Freshness Assurance of Authentication Protocols

Kwok-yan Lam and Dieter Gollmann

Department of Computer Science, Royal Holloway, University of London

Abstract. This paper describes various ways of providing freshness assurance of authentication protocols. It approaches the issue by discussing the notion of time in distributed authentication. In the context of authentication, we identify the places where the concept of time is needed, and describe the ways that timeliness of authentication protocols can be achieved.

Key words Distributed Operating Systems, Authentication Protocols, Network Security, Clock Synchronization Protocol

1 Introduction

The notion of time was intentionally ignored in the early stage of computing history when programs were written mainly for numerical applications. It is not necessary to require such programs to contemplate the passage of time when they are expected to run on a uniprocessor architecture. The fact that the notion of time was not introduced in computer programs simplified the process of software development. In a multiprogramming environment, it is the task of the operating system to ensure that independent programs executing concurrently do not interfere with each other; hence programmers are relieved from the trouble of triggering a context switch after executing for a certain period of time.

The development of distributed programming required the concept of time to be introduced in computer programs. When a program is designed to run on a number of processors distributed over a connected network, there must be some way to allow the activities of its distributed components to be synchronized. Hence it is necessary that various components of the distributed program have a consistent view to the order in which events occur during the course of program execution. The concept of logical clocks was introduced as a tool for specifying and designing distributed programs without using physical clocks. This concept arose from the abstract point of view that a clock is just a way of assigning a number to an event, where the number is thought of as the time at which the event occurred.

In the context of computer programming, the concept of real-time clocks evolved as situations were encountered where logical clocks did not meet the needs of an application. When a distributed program, e.g. an airline reservation program, is required to anticipate human activities, the way how the set of system events are ordered must be consistent with the view of human users. This requirement implies that the computer system must be able to assign numerical values (clock values) to events in the same way as human beings do. Therefore, real-time clocks are used for ordering distributed events. Note that the concept of real-time clocks is necessary for any application that is specified in terms of physical time, e.g. programs that are developed to operate under real-time constraints.

With regard to the security of distributed systems, however, the concept of time arises because of the threat of message replays in an open network environment. Information transmitted over communication lines is vulnerable to various kinds of security attacks such as eavesdropping and tampering. The use of cryptographic techniques is usually helpful for preventing unauthorized information disclosure and detecting unauthorized modification of information. However, an intruder may still be able to violate a system's security policy without disclosing or modifying the contents of messages in transit. In some cases, it is sufficient to simply record a message and then to replay it after some time. We shall discuss this in more detail in the next section.

Therefore, security protocols need to be equipped with the concept of time in order to counteract replay attacks. Here, the use of time is not for ordering distributed events nor for triggering events within some specified time, instead time is used to provide freshness assurance of objects for applications that are vulnerable to replay attacks.

The objective of this paper is to discuss the role of time in distributed authentication. Authentication plays an important part in ensuring data secrecy and authenticity because of its goals of allowing any pair of communicating parties to mutually verify each others identity and to distribute cryptographic keys for communications between the pair. Owing to the possibility of replaying messages by an intruder, the design of authentication protocols needs to take into account the concept of time, and to allow the legal participants of an authentication protocol to use different keys for communication sessions initiated at different time. In this discussion, we assume that message passing is the only mechanism by which distributed processes communicate and we are mainly concerned with freshness of contents of messages.

2 Time in Distributed Authentication

Communicating parties in a distributed system, frequently refered to as principals, may wish to convince themselves of the identity of the principals they are communicating with not only because of personal curiosity but also because of legal and financial implications. Principals may be held responsible for activities perpetrated in their 'name' or be billed for services used in their 'name'. Usually, the identity of a principal is linked to certain data, called its credentials, which contain some secret information. The link between a principal's identity and its credentials are guaranteed by some trusted agency. During authentication, a claimed identity is verified by demonstrating knowledge of the appropriate secret. To mutually authenticate two principals, the role of claimant and verifier is then reversed. If the verifier does not trust the agency guaranteeing the claimant's credentials, authentication may still be possible if a chain of trust can be established between the verifier and that agency, using intermediary agencies. Authentication may demand a certain administrational and computational effort from the principals. Thus, in designing authentication schemes, we have to be aware that principals may be a variety of computing devices, ranging from workstations to smart cards, and may have quite different functions in a distributed system.

Passwords provide a simple authentication scheme. The claimant is authenticated by presenting its password to the verifier. An obvious weakness of this scheme is the fact that the verifier is now in possession of the claimant's secret and could pose as the claimant in communications with third parties. To prevent impersonation, it is thus essential that secrets are not divulged during authentication. Cryptographic mechanisms exist that demonstrate knowledge of a secret without giving it away. Authentication that protects the claimants' secrets is called strong authentication. However, strong authentication is not sufficient to foil impersonation. If a claimant would always use the same messages to demonstrate knowledge of its secret, any third party could successfully assume the claimant's identity by replaying those messages. (We assume that messages passed in the distributed system are accessible to other users of the system.) More strongly, if the same authenticating message would be accepted twice by a verifier, replay attacks are feasible. If, for example, a fixed one-way hash function were applied to a password before transmitting it to the verifier, nothing would have been achieved. An attacker, although unable to reconstruct the password, need only present its hashed version to the verifier to assume a false identity. Cash dispensing machines provided another popular example for replay attacks. If a message that dispenses a specified amount of cash while debiting that amount to some specified account would be accepted as valid when replayed, an attacker could repeatedly extract money at the cost of a victim who would find it difficult to prove that these transactions had been fraudulent. Authentication schemes therefore have to check the freshness of messages.

There exist various ways of coping with this problem. We could focus on the messages and issue sequence numbers to distinguish old from new messages (see our previous reference to logical clocks). Each verifier keeps track of the last sequence number for each claimant it has dealt with. New messages are accepted only if their sequence number is higher than the sequence number recorded. Existing authentication protocols do not favour this approach, probably because of the administrational overheads incurred, in particular in large open distributed systems. Instead, they try to establish the 'actual presence of the claimant', i.e. a synchronization in the original meaning of the word. For this purpose it seems natural to employ timestamps that are obtained from clocks. However, we will follow Turski [15] in argueing that time derived from clocks is certainly not mandated by the postulation of our problem and may possibly be even inferior to other mechanisms that achieve the same goal. Indeed, by taking recourse to (local) clocks we find ourselves addressing questions of clock synchronization in a hostile environment. To solve this problem some sort of key exchange seems almost mandatory and it has been argued that we have thus only shifted the problem without actually changing its nature [3]. Challenge/response (handshake) protocols provide an alternative to clock based synchronization. The price that has to be paid for abandoning clocks is an increase in the communication between principals. In this paper, we will examine how existing authentication protocols handle synchronization and explore the relative merits of the two main design alternatives.

In distributed authentication, the notion of time does not only help to protect against replay attacks. Security management requires mechanism for terminating the validity of a principal's credentials. Immediate revocation is possible when the credentials are kept with a trusted agency, where they can be obtained by the verifier, or when the verifier always checks the validity of credentials with a trusted agency. Both solutions require communication between the agency and the verifier. Although immediate revocation of credentials would be preferable for reasons of security, many schemes settle for the less expensive method of restricting the life-time of credentials. Again, we find as the two main alternatives the reference to properly synchronized clocks, or perhaps more accurately, properly synchronized calendars, and a handshake protocol, this time between the verifier and the principal guaranteeing the claimant's credentials.

3 Clock-Based Authentication

The use of clocks is one way among others to achieve freshness in distributed authentication. Freshness is assured by the incorporation of timestamps in each encrypted message. A principal accepts a message as fresh only if the message contains a timestamp which is close enough to its knowledge of current time. As a consequence, delays before secure communications start can be reduced. This feature is especially desirable for distributed applications which use a connectionless communication service as a major mechanism for interprocess communications. In addition, revocation of access rights in a distributed system is made possible if the lifetime of credentials is indicated by timestamps. This offers an economical approach to implement revocation of rights, but at the cost of delaying the time at which revocation actions are effective.

In order to illustrate the idea, we use the Kerberos Authentication System as an example. Kerberos is an authentication system developed at MIT within the project Athena [8, 14]. Athena aims at providing computing resources to students across and beyond the whole campus and thus Kerberos is mainly geared towards authenticating the client/server communications that are predominant in such an environment. A more detailed analysis of the environment Kerberos was developed for and its influence on the design is given in [3].

Kerberos is based on DES [10] and the Needham-Schroeder key exchange protocol [11]. Principals are authenticated by the possession of some prearranged secret cryptographic key, derived with a one-way algorithm from their password. This key is shared with the Kerberos authentication server (KAS). The KAS issues a ticket, which, in general, is valid throughout one login session, and enables a principal to obtain other tickets from ticket granting servers (TGS) to access network services requiring authentication. Tickets essentially consist of cryptographic session keys, generated by the TGS, and timestamps, derived from local clocks, along with identifiers. The session keys are used for symmetric encryption during later communication with the desired server/service. As Kerberos servers generate the keys used in subsequent communications between principals, Kerberos servers have to be on line and trust in the servers has to encompass trust that servers will not misuse their ability to eavesdrop.

The following table, taken from [16], shows the steps involved in establishing a mutually authenticated session key between a server *S* and a client *C* acting on behalf of a user *U*. Each protocol step is numbered by an integer appearing in front of its description. The sequence of activities from step 1 to step 8 are regarded as part of the login procedure that a user needs to perform in order to access the computer system. A successful completion of these eight steps results in possession of a valid ticket to access a TGS and knowledge of the encryption key for communication with the TGS over the current login session. In this discussion, we are mainly interested in the actions specified from step 9 to step 18. This sequence of steps describes the way through which a client and a server establish a shared communication key and verify each other's identity.

1.	U→C	: U
2.	C→KAS:	U,TGS
3.	KAS	: $\text{tick}_{\text{TGS}} = \{\text{U,TGS,k,T,L}\}_{k_{\text{TGS}}}$
4.	KAS→C:	$\{\text{TGS},k,\text{T,L},\text{tick}_{\text{TGS}}\}_{k_U}$
5.	C→U	: 'Password ?'
6.	U→C	: passwd
7.	C	: decrypt $\{\text{TGS},k,\text{T,L},\text{tick}_{\text{TGS}}\}_{k_U}$
		: where $k_U = \text{f(passwd)}$ to obtain $k,\text{tick}_{\text{TGS}}$;
8.		: stop if decryption fails or T is out of date;
9.	C→TGS:	$\text{S},\text{tick}_{\text{TGS}}, \{\text{C},\text{T}_1\}_k$
10.	TGS	: obtains k from tick_{TGS}, and thus also C,T_1;
11.		: checks freshness of T_1 with respect to local clock;
12.		: $\text{tick}_S = \{\text{C,S},k',\text{T',L'}\}_{k_S}$
13.	TGS→C:	$\{\text{S},k',\text{T',L'},\text{tick}_S\}_k$
14.	C	: decryption with k gives k',tick_S
15.	C→S	: $\text{tick}_S, \{\text{C},\text{T}_2\}_{k'}$
16.	S	: obtains k' from tick_S, and thus also C,T_2;
17.		: checks freshness of T_2 with respect to local clock;
18.	S→C	: $\{\text{T}_2+1\}_{k'}$

With reference to the table above, a client *C* wanting some service from a server S does so by performing step 9 of the above sequence. The transmitted message carries enough information (a ticket containing a session key k and a good timestamp encrypted using k) to allow the TGS to be convinced that the received message was generated by C very recently (steps 10-11). The TGS, if satisfied with the claimant's word, grants a ticket and a session key to C which are used together to access S (step 12). The client C then uses the received ticket to access S in a similar manner. It is worth noting that this protocol employs timestamps whenever freshness assurance of messages is needed (see steps 9,13,15 and 18).

To appreciate the merits of clock-based authentication, we now discuss how Kerberos exploits the assumed behavior of clocks. There are three main points that we are most interested in. Firstly, we would like to know the number of message exchanges between C and S before authenticated communication starts. Secondly, we are concerned with the amount of long term data that a server has to maintain for each communicating client. Thirdly, we are interested in the ways that revocation of right can be achieved. Back to the table again, the client C first contacts server S at step 15. The message passed from C to S in this step allows S to retrieve the authorized session key to communicate with C which in turn is used to interpret the data structure $\{C, T_2\}_{k'}$, which is called an authenticator, in order for verifying the authenticity of the claimant. Any request for service from S can also be carried by the same message which is protected by proper security mechanisms. Therefore, the first message exchange between C and S is the first message that C contacts S for requesting services. This is a desirable feature in that C and S are not required to go through a handshake procedure before request messages are sent. In addition, the server S, being capable of learning the authorized communication key from a valid ticket (a ticket that contains a valid timestamp) and of checking freshness of the message using the authenticator, is not required to maintain any state information for ensuring secure communication between C and itself. This feature, together with the previous one, is very useful when building distributed applications in the Athena environment. Further, revocation of user rights is possible, because authentication servers are asked for permission at each login and KAS tickets have a limited life-time. The use of timestamps for indicating tickets' life-time, though not providing immediate revocation, has the advantage that a server is not required to contact the KAS (or TGS) whenever a ticket is required.

One major drawback of the timestamp-based approach is that it assumes the presence of a globally accessible clock. This globally accessible clock must be highly reliable and available, because security checks are invloved in every distributed operation of a properly protected system and authentication plays an essential role in most security functions. The clock is also required to be highly secure in that the security strength of a clock-based authentication mechanism relies on the fact that timestamps are correctly generated by it.

In a distributed system, rather than using a single, centralised, time service for timestamp generation, a reliable global clock is usually approximated by using the individual processor clocks and requiring each processor to bring their clock values close to each other by means of some fault-tolerant clock synchronization protocol [9, 13]. Each individual processor is said to implement a local time server. A distributed implementation of the clock service reduces the reliance on a single component of the system, thus makes the clock service more available.

The clock synchronization protocol itself must be fault-tolerant so that the clock values on each correct processor are reliable in the face of network and other processor failures. The time service must provide accurate and precise time, even with relatively large stochastic delays on the communication paths. A clock synchronization algorithm operates by measuring clock offsets between the various time servers in the distributed system and so is vulnerable to statistical delay variations on the various communication paths. As the size of the network increases, the paths involved can have wide variations on their delays. Well-designed algorithms are needed to improve the accuracy of delay and offset measurements made over statistically noisy communication paths and to select the best clock from among a set of mutually suspicious clocks. In addition, it is necessary that the synchronization protocol operates continuously and provides update information at rates sufficient to compensate for the expected drifts among the set of clocks.

In order to replace the role of a globally accessible clock in distributed authentication, the system of closely synchronized clocks must be capable of handling possible security attacks. Therefore, cryptographic techniques should play an important part in the clock synchronization protocol. The Byzantine Clock Synchronization Protocol discussed in [7] is one such example that makes use of digital signatures. The problem, however, is that performance of the synchronization protocol may be degraded in that cryptographic operations introduce extra delay in every communication path. Further, the ways that keys for such operations are distributed raises another problem since, almost invariably, key distribution is one of the original goals of authentication protocols. At a first glance, this seems to suggest that the philosophy of building an authentication mechanism whose correctness relies heavily on a yet "to be secured" clock syncrhonization protocol is questionable, because the reliance relationship is a mutual one.

Worse yet, even if a highly secure clock synchronization protocol is available, the security strength of clock-based authentication is still doubtful. To understand this, one should note that message delivery has a finite speed and that distributed clocks are unlikely to have identical values at all time. Therefore, the recipient of a timestamped message should take this into account and prepare to accept messages whose timestamps are not exactly the same as that of its local clock. The size of such an acceptance window is determined by system parameters such as expected message delivery delay and performance of the clock synchronization protocol. The acceptance window is assumed to be large enough that a majority of fresh messages are accepted correctly by their intended recipients, and yet small enough to detect intruders' replay attacks. However, this assumption is very difficult to realise since intruders are most likely to have their tools running in the system and ready to perform the replay attack whenever target messages appear [3]. More importantly, it is unreasonable to assume that replay messages necessarily take longer than the expected message delivery delay. Therefore, however small the acceptance window is, it must be larger than the minimal possible replay time. If the resulting gain of a successful replay attack is significant, we should not under-estimate an intruder's ability to replay within the acceptance window. This seems to suggest that timestamp-based approach should not be used in a sensitive environment.

4 Authentication by Challenge/Response

The use of challenge-response operations is another major approach to ensure freshness in distributed authentication. With this approach, a principal A expecting a fresh message from another principal B first sends a random number (challenge) to B, and requires that this random number appears in the subsequent message (response) received from B. These protocols come in various shapes. The challenge could be protected so that it can only be read by the legitimate recipient. Any correct response therefore would have to originate from there. Alternatively, the challenge could be open but the expected response be such that it can only be provided by the legitimate recipient.

In all cases, it is A's responsibility to ensure the quality of the random number it chooses. Incidentally, randomness is a rather misleading term, both because different readers may disagree in their intuitive interpretation of randomness and because different protocols have different requirements on the nature of the challenges. If, for example, the response were the challenge encrypted by a varying session key, even a fixed challenge value would be acceptable. If the same key were used repeatedly in such a protocol, then the challenge could be a time-stamp. In this case it is only important that it is highly unlikely that the same time-stamp is used twice, i.e. the challenge has to be a nonce. In other situations the challenge may be required to be 'meaningless' as a recipient may refuse to sign arbitrary messages. Finally, in a protocol where the value of the challenge contributes to the generation of a session key, it may well be advisable that the challenge is chosen truly at random. The authentication scheme discussed below can serve as an example for such a protocol.

When we are dealing with 'truly random' challenges, i.e. challenges of the last kind where each possible value may be picked with equal probability, then we face the problem of automating such a random choice. True random generators could be used but would not be part of common computing and communications equipment. The existing hardware could be used for implementing pseudo random generators. This approach solicits two remarks. Firstly, a poor or corruptable pseudo-random generator may open new possibilities for attacking a scheme. This, of course, is a well known fact which has been highlighted by recent discussions on the proposed NIST digital signature standard. Secondly, we may view the approximation of randomness by pseudo randomness as a situation similar to that encountered in clock-based systems, where a global clock was approximated by local clocks.

We use Selane [2, 6], developed at the E.I.S.S., as an example for a challenge/-response-based authentication service for distributed systems. Principals obtain their credentials from Secure Key Issueing Authorities (SKIAs). These SKIAs are active only during user registration and may be offline as they are not involved subsequently in authentication between principals. The SKIAs can be set up in two different ways. In one case, the SKIAs are privy to the session keys generated by the principals. In the second case, the SKIAs have only to be trusted to issue correct credentials. The ElGamal public key signature system is used for user registration and authentication [4]. Authentication is achieved by establishing a common session key that is derived from the principals' credentials. Hierarchical network structures are possible. Two networks can be connected by setting up a common SKIA on top of the respective SKIAs. This happens at the expense of having to re-register all principals.

A challenge/response-based protocol is used to establish the freshness and authenticity of session keys. This mechanism is not available for revocation as the SKIAs are not involved in the actual process of authentication. Presently, revocation of credentials is only partially addressed by setting an expiry date. If, however, one would allow for on-line Credential Guaranteeing Agencies so that verifiers can check the validity of credentials with these agencies, then credentials could be revoked immediately.

The authentication protocol relies strongly on some mathematical properties of the underlying cryptographic mechanisms. The SKIA issues a credential of the form (m_A, r_A, s_A) to the principal A, where m_A is A's identity and (r_A, s_A) the SKIA's signature of m_A. The value s_A has to be kept secret by A. There exist functions f and g so that

$$f(m_A, r_A, x) = g(g(r_A, x), s_A).$$

We omit the details and refer the reader to [5, 1, 2, 6]. The following table gives a simplified description of a one-way authentication from principal A to principal B.

A→B	:	m_A, r_A
B	:	chooses a nonce n_B and
	:	computes and retains $k_A = f(m_A, r_A, n_B)$
B→A	:	$g(r_A, n_B)$
A	:	computes and retains $k_A = g(g(r_A, n_B), s_A)$

At this stage, B can be sure that k_A is a key shared with A. By changing their roles, A and B can generate another key, k_B, where A can be sure that k_B is a key shared with B. A session key k_{AB} could be formed by combining these two keys. However, it is still possible that k_A is shared between A and an attacker C, or that k_B is shared between B and an attacker C'. Thus, the combination k_{AB} is not necessarily a key shared by A and B. Further four steps in a handshake protocol can establish that this is the case. Messages encrypted with a key shared with an attacker would not be compromised but be unreadable for the intended recipient.

Thus, overall eight steps are required before secure communication between A and B starts. If A is known to initiate the communication, only two steps would be necessary to allow A to establish that k_{AB} is shared with B. For example, A could challenge B with a nonce n_A and proceed if it receives the response $\{n_A\}_{k_{AB}}$. (At that moment, the combined key is indeed shared between A and B but only A is aware of it.) In addition, principals have to keep track of the state of the protocol they have reached and of the keys and nonces used in the challenges.

Selane was developed mainly with 'symmetric' relationships between principals in mind. Now, consider the case where A is a client approaching a server B and compare the actions B has to take in Selane with the actions of the server S in Kerberos. Clearly, in this respect B is at an disadvantage as it is involved in more steps and has to keep state information for all its current clients. The advantage this price has been paid for is the independence of an underlying clock service that may compromise the security of the authentication system.

5 Conclusion

We have discussed the major ways that freshness assurance of distributed authentication is provided. The discussion was based on the notion of time which plays an important role in the design of authentication protocols. This concept is usually realized in either of two ways: challenge/response or clock. Both have their own advantages and disadvantages. This paper highlighted the important issues involved in the design of authentication protocols with freshness assurance, and suggested aspects for further research. We are currently conducting extensive experiments to examine the quantitative differences between the two approaches.

References

1. Bauspieß, F., Knobloch, H.-J.: How to Keep Authenticity Alive in a Computer Network. Proceedings Eurocrypt'89, Springer LNCS **434** (1990) 38–46
2. Bauspieß, F.: SELANE: An Approach to Secure Networks. Proc eedings SECURICOM 90 (1990) 159–164
3. Bellovin, S.M., Merritt, M.: Limitations of the Kerberos Authentication System. ACM Computer Communications Review **20(5)** (1990) 119–132
4. ElGamal, T.: A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory **31** (1985) 469–472
5. Guenther, C.G.: An Identity-Based Key-Exchanged Protocol. Proceedings Eurocrypt'89, Springer LNCS **434** (1990) 29–37
6. Horster, P., Knobloch, H.-J.: Protocols for Secure Networks. Proceedings Eurocrypt'91, Springer LNCS **547** (1991) 399–408
7. Lamport, L., Melliar-Smith, P.M.: Byzantine Clock Synchronization. ACM Operating Systems Review **20(3)** (1986) 10–16

8. Miller, S.P., Neuman, C., Schiller, J.I., and Saltzer, J.H.: Kerberos Authentication and Authorization System. Project Athena Technical Plan Section E.2.1, MIT (July 1987)
9. Mill, D.: Internet Time Synchronization: the Network Time Protocol. RFC 1129 (October 1989)
10. National Bureau of Standards: Data Encryption Standard. FIPS Publication 46 (1977)
11. Needham, R.M., Schroeder, M.: Using Encryption for Authentication in Large Networks of Computers. CACM **21(12)** (1978) 993–999
12. R.M. Needham, R.M., Schroeder, M.: Authentication Revisited. ACM Operating Systems Review **21(1)** (1987) 7
13. Schneider, F.B.: A Paradigm for Reliable Clock Synchronization. Proceedings of the Advanced Seminar on Real-Time Local Area Networks (1986)
14. Steiner, J.G., Neuman, C., Schiller, J.I.: Kerberos: An Authentication Service for Open Network Systems. Usenix Workshop Proceedings, UNIX Security Workshop, Portland (1988)
15. Turski. W.M.: What to do when we cannot depend on time. Workshop on Mathematical Concepts of Dependable Systems, Oberwolfach (1990)
16. Woo, T.Y.C., Lam, S.S.: Authentication for Distributed Systems. IEEE Computer **25(1)** (1992) 39–52