Timestamps and authentication protocols

Chris J. Mitchell

Technical Report
RHUL–MA–2005–3
25 February 2005

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

Timestamp-based authentication and key establishment protocols have received a number of criticisms, despite their potential efficiency advantages. The purpose of this paper is to propose a novel timestamp management method which has the potential to increase the scope of applicability of such protocols. Since timestamp-based protocols typically involve one less message than challenge-response protocols, the potential efficiency gains are considerable.

1 Introduction

As has been widely discussed in the literature, see for example chapter 10 of [4], (entity) authentication protocols typically employ one of three types of time variant parameters: timestamps, sequence numbers and nonces. Protocols based on timestamps or sequence numbers typically require one less message than protocols based on nonces, and protocols based on the timestamps have the further advantage of being stateless[1], whereas the other two types of protocol are not.

However, a number of problems with timestamp-based protocols have been widely reported — see, for example, [2, 4]. One commonly reported problem is the difficulty in ensuring that the clocks held by the communicating parties are securely synchronised. In particular it is normally assumed that such protocols require there to be a reliable source of time within the network, and for all parties to conduct regular secure resynchronisation sessions with this reliable time source to ensure their clocks are correct.

In this paper we reconsider the issue of clock synchronisation and suggest that the problems with such protocols may not be so serious after all. In particular, the need for routine resynchronisation with a reliable time source can be avoided. Note that this is potentially significant because, if timestamp-based protocols become practical in a wider range of environments, considerable efficiency savings may be realisable.

2 Timestamps reconsidered

We start by making two key observations about timestamp-based authentication protocols. These observations are critical to our subsequent analysis.

[1] Of course, the maintenance of a clock value could be regarded as state information, but it is typically provided by separate hardware and hence is not stored state in the normal sense of the term

- Although it is necessary for the clocks held by a group of communicating devices to be synchronised, it is not necessary for the clocks to be correct. It is simply necessary that all the devices agree on the value of the current time *for the purposes of authentication*.

- If the clock values used for authentication purposes are only ever adjusted forwards, then replays of messages are prevented.

Note that the second observation is based on the assumption that the 'normal' safeguards are in place for the use of timestamp-based authentication protocols. That is, we are assuming that the clocks used by the communicating devices are not precisely synchronised, but that each device can safely assume that the differences in clock values are less than some threshold value T. Moreover, there is also an assumption that messages sent from one device to another are subject to a maximum transit delay of D. Then the recipient of a message, with current clock value of t_c, will accept the message as 'fresh' as long as the timestamp in the message (t_m say) satisfies $t_c - T - D \leq t_m \leq t_c + T$. This time acceptance interval, i.e. $[t_c - T - D, t_c + T]$ is sometimes referred to as the 'window of acceptance'. Then, in order to detect message replays occurring within this window (and as in [1, 3, 4]), the message recipient must retain copies of all messages received and accepted within this window, and compare them with all newly received messages so as to detect replays. Note that it is only necessary to store a hash of each received message, and compare it with the hash of the newly received message, where the hashes are computed using a collision-resistant cryptographic hash-function (see, for example, [4]).

We can now describe how these observations can be used to implement timestamp-based authentication protocols in a simple way, even when there is no reliable source of time within the network.

3 A new implementation scenario

We start by making some assumptions about the operation of each device within a group of communicating devices.

- We suppose that each device within a group of communicating devices possesses a clock that is reasonably reliable (e.g. accurate to within a few seconds a day). This is not an unreasonable assumption in many scenarios; for example, many mobile network devices, including many if not most mobile phones, and almost all PCs and electronic organisers (personal digital assistants), have such a clock.

- We next suppose that a timestamp-based authentication and/or key establishment protocol is used between pairs of devices within this network. It is not important to the discussion here exactly how this protocol operates — indeed it could be based on shared secrets, e.g. using an online trusted third party, or on public/private key pairs and an associated Public Key Infrastructure. We assume only that the protocol uses timestamps to guarantee the freshness of messages, and that the protocol is designed in such a way that the recipient of a timestamped protocol message can guarantee its origin and integrity by cryptographic means.

- We further suppose that each device maintains a 'clock offset' value, used purely for the purposes of entity authentication and/or key establishment. Specifically, the time value used for security protocol purposes is always the sum of the clock value and the clock offset value. If the clock value is ever adjusted, then the clock offset value must also be adjusted to ensure that the sum of the clock value and the clock offset is never decreased. This would most easily be achieved by ensuring that if the clock value is moved back by δ seconds, then the offset is simultaneously increased by δ seconds. To avoid unnecessary increases in the sum of the clock value and the offset, we also suppose that the offset is reduced by δ seconds whenever the clock is moved forward by δ seconds. Since, as we describe below, the clock offset value is never reduced (except as described immediately above), the sum of the clock value and the clock offset never decreases (cf. Section 2).

We now consider how a device uses timestamps within the authentication (or key establishment) protocol. We separate our discussion into two parts: receipt of messages, where the timestamp is used to decide whether or not to accept a received message, and transmission of messages, where a timestamp is generated and inserted into the message.

When a device receives a protocol message containing a (protected) timestamp, this timestamp is compared with the current clock value for the device in the following way. Suppose the timestamp in the message is t_m, the current clock value is t_c, the current clock offset value (stored within the device) is t_o, and T and D are as above. Then the message is accepted as fresh as long as the following inequality holds:

$$t_c + t_o - T - D \leq t_m.$$

Moreover, if $t_m > t_c + t_o$, then $t_m - t_c - t_o$ is added to the clock offset value.

When a device sends a protocol message containing a timestamp, this timestamp is set equal to $t_c + t_o$.

4 Analysis and failure handling

We now consider how this timestamp management process provides the required freshness checking for protocol messages. First note that, as in Section 2, we suppose that every device maintains a list of all messages received with a timestamp t_m satisfying $t_c+t_o-T-D \leq t_m$. As soon as the timestamp t_m in a stored message no longer satisfies this inequality it can be discarded. Every received message is compared with the current list of stored messages, and discarded if it matches.

Since the value t_c+t_o never decreases, and given that T and D are fixed, it should be clear that once a message has been successfully received by a device, any replay of this message will be rejected. This is the primary objective of freshness checking. This has been achieved without any assumptions about routine clock resynchronisation.

Of course, there is a danger that, if device A's clock+offset value is significantly in advance of the clock+offset value of device B, all messages sent from B to A will be rejected. This situation could arise if there is no routine clock resynchronisation and messages are sent between devices relatively infrequently. There are a number of ways in which this could be dealt with. We mention two possible approaches.

1. The first solution would be for the recipient of any message which is rejected because of an 'old' timestamp to respond to the message sender with a cryptographically protected timestamp equal to $t_c + t_o$. (This of course assumes that the cryptographic keys necessary for such a transfer are in place). When the sender of the failed message receives this message, it will increase its clock offset appropriately, as in Section 3.

2. The second solution would involve the message recipient informing a third party about the problem, e.g. a network management entity, who would then send an appropriate message to the originating device, causing it to update its clock offset.

5 Hazards and countermeasures

We now consider how the scheme described above might go wrong.

- **Clock failures**. The first and perhaps most obvious problem we consider is where the clock within one device in the network fails catastrophically. By catastrophically we mean that the sum of the clock value and the clock offset is set equal to a very large value — somewhere close to the maximum possible. Every time this failed device

sends a message, the recipient will be obliged to reset its clock offset to the largest possible value, and this situation may rapidly disseminate across the network. Once the maximum clock value is reached, the whole network will potentially fail.

There are a number of possible means that can be used to avoid such a catastrophic situation. Firstly, we can impose restrictions on the size of any increase in the clock offset value. If the difference is too large to be credible, an error message can be generated and sent to either the message originator or a network management entity (or both).

Secondly, we can modify the system so that clock offset values are not routinely updated as described in Section 3. If the recipient of a message (A say) finds that $t_m > t_c + t_o$, then there are two possible cases. If $t_m - t_c - t_o$ is less than some defined threshold then no action is taken by A (i.e. if the clock on the sending device is just slightly ahead of the clock of A, then no adjustments are made). However, if $t_m - t_c - t_o$ is greater than the threshold value then A sends a request to a trusted third party (TTP) for a time resynchronisation. This TTP will respond with a cryptographically protected timestamp, which can be used to update the clock offset of A.

In such a situation, if some of the network devices have clock values in advance of this TTP, then these devices will not be able to receive messages from other entities in the network. This problem could be avoided by requiring such devices to conduct a nonce-based resynchronisation with the TTP. In such a circumstance (and only in this situation) these devices would be required to change their clock offset so that the sum $t_c + t_o$ is reduced.

- **Malicious clock manipulation**. The clock failure scenario is essentially an accidental 'Denial of Service' (DoS) attack. There are also variants of this which may arise through deliberate manipulation of a genuine member of a network. That is, a malicious entity could deliberately cause clock offsets to be systematically increased to ultimately give rise to a DoS. If clock offset adjustments are size limited, then the malicious attacker could bypass this protection by arranging for a large number of smaller clock increments. However, the second countermeasure to a failed clock prevents this attack, at the cost of increased network traffic. (Note that the messages to the TTP could also enable such attacks to be detected).

- **Postdated message ('suppress replay') attack**. We now consider a problem first described by Gong [2]. Essentially, suppose the clock

value added to the clock offset for device A is substantially ahead of the corresponding value for device B. Then a message sent from A to B can be forcibly delayed by a considerable period of time without detection by B. Indeed, by the time the message arrives at B, A may no longer be 'live', which, if the protocol is providing entity authentication, can be regarded as a protocol failure.

Dealing with such a situation is very difficult indeed. However, such an attack will probably only be a problem in certain circumstances. Indeed, if the protocol is being used for key establishment at the start of a secure session, then this attack is most unlikely to be an issue. The conclusion is that a timestamp-based protocol should only be used with care, but the same is true of almost every security solution.

6 Concluding remarks

There are some similarities between the timestamp management proposal described here to a previous proposal to use timestamps to generate sequence numbers (see [5]). However, this latter scheme was designed for use in a particular type of network, namely where a single 'server' entity communicates with a large number of 'client' entities, and clients do not intercommunicate. The solution described here is not specific to such a network architecture.

The solution described enables timestamp based protocols to be used in the absence of routine clock resynchronisations. There are possible problems with DoS attacks, and also with suppress replay attacks. However, depending on the environment in which the protocol is used, these problems may not be significant. For networks where devices have limited communications resources, e.g. mobile and ad hoc networks, the proposed timestamp management scheme may be an efficient alternative to the use of nonce-based protocols.

References

[1] L. Chen, D. Gollmann, and C. Mitchell. Tailoring authentication protocols to match underlying mechanisms. In J. Pieprzyk and J. Seberry, editors, *Information Security and Privacy — Proceedings: First Australasian Conference, Wollongong, NSW, Australia, June 1996*, pages 121–133. Springer-Verlag, Berlin, 1996.

[2] L. Gong. A security risk of depending on synchronised clocks. *ACM Operating Systems Review*, **26**(1):49–53, January 1992.

[3] K.-Y. Lam. Building an authentication service for distributed systems. *Journal of Computer Security*, 2:73–84, 1993.

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[5] C. J. Mitchell. Making serial number based authentication robust against loss of state. *ACM Operating Systems Review*, 34(3):56–59, July 2000.