A Family of Attacks upon Authentication Protocols

Gavin Lowe

Department of Mathematics and Computer Science,
University of Leicester

e-mail: `gavin.lowe@mcs.le.ac.uk`
URL: `http://www.mcs.le.ac.uk/~gl7/`

January 1997

Abstract

In this paper we present four similar attacks upon well known authentication protocols, and suggest that similar attacks exist for other protocols. Each of these attacks causes an agent B to think that another agent A is attempting to set up two (or more) simultaneous sessions with B, when in fact A is trying to establish only a single session. We describe how such an attack may have serious consequences.

1 Introduction

In a distributed computer system, it is necessary to have some mechanism whereby an agent can be assured of another's identity—the first agent should become sure that it really is talking to the other, rather than to an imposter impersonating the other agent. This is the role of an *authentication protocol*.

In this paper we present attacks upon a number of authentication protocols. All the protocols we consider have a similar objective: in each protocol, an *initiator* A seeks to establish a session with a *responder* B, possibly with the help of a trusted *server* S. When the protocol is complete, the responder should be assured of the initiator's identity (and in some cases vice versa).

All the attacks we consider are very similar in nature, and produce similar results. In each case, an intruder is able to replay a message so as to trick the responder into thinking that the initiator is in fact trying to establish *two* (or more) simultaneous sessions. We term this error—where one agent is mistaken about the multiplicity of sessions—a *multiplicity error*, and we term the corresponding attack a *multiplicity attack*.

We are making two assumptions here.

- We are assuming that the responder is indeed able to carry out two simultaneous sessions; this seems a reasonable assumption given modern multi-tasking computers.

- We are also assuming that the responder is not able to recognize the fact that a message is repeated; again this seems reasonable: recognizing repeated messages would require considerable inter-thread communication, which may be very costly, especially if the responder is a distributed process.

Another way of considering the above two assumptions is that they identify conditions under which the protocol can not be securely operated: at the very least, any protocol should come with a warning as to what must be true about the environment in which it is implemented.

There are situations in which the attacks outlined above lead to more serious security breaches. In each case, the intruder I is able to overhear messages sent from A to B in the genuine session (although he may not be able to decrypt them), and then replay them in the second session. Depending on the circumstances, this could be relatively harmless, or extremely damaging; one particularly damaging example would be where B is a bank, and the repeated message from A is of the form "Transfer £1000 from my account to I's".

I would claim that the fact that the intruder is able to cause a multiplicity error represents a failure of authentication; however, others may disagree. Part of the problem is that different people seem to mean different things by the word "authentication"; this highlights the importance of formalizing precisely what we mean by "authentication". To this end, in [Low96c] I identified eight different meanings of the word, and formalized each using CSP [Hoa85]. In particular, two of my authentication specifications are designed to differentiate between protocols for which multiplicity attacks are or are not possible. My philosophy is that the designer of any authentication protocol should be clear about precisely what it achieves; thus a user can decide whether or not the protocol is suitable for a particular situation.

Before discussing the protocols themselves, and the attacks upon them, we explain some of the ingredients of the protocols.

The protocols all use either *shared key* or *public key cryptography*, to encrypt messages for *secrecy* or for *sender authentication* (to assure the receiver of the identity of the sender).

In shared key cryptography, each pair of agents, A and B, share a key, which we denote K_{ab}; this is used to encrypt all messages between these agents. The same shared key can then be used to decrypt the message. This ensures both secrecy and sender authentication. We will write $\{m\}_k$ for message m encrypted with key k. In some of the protocols we discuss, A and B initially share keys K_{as} and K_{bs}, respectively, with a server S; the protocol aims to establish a key K_{ab} shared between A and B.

In public key cryptography, each agent A possesses a *public key*, denoted K_a, which any other agent can obtain from a key server. It also possesses a *secret key*, K_a^{-1}, which is the inverse of K_a. Any agent can encrypt a message m using A's public key to produce $\{m\}_{K_a}$; only A can decrypt this message, so this ensures secrecy. A can sign a message m by encrypting it with its secret key, to produce $\{m\}_{K_a^{-1}}$; any other agent in possession of A's public key can then decrypt this message; the encryption using A's secret key achieves sender authentication: it assures other agents that the message really did originate from A.

The protocols make use of *nonces*: random numbers generated with the purpose of being used in a *single* run of the protocol. We denote nonces by N_a and N_b: the subscripts are intended to denote that the nonces were generated by A and B, respectively.

Messages in the protocols also include *timestamps*, denoted T_a and T_b, which represent the time at which the message was generated. Again the subscripts denote the identities of the agents generating the timestamps. When an agent receives a message containing a timestamp, it should check that the timestamp is close to the current time, making some allowance for network delay and differences between local clocks. We assume that the agents have access to a secure, reliable timeserver, so that the differences between the values of local clocks is small (Gong [Gon92] describes how attacks can occur if the intruder is able to fool an agent about the correct time).

We assume that the intruder I is a user of the computer network, and so is able to set up standard sessions with other agents, and other agents may try to set up sessions with I. We assume that the intruder can overhear and intercept any messages in the system, and introduce new messages using information he has learnt.

2 The Wide Mouthed Frog Protocol

The Wide Mouthed Frog Protocol aims to establish a shared key K_{ab} for use in a session between A and B. The key is generated by A, and sent to B via a trusted server S. The protocol can be described by:

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

Each message contains a timestamp, which is intended to guarantee the recentness of the message.

Anderson and Needham [AN95] have described an attack upon this protocol, which involves the intruder replaying messages at the server, so that the timestamp is continually updated. This attack makes use of the fact that the two messages have the same form, so is easily prevented by removing this similarity.

We now describe a different, independent attack, whereby an intruder makes B believe that A has established two sessions with it, when A wanted only a single session. The attack involves two interleaved runs of the protocol, which we call α and β; we denote, for example, message 2 of run α by $\alpha.2$.

The attack proceeds as follows:

$$\begin{array}{lrcl} \text{Message } \alpha.1. & A \to S & : & A, \{T_a, B, K_{ab}\}_{K_{as}} \\ \text{Message } \alpha.2. & S \to B & : & \{T_s, A, K_{ab}\}_{K_{bs}} \\ \text{Message } \beta.2. & I(S) \to B & : & \{T_s, A, K_{ab}\}_{K_{bs}}\,. \end{array}$$

Here and henceforth, we write $I(S)$ to represent the intruder I imitating S, etc. In run α, the protocol proceeds in the normal way, and A establishes a session with B using the key K_{ab}. Then, in message $\beta.2$, the intruder impersonates S, and replays message $\alpha.2$; B then believes that A is trying to establish a second session with it.

Alternatively, the intruder could replay message $\alpha.1$ at the server, as message 1 of a new run; this would lead to B receiving a second message 2, with a similar effect as above.

The reason this attack succeeds is that the timestamps provide only partial authentication of A: they show that A is currently trying to establish *a* session, but they don't show how many. This attack can be avoided by adding a nonce handshake to the end of the protocol:

$$\begin{array}{llcl} \text{Message 1.} & A \to S & : & A, \{T_a, B, K_{ab}\}_{K_{as}} \\ \text{Message 2.} & S \to B & : & \{T_s, A, K_{ab}\}_{K_{bs}} \\ \text{Message 3.} & B \to A & : & \{N_b\}_{K_{ab}} \\ \text{Message 4.} & A \to B & : & \{N_b + 1\}_{K_{ab}}\,. \end{array}$$

This change will prevent the above attack from working: B will generate two different nonces, one for each run of the protocol, and expect back two corresponding message 4s; however, A will return only one such message, and the intruder is unable to generate the other. Note that the timestamps are still necessary to prevent the replay of old, possibly compromised, keys.

Unfortunately, this fix destroys one of the Wide Mouthed Frog's most attractive features, namely its conciseness.

3 The Denning-Sacco shared key protocol

In [DS81], Denning and Sacco suggested the following protocol, which is based on the Needham-Schroeder Shared Key Protocol:

$$\begin{array}{llcl} \text{Message 1.} & A \to S & : & A, B \\ \text{Message 2.} & S \to A & : & \{B, K_{ab}, T_s, \{K_{ab}, A, T_s\}_{K_{bs}}\}_{K_{as}} \\ \text{Message 3.} & A \to B & : & \{K_{ab}, A, T_s\}_{K_{bs}}\,. \end{array}$$

The protocol aims to establish a shared key K_{ab}, and to authenticate A to B.

However, this protocol is prone to a multiplicity attack:

$$\begin{array}{lrcl}
\text{Message } \alpha.1. & A \to S & : & A, B \\
\text{Message } \alpha.2. & S \to A & : & \{B, K_{ab}, T_s, \{K_{ab}, A, T_s\}_{K_{bs}}\}_{K_{as}} \\
\text{Message } \alpha.3. & A \to B & : & \{K_{ab}, A, T_s\}_{K_{bs}} \\
\text{Message } \beta.3. & I(A) \to B & : & \{K_{ab}, A, T_s\}_{K_{bs}}\ .
\end{array}$$

In run α, the protocol proceeds as normal. Then the intruder replays message $\alpha.3$ in message $\beta.3$, and the responder B is fooled into thinking that A is trying to set up two sessions with him.

This attack is prevented by adding a nonce handshake:

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

4 The CCITT X.509 Protocol

Our next example comes from [CCI87, BAN89]. Whereas our previous protocols have been involved with establishing authentication *before* the transfer of data, this protocol includes the transfer of user data X_a and Y_a from A to B, and data X_b and Y_b from B to A; the protocol is supposed to provide user authentication for X_a and X_b, and ensure the secrecy of Y_a and Y_b. The protocol is based around the use of public and secret keys. It may be described as follows:

$$\begin{array}{lrcl}
\text{Message 1.} & A \to B & : & A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}} \\
\text{Message 2.} & B \to A & : & B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}} \\
\text{Message 3.} & A \to B & : & A, \{N_b\}_{K_a^{-1}}\ .
\end{array}$$

The attack is somewhat complicated, involving three simultaneous runs of the protocol.

$$\begin{array}{lrl}
\text{Message } \alpha.1. & A \rightarrow B & : A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}} \\
\text{Message } \alpha.2. & B \rightarrow A & : B, \{T_b, N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}} \\
\text{Message } \alpha.3. & A \rightarrow B & : A, \{N_b\}_{K_a^{-1}} \\
\text{Message } \beta.1. & I(A) \rightarrow B & : A, \{T_a, N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}} \\
\text{Message } \beta.2. & B \rightarrow I(A) & : B, \{T_b', N_b', A, N_a, X_b', \{Y_b'\}_{K_a}\}_{K_b^{-1}} \\
\text{Message } \gamma.1. & A \rightarrow I & : A, \{T_a', N_a', I, X_a', \{Y_a'\}_{K_i}\}_{K_a^{-1}} \\
\text{Message } \gamma.2. & I \rightarrow A & : I, \{T_i, N_b', A, N_a', X_i, \{Y_i\}_{K_a}\}_{K_i^{-1}} \\
\text{Message } \gamma.3. & A \rightarrow I & : A, \{N_b'\}_{K_a^{-1}} \\
\text{Message } \beta.3. & I(A) \rightarrow B & : A, \{N_b'\}_{K_a^{-1}} .
\end{array}$$

In run α, A performs a normal run of the protocol with B. In run β, the intruder impersonates A to persuade B that A is trying to establish a second session. In message $\beta.1$, the intruder replays message $\alpha.1$. Then in message $\beta.2$, B returns a new message, issuing a nonce challenge N_b'. At this point, I is unable to answer the nonce challenge, and so causes A to initiate a run of the protocol (run γ) with I, and uses A as an oracle. In message $\gamma.2$, the intruder returns a message, including the nonce N_b' just received from B. A then returns this nonce in message $\gamma.3$, encrypted with its own private key. This message is precisely what the intruder needs to answer B's nonce challenge, in message $\beta.3$. Thus, in run γ, the intruder uses A as an oracle in order to obtain the message $\{N_b'\}_{K_a^{-1}}$, which it uses to answer B's nonce challenge in run β.

In the original description of the protocol [CCI87], it is stated that B does not need to check the timestamp T_a. In [BAN89], this is shown to be incorrect: the authors describe an attack where the intruder uses A as an oracle to answer a nonce challenge from B, as in the above attack. We have now shown that even if B does check the timestamp, the protocol is still flawed. In [BAN89], it is suggested that if B's identity is included in message 3, then this prevents an intruder from using A as an oracle. This will also prevent the above attack. This also makes the timestamp T_a redundant: the nonce challenge assures B of the presence of A. Further, we do not need both the timestamp T_b and the nonce N_a, so we remove the former. This gives us the following protocol:

$$\begin{array}{lcl}
\text{Message 1.} & A \rightarrow B & : A, \{N_a, B, X_a, \{Y_a\}_{K_b}\}_{K_a^{-1}} \\
\text{Message 2.} & B \rightarrow A & : B, \{N_b, A, N_a, X_b, \{Y_b\}_{K_a}\}_{K_b^{-1}} \\
\text{Message 3.} & A \rightarrow B & : A, \{B, N_b\}_{K_a^{-1}} .
\end{array}$$

5 The SPLICE/AS Protocol

In [YOM90], the following protocol was suggested, which aims to provide each of A and B with the other's public key, and to establish mutual authentication:

$$\begin{array}{lrcl}
\text{Message 1.} & A \rightarrow S & : & A, B, N_a \\
\text{Message 2.} & S \rightarrow A & : & S, \{S, A, N_a, K_b\}_{K_s^{-1}} \\
\text{Message 3.} & A \rightarrow B & : & A, B, \{A, T_a, L, \{N_a'\}_{K_b}\}_{K_a^{-1}} \\
\text{Message 4.} & B \rightarrow S & : & B, A, N_b \\
\text{Message 5.} & S \rightarrow B & : & S, \{S, B, N_b, K_a\}_{K_s^{-1}} \\
\text{Message 6.} & B \rightarrow A & : & B, A, \{B, N_a' + 1\}_{K_a} .
\end{array}$$

Attacks on this protocol were presented in [HC95] and [CJ95]. In [CJ95], Clark and Jacob suggested changing the protocol as follows:

$$\begin{array}{lrcl}
\text{Message 1.} & A \rightarrow S & : & A, B, N_a \\
\text{Message 2.} & S \rightarrow A & : & S, \{S, A, N_a, B, K_b\}_{K_s^{-1}} \\
\text{Message 3.} & A \rightarrow B & : & A, B, \{T_a, L, \{A, N_a'\}_{K_b}\}_{K_a^{-1}} \\
\text{Message 4.} & B \rightarrow S & : & B, A, N_b \\
\text{Message 5.} & S \rightarrow B & : & S, \{S, B, N_b, A, K_a\}_{K_s^{-1}} \\
\text{Message 6.} & B \rightarrow A & : & B, A, \{N_a' + 1\}_{K_a} .
\end{array}$$

However, this protocol is still subject to a multiplicity attack:

$$\begin{array}{lccl}
\text{Message } \alpha.1. & A \rightarrow S & : & A, B, N_a \\
\text{Message } \alpha.2. & S \rightarrow A & : & S, \{S, A, N_a, B, K_b\}_{K_s^{-1}} \\
\text{Message } \alpha.3. & A \rightarrow B & : & A, B, \{T_a, L, \{A, N_a'\}_{K_b}\}_{K_a^{-1}} \\
\text{Message } \alpha.4. & B \rightarrow S & : & B, A, N_b \\
\text{Message } \alpha.5. & S \rightarrow B & : & S, \{S, B, N_b, A, K_a\}_{K_s^{-1}} \\
\text{Message } \alpha.6. & B \rightarrow A & : & B, A, \{N_a' + 1\}_{K_a} \\
\text{Message } \beta.3. & I(A) \rightarrow B & : & A, B, \{T_a, L, \{A, N_a'\}_{K_b}\}_{K_a^{-1}} \\
\text{Message } \beta.4. & B \rightarrow S & : & B, A, N_b' \\
\text{Message } \beta.5. & S \rightarrow B & : & S, \{S, B, N_b', A, K_a\}_{K_s^{-1}} \\
\text{Message } \beta.6. & B \rightarrow I(A) & : & B, A, \{N_a' + 1\}_{K_a} .
\end{array}$$

The intruder simply replays message $\alpha.3$ to establish a second session.

Again, this attack can be prevented by adding an extra nonce challenge.

6 Conclusions

In this paper we have presented similar attacks upon four different protocols. In each attack, an intruder I can trick an agent B into thinking that another agent A is trying to establish two sessions with him, when in fact A is only trying to establish a single session. The attacks also allow the intruder to replay in the second (fake) session messages sent from A to B in the first session: this may have serious consequences.

In each attack, the underlying flaw was that the only assurance that B received about the presence of A was a timestamp: this assures B that A is currently present and participating in the protocol, but does not tell B anything about how many runs of the protocol A is involved in. Our findings suggest that timestamps should not be used as the sole means of authenticating agents if the number of resulting sessions could be considered important.

These multiplicity attacks apply to more protocols than we have given here; for example, Bellovin and Merritt [BM90] describe a similar attack on the Kerberos protocol [MNSS87]. We suspect that the attacks can be applied to yet more protocols.

It is often claimed that nonces and timestamps can both be used to provide *freshness* (for example, see [BAN89]): that is, they provide a guarantee that the message in question has not appeared in any message before the current run of the protocol. The attacks in this paper show that this is not true: timestamps merely prove that a particular message was created *recently*.

Given the number of protocols that have been suggested, only subsequently to be found to be subject to attacks, it is clearly important to develop better methods for analysing protocols. One recently developed technique is to encode the protocol and an intruder in CSP [Hoa85], and then use the FDR refinement tool [Ros94] to verify that no successful attacks can occur (indeed, some of the above attacks were discovered using this technique); see [Ros95, Low96a, LR96, Low96b] for more details.

References

[AN95] Ross Anderson and Roger Needham. Programming Satan's computer. In J. van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[BM90] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, 1990.

[CCI87] CCITT draft recommendation X.509. The directory-authentication framework, version 7, 1987. Gloucester.

[CJ95] John Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151–155, 1995.

[DS81] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[Gon92] Li Gong. A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1):49–53, 1992.

[HC95] Tzonelih Hwang and Yung-Hsiang Chen. On the security of SPLICE/AS—the authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[Low96a] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[Low96b] Gavin Lowe. Casper: A compiler for the analysis of security protocols, 1996. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~gl7/Security/Casper/index.html`.

[Low96c] Gavin Lowe. A hierarchy of authentication specifications. Technical Report 1996/33, Department of Mathematics and Computer Science, University of Leicester, 1996.

[LR96] Gavin Lowe and A. W. Roscoe. Using CSP to detect errors in the TMN protocol. In preparation, 1996.

[MNSS87] S. P. Miller, C. Neumann, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, MIT, 1987. Available from URL `ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS`.

[Ros94] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, 1995.

[YOM90] S. Yamaguchi, K. Okayama, and H. Miyahara. Design and implementation of an authentication system in WIDE Internet environment. In *Proc. 10th IEEE Region Conf. on Computer and Communication Systems*, 1990.