Here the supplier and the prover are literally one entity even if the system is supposed to supply the current timestamp. This is because the format of timestamp is public knowledge and its value corresponding to a moment in the future or in the past is also well-known, thus the prover could choose to use any timestamp. Consequently, the verifier must trust the prover to be honest in not using incorrect timestamps intentionally and also competent in not making a mistake unintentionally. In other words, the security of this usage depends on the prover using correct timestamps and the verifier having access to accurate clocks, even though timestamps are quite predictable and not random.

On the other hand, the supplier and the verifier are two distinct entities. This feature makes possible authenticated one-way communication (such as secure electronic messages) [Needham 78].

Finally, since maintaining the synchronization of clocks in a secure and practical way generally requires a secure authentication scheme, basing the latter on the availability of synchronized clocks is somewhat dubious. The proper solution is perhaps to consider the two requirements concurrently. It seems possible to solve this chicken and egg problem, at least when there are available hardware clocks which tick at a reasonably accurate rate [Reiter 93a].

There are other approaches to secure clock synchronization (e.g., [Simons 90, Lam 92]). We also think it possible to arrange nonce-based authentication as a backup mechanism (for authentication as well as for clock re-synchronization) whenever clocks are suspected to be out of synchrony with others. In such a hybrid scheme, we would need to add extra mechanisms in clock synchronization schemes so that a clock knows when it might be out of range. The status of a clock could always be communicated to the upper (e.g., application) layers if desired.

**Truly random numbers.** Usually random numbers are used in challenge-response systems. Here the supplier is generally the same entity as the verifier since no one else would know how and when a challenge is chosen. Consequently, a round of challenge and response requires at least two messages, and authenticated one-way communication is infeasible.

There are at least three typical ways to use random numbers as challenges. Suppose $Na$ is $A$'s challenge, $A$ and $B$ share key $k$, and $f()$ is a function such as increment.

**Usage 1.**
1. A $\rightarrow$ B: $Na$
2. B $\rightarrow$ A: $\{Na\}_k$

**Usage 2.**
1. A $\rightarrow$ B: $\{Na\}_k$
2. B $\rightarrow$ A: $Na$

**Usage 3.**
1. A $\rightarrow$ B: $\{Na\}_k$
2. B $\rightarrow$ A: $\{f(Na)\}_k$

It is crucial that $A$ chooses $Na$ carefully so that the same challenge is never reused for the same purpose or in the same fashion, and that no one (sometimes not even $A$ itself since these challenges are truly random numbers) can predict future values of $Na$.

Now $A$ does not have to trust $B$'s honesty and competence as in the case of using timestamps, but $A$ has to be careful not to make mistakes by itself.

In usage 3, the purpose of $f()$ is to ensure that messages 1 and 2 are not identical [Needham 78]. More care has to be taken to ensure that it is not easy for an intruder to manipulate message 1 to yield a valid message 2 [Boyd 90].

Interestingly, the use of random numbers as challenges also requires the availability of clocks that tick at some reasonably accurate (or at least fast) rate so that challenges could be timed out. This requirement is not explicitly stated in many protocol designs based on the challenge-response method.

**Counter values.** Using successive readings from a monotonic counter as challenges is a representative case of using predictable (to all parties) sequence numbers as challenges.

The supplier and the verifier are one entity and a counter value is used just as a random challenge. However, since a future counter value is predictable to an outsider, not all three usages listed above would be secure. In particular, in usage 1, an attacker could breach security by inflating $Na$ and withholding an answer from $B$ for future replay attack – a form of suppress-replay attack [Gong 92]. The attacker could also attempt to predict a plausible reply in usage 2. Only usage 3 is secure. Even in this case, since a future challenge is predictable to the prover, the verifier has to depend on the honesty and competence of the prover.

**Pseudo-random numbers.** If the supplier and the verifier keep the algorithm for choosing challenges secret so that the prover cannot predict future challenges (unlike the case with counters), then the verifier no longer relies upon the honesty and competence of the prover.

Consequently, all three usages are secure, given that the algorithm to produce pseudo-random sequence numbers are secure in the sense that the next number in sequence is unpredictable (to the prover) even if all previous numbers in the sequence are known. The protocol using event markers [Bauer 83] is a representative case.

Usually the supplier and the verifier are one entity so that the verifier can in theory predict its own future challenges. Thus $A$ itself must be competent in not unintentionally using future challenges; otherwise, suppress-replay attack again becomes possible. To completely avoid this weakness, truly random numbers as discussed earlier, possibly generated from some physical phenomenon, need to be used.

**Synchronized counters or pseudo-random number generators.** If the supplier and the verifier securely keep their respective counters in synchrony, the counters become logical clocks [Lamport 78], and the case is the same as using real time clocks. Such counters could be used for one-way authenticated communication, as in usage 0. The other three usages become unnecessary, though only usage 3 is secure.

If, instead of counters, the pseudo-random generators are kept in synchrony, these can also be used for one-way authenticated communication. However, they do not normally form logical clocks because the randomness makes comparison difficult, which is unlike counter values, which could be signed and verified by third parties as well [Reiter 93b]. Since a future challenge is unpredictable, all three usages are secure, although they are not really necessary because of the synchronization.

Nevertheless, since future challenges are predictable to those who know the algorithm for generating challenges (e.g., supplier, prover, and verifier), the verifier depends on the honesty and competence of the prover. Securely resynchronizing the counters or the number generators is also important.

**Fresh encryption keys.** A freshness identifier can be a fresh encryption key [Gong 90a], such as a session key chosen and distributed by an authentication server, or a key chosen and agreed upon by $A$ and $B$. In this case, the security of any exchange depends on the quality of the key (i.e., the honesty and competence of the supplier) and on the prover and verifier not leaking the key intentionally or by accident. Any message encrypted with this key is fresh. The security of usage 2 depends on the type of challenge used. The problem of reusing the same fresh message is discussed in the next section.

**Discussion.** We summarize the discussion so far in a table below. Note that usage 0 corresponds to one-way authenticated communication. It is obvious from this table that one can use synchronization to increase performance, but at the cost of having to depend on the security of the remote site as well as that of the synchronization scheme itself.

| | supplier = prover | supplier = verifier | trust prover | secure usage |
|---|---|---|---|---|
| timestamp (synchronized) | yes | no | yes | 0 |
| truly random | no | yes | no | 1,2,3 |
| counter | no | yes | yes | 3 |
| counter (synchronized) | yes | no | yes | 0,3 |
| pseudo-random | no | yes | no | 1,2,3 |
| pseudo-random (synchronized) | yes | no | yes | 0,1,2,3 |
| fresh key | yes/no | yes/no | yes | 0,1,3 |

The secure usages discussed above correspond to the intuitive ideas behind the freshness rules in BAN-like logics [Burrows 90, Gong 90b, Needham 92]. The principle underlying all freshness rules can be summarized as: if a message cannot be produced (in a computationally feasible way) without the knowledge of some text which is fresh, then the message is fresh. In this aspect, the BAN logic [Burrows 90] is more conservative than GNY [Gong 90b] in that only a small number of usages are expressed as rules. For example, GNY has rules that correspond to the following usage:

**Usage 4.**
1. A $\rightarrow$ B: $Na$
2. B $\rightarrow$ A: $\{Na\}_k^{-1}$

Such a usage is unusual, but feasible, as long as $Na$ is not chosen or constructed such that $\{Na\}_k^{-1}$ is easily predictable without the knowledge of $k$. Therefore, in this usage, one probably should not generate pseudo-random numbers ($Na$) by encrypting (even with a key unknown to other parties) a series of counter values.

GNY also covers the use of one-way hash functions. Suppose $h()$ is such a function.

**Usage 5.**
1. A $\rightarrow$ B: $Na$
2. B $\rightarrow$ A: $f(k, Na)$

**Usage 6.**
1. A $\rightarrow$ B: $Ta, f(k, Ta)$

The rationale is that given a well-chosen (e.g., fresh) challenge $Na$ or a timestamp $Ta$, a party who can construct a response $f(k, Na)$ or $f(k, Ta)$ must know $k$. Note that the use of a timestamp $Ta$ would enable authenticated one-way communication. GNY also has rules corresponding to similar usages based on public-key cryptosystems.

## 3 Message Replay

Replay attack is generally taken to mean an attack in which a past message is played back to the same recipient. This is somewhat misleading, however, since the most general definition of replay is perhaps just any *reuse* (possibly after manipulation) of (past) messages. All following variations of replay attack, many of which have been touched upon in the literature, satisfy this definition. We collect them here and discuss how to prevent and detect them.

**Simple replay.** Replays in this simplist form could take place without the need to modify past messages [Needham 78, Denning 81]. A proper defense is to embed freshness identifiers in messages.

**Repetition that can be logged.** For example, an attacker could replay a timestamped message within the valid time window. A proper defense is to keep those messages that have been received and still valid in (possibly non-volatile) storage, and check new messages against the stored ones [Steiner 88, Luan 90].

**Repetition that cannot be detected.** This situation could arise because the original message could be suppressed and thus did not arrive at its destination as expected. A solution is to embed freshness identifiers in messages but to make sure that messages are not postdated [Gong 92].

**Backward replay without modification.** This is replay back to the message sender rather than its originally intended recipient. This attack would be possible if one cannot recognize messages sent by oneself that are encrypted with a shared-key cryptosystem (such as DES). Another necessary condition for the attack is that two messages exchanged between the two parties have identical format [Boyd 90, Gong 90a, Bird 93].

One possible solution is to arrange messages so that they can be uniquely identified by their patterns, formats or certain fields. For example, every message could bear a protocol identifier, a version number, a message sequence number, and so on. Or a message could include a direction bit [Steiner 88, Boyd 90]. However, this becomes tricky when the recipient's address is only an alias and the real address would have required a different direction bit. Some of these methods may introduce a form of explicit redundancy or verifiable plaintext, which is undesirable in some situations [Lomas 89]. Perhaps the easiest solution is to design the protocols such that the messages one receives or sends out all have distinct formats.

Note that even if the two parties use two distinct keys (in a shared-key cryptosystem) to encrypt the traffic in different directions, one of them may make a mistake and use the key that it should not use. Standard usage of public key systems [Diffie 76, Rivest 78] could easily resolve this issue of sender verification.

**Replay to get a different interpretation** or **replay in another protocol**. This is a generalization the case above, for which solutions would require enforced formats or inclusion of redundancy, as noted in [Tsang 87, Molva 92].

**Replay after modification.** For example, one message could be modified (e.g., truncated) to appear as another valid message. Proper solutions include integrity checks [Boyd 90, Stubblebine 92].

## 4 Confidentiality

As noted earlier in section 2, to use fresh encryption keys as freshness identifiers, the confidentiality of such keys must be protected. On the other hand, lack of freshness in certain messages may result in breach of confidentiality. For example, if the original Needham-Schroeder protocol [Needham 78] is followed by exchange of confidential information encrypted with the session key, as illustrated below:

1. $A \rightarrow S$: $A, B, Na$
2. $S \rightarrow A$: $\{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
3. $A \rightarrow B$: $\{Kab, A\}_{Kbs}$
4. $B \rightarrow A$: $\{Nb\}_{Kab}$
5. $A \rightarrow B$: $\{Nb - 1\}_{Kab}$
6. $B \rightarrow A$: $\{\text{secret information}\}_{Kab}$

then a replayed message 3 could mislead $B$ into using a compromised session key ($Kab$), and the confidentiality of future messages (e.g., message 6) encrypted under this key is also compromised.

Let us consider one more example. Suppose that $A$ sends $B$ a message encrypted with a key which is securely shared between them. Merely from the fact that the message is properly encrypted, $B$ cannot assume that only $A$ and $B$ know the content of that message, since $A$ could have received the content earlier from another party $C$. $A$ cannot make the same assumption either because $B$ could later forward the content of the message to another party $D$ and $A$ is not guaranteed to be able to detect such a forwarding.

These examples imply that the analysis of confidentially in cryptographic protocols cannot be separated from other issues, such as freshness. In particular, encryption cannot be equated with confidentiality – one cannot establish the confidentiality of encrypted messages by merely examining the syntax of a protocol specification and searching for encryption operators.

Integrating the handling of confidentiality with that of all other issues in a logic of cryptographic protocols is a difficult task. It appears possible to ease this task by restricting the class of protocols in the case of using conventional cryptosystems, parallel to what Dolev and Yao [Dolev 83] achieved for the use of public-key cryptosystems.

To conclude the discussion, we note that due to the numerous ways protocols could be constructed, it is difficult to devise a formal method for analyzing cryptographic protocols that could deal with all important issues and cover all cases. Similar to program verification, we may need to find and agree upon a set of protocol primitives, which form a sufficient basis to construct all or most useful protocols, and for which we can devise sound inference rules that handle all important issues including confidentiality.

## Acknowledgment

Teresa Lunt and Peter Neumann provided helpful comments on an earlier version of this paper.

## References

[Bauer 83] R.K. Bauer, T.A. Berson, and R.J. Feiertag, "A Key Distribution Protocol Using Event Markers," *ACM Transactions on Computer Systems*, Vol.1, No.3, August, 1983, pp.249-255.

[Bird 93] B. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "Systematic Design of a Family of Attack-Resistant Authentication Protocols," to appear in *IEEE Journal on Selected Areas in Communications*, 1993.

[Boyd 90] C.A. Boyd, "Hidden Assumptions in Cryptographic Protocols," *Proceedings of IEE*, Part E, Vol.137, No.6, November, 1990, pp.433-436.

[Burrows 90] M. Burrows, M. Abadi, R.M. Needham, "A Logic for Authentication," *ACM Transactions on Computer Systems*, Vol.8, No.1, February, 1990, pp.18-36.

[Dolev 83] D. Dolev and A.C. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, Vol. IT-29, No.2, March, 1983, pp.198-208.

[Denning 81] D.E. Denning and G.M. Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, Vol.24, No.8, August, 1981, pp.533-536.

[Diffie 76] W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, Vol. IT-22, No.6, November, 1976, pp.644-654.

[Gong 90a] L. Gong, "Cryptographic Protocols for Distributed Systems," PhD dissertation, University of Cambridge, April, 1990.

[Gong 90b] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols," Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, California, May, 1990, pp.234-248.

[Gong 92] L. Gong, "A Security Risk of Depending on Synchronized Clocks," *ACM Operating Systems Review*, Vol.26, No.1, January, 1992, pp.49-53.

[Lam 92] K.Y. Lam and T. Beth, "Timely Authentication in Distributed Systems," Proceedings of European Symposium on Research in Computer Security, Toulouse, France, November, 1992. Published as *Lecture Notes in Computer Science*, Vol.648, Springer-Verlag, 1992, pp.293-303.

[Lamport 78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, Vol.21, No.7, July, 1978, pp.558-565.

[Lomas 89] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, "Reducing Risks from Poorly Chosen Keys," in Proceedings of the 12th ACM Symposium on Operating System Principles, Litchfield Park, Arizona, December, 1989. Published as *ACM Operating Systems Review*, Vol.23, No.5, pp.14-18.

[Luan 90] S.W. Luan and V.D Gligor, "On Replay Detection in Distributed Systems," in Proceedings of the 10th International Conference on Distributed Computing Systems, Paris, May, 1990, pp.188-195.

[Mills 90] D.L. Mills, "On the Accuracy and Stability of Clocks Synchronized by the Network Time Protocol in the Internet System," *ACM Computer Communication Review*, Vol.20, No.1, January, 1990, pp.65-75.

[Molva 92] R. Molva, G. Tsudik, E. van Herreweghen, and S. Zatti, "KryptoKnight Authentication and Key Distribution System," Proceedings of European Symposium on Research in Computer Security, Toulouse, France, November, 1992. Published as *Lecture Notes in Computer Science*, Vol.648, Springer-Verlag, 1992, pp.155-174.

[Needham 78] R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, Vol.21, No.12, December, 1978, pp.993-999.

[Needham 92] R.M. Needham, "Reasoning about Cryptographic Protocols," invited talk at the European Symposium on Research in Computer Security, Toulouse, France, November, 1992.

[Reiter 93a] M. Reiter, K. Birman, and R. van Renesse, "Fault-Tolerant Key Distribution," TR 93-1325, Department of Computer Science, Cornell University, Ithaca, New York, January, 1993.

[Reiter 93b] M. Reiter and L. Gong, "Preventing Denial and Forgery of Causal Relationships in Distributed Systems," to appear in Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy, Oakland, California, May 24-26, 1993.

[Rivest 78] 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.

[Steiner 88] J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," in Proceedings of the USENIX Winter Conference, February, 1988, pp.191-202.

[Stubblebine 92] S. Stubblebine and V.D. Gligor, "On Message Integrity in Cryptographic Protocols," in Proceedings of the 1992 IEEE Symposium on Security and Privacy, Oakland, California, May, 1992, pp.85-104.

[Simons 90] B. Simons, J.L. Welch, and N. Lynch, "An Overview of Clock Synchronization," in B. Simons and A. Spector (Eds.), *Fault-Tolerant Distributed Computing*, published as *Lecture Notes in Computer Science*, No.448, Springer-Verlag, 1990.

[Tsang 87] W.W. Tsang and J.W. John, "A Neglected Aspect of Digital Signatures," TR A5/87, Department of Information Systems and Computer Science, National University of Singapore, May, 1987.

[Voydock 83] V.L. Voydock and S.T. Kent, "Security Mechanisms in High-Level Network Protocols," *Computing Surveys*, Vol.15, No.2, June, 1983, pp.135-171.