Authentication using cryptography

Chris Mitchell

Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX

Email: `cjm@dcs.rhbnc.ac.uk`

8th September 1997

ABSTRACT

The purpose of this paper is to describe the nature of user authentication and entity authentication based on the use of cryptographic methods. The relationship between key distribution and entity authentication is described, and examples of practical authentication protocols are given, together with some of the pitfalls awaiting designers of such protocols. References to the growing body of standardised authentication techniques are also provided.

1. What is authentication?

Authentication is a much over-used word in the context of Information Security. Even ISO 7498-2, [1], the 'OSI Security Architecture' recognises two distinct meanings for the word. ISO 7498-2 distinguishes between *data origin authentication*, i.e. verifying the origin of received data, and *(peer) entity authentication*, i.e. verifying the identity of one entity by another.

We are primarily concerned here with the second of these two services, namely entity authentication. Entity authentication is typically achieved using an *authentication exchange mechanism*. Such a mechanism consists of an exchange of messages between a pair of entities, and is usually called an *authentication protocol*. In OSI-speak, the word 'protocol' is reserved for the specification of the data structures and rules governing communication between a pair of peer entities, and this is why ISO 7498-2 speaks of authentication exchange mechanisms. Here we abuse the OSI notation, following generally accepted practice, and call them authentication protocols.

ISO 7498-2 defines entity authentication as 'the corroboration that an entity is the one claimed'. We also need to distinguish between protocols providing *unilateral authentication* and those providing *mutual authentication*. Unilateral authentication is defined as 'entity authentication which provides one entity with assurance of the other's identity but not vice versa' and Mutual authentication is defined as 'entity authentication which provides both entities with assurance of each other's identity'.

Entity authentication can only be achieved for a single instant in time. Typically, a mutual authentication protocol is used at the start of a connection between a pair of communicating entities. If security (e.g. confidentiality, integrity) is required for information subsequently exchanged during the life of the connection, then other cryptographic mechanisms will need to be used, e.g. encipherment or the use of *Message Authentication Codes (MACs)*, to protect that data. The keys needed for these cryptographic operations can be agreed and/or exchanged as part of the authentication protocol.

Thus one application of entity authentication is 'authenticated session key establishment'. Other applications exist which are not directly related to session key exchange. Examples of such applications include:

- secure clock synchronisation,
- secure RPC (remote procedure call), and
- secure transactions.

We have so far used the term entity, without actually saying what an entity is. For our purposes, an entity will either be a computer or a human user (perhaps equipped with some kind of authentication token). Thus, we mean to cover the two most commonly occurring applications of entity authentication protocols, namely:

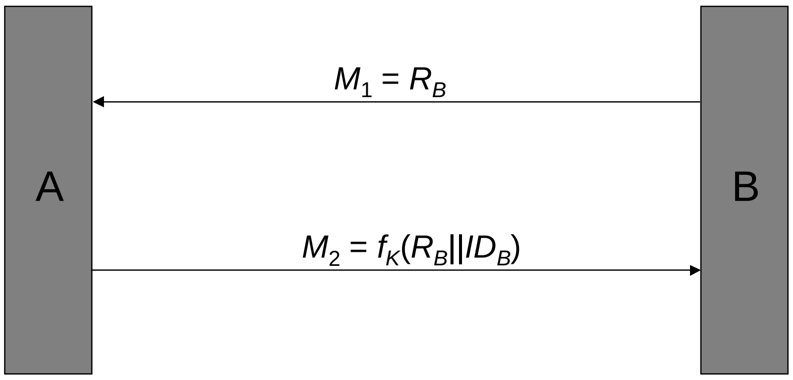
- where both entities are computers or other devices (e.g. a mobile telephone), and
- where one entity is a human and the other is a computer.

The latter case is often referred to as 'user authentication'. However, although the applications are subtly different, and are often discussed separately, the underlying problem

[1] ISO 7498-2: 1988, *Information processing systems – Open system interconnection – Basic reference model – Part 2: Security architecture*.

is essentially the same, and hence we do not make this distinction here. In fact, given that the human user has some kind of authentication token capable of performing calculations on the user's behalf, precisely the same protocols can be employed for the two applications.

Before proceeding, we give a simple example of a unilateral authentication protocol. In this protocol, entity *A* authenticates itself to entity *B*, but not vice versa. We suppose that, prior to use of this protocol, *A* and *B* have established a shared secret key *K*. This key is used with a cryptographic check function *f*; an example of such a function is provided by a block cipher based MAC. As shown in Figure 1, this protocol has two messages. In the first message, M_1, *B* sends *A* a random 'challenge' R_B. In the second message, M_2, *A* responds with a check value, computed as a function of the random challenge and an identifier for *B* (denoted ID_B). Note that in this example, as throughout, $X||Y$ denotes the concatenation of data items *X* and



Y.

Figure 1 - A simple challenge-response protocol

It is simple to see that the use of a secret key means that no-one apart from A is able to compute the correct response to B's challenge. Other aspects of the design of this protocol will be explained in the text below.

This type of protocol is often called a challenge-response (CR) protocol. The exact origins of these protocols is far from clear, although the seminal paper of Needham and Schroeder, [2], provides the first published examples of such protocols. CR protocols are widely used in many communications systems. For example, a CR protocol is the basis of the authentication scheme in the Global System for Mobile Communications (GSM), the standard for European digital mobile telephony, and CR protocols are widely used in token-based user authentication systems such as the Racal Watchword system.

[2] R.M. Needham and M.D. Schroeder, 'Using encryption for authentication in large networks of computers'. *Communications of the ACM* **21** (1978) 993-999.

2. Objectives of an authentication protocol

An authentication protocol consists of an exchange of messages between two parties. If it is a mutual authentication protocol then, after successful completion of the protocol, both parties should be sure of the origin, integrity and freshness of all the messages they have received. Therefore, they will have confidence in the presence of the claimed entity at the other end of the link. Information conveyed in the messages can then be used for a variety of purposes, notably key establishment, as we discuss below.

There are some much more sophisticated models of authentication protocols, including some set up formally. It has been necessary to devise these models because of the difficulty of formalising the apparently simple goals of an authentication protocol. The shortcomings of previous informal and formal models have been shown by the many varied attacks that have been discovered on authentication protocols. It is beyond the scope of this paper to describe these formal models or all the possible attacks; a good guide to some of the possible weaknesses in authentication protocols can be found in [3]. In the next section, we outline some of the simpler issues.

3. What can go wrong?

Whilst the objectives for an authentication protocol are relatively straightforward to express, and the protocols themselves contain only a few messages, analysing these protocols can be unexpectedly difficult. As we have already mentioned, the literature is full of examples of apparently secure protocols that have been shown to possess subtle flaws, which might compromise the protocol users in certain circumstances. To give a flavour of some of the problems that can arise we consider some of the most basic design issues for an authentication protocol.

The simplest (unilateral) authentication method involved use of a password. The entity being authenticated, when challenged, sends a copy of a secret password. The system then checks a one-way function of the password against the stored value for that entity. This is the basis of the Unix authentication system. The shortcomings of such an approach are all too clear – anyone capable of intercepting the communications between the two entities can obtain the secret password, compromising the system. Nevertheless, where such interception is deemed infeasible, or the impact of compromise is small, such systems remain very widely used.

Probably the most ‘obvious’ way of dealing with the problem of interception of passwords is to use an encryption mechanism to hide the password whilst in transit. However, it is also not hard to see that simple encryption of a password does little to help the interception problem. This is because the possessor of an intercepted encrypted password can simply replay the encrypted password, which will continue to be accepted! Something must be done to make the password message ‘one time’ (i.e. so that yesterday’s message cannot be replayed today).

There are two simple ways of achieving this one-time effect. The first employs a random challenge that is used in computing the password-based response – this is precisely the basis of the protocol described in Figure 1. An alternative approach, and one that can result in a unilateral authentication protocol with just one message, is to use a time-stamp. An example of such a protocol is given in Figure 2, where T_A denotes a time-stamp chosen by A.

[3] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of applied cryptography*. CRC Press, Boca Raton, 1997.

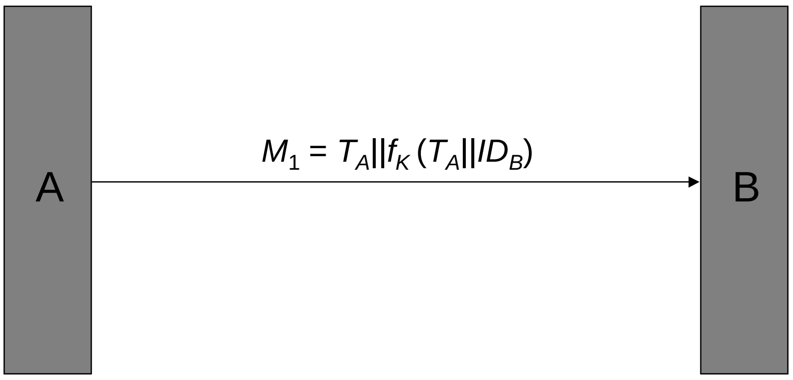


Figure 2 - Time-stamp based unilateral authentication

The time-stamp ensures that the authentication message is different every time. Entity *B* first verifies that the supplied time-stamp T_A is sufficiently current (and within a fixed error margin of its current time). *B* then recomputes the check-value using the supplied time-stamp, and compares it with the value sent in the message.

A further alternative, which is really a variant on the time-stamp approach, is to replace the time-stamp T_A with a bilateral sequence number. In other words, *A* and *B* both maintain a counter value, purely for use in authenticating *A* to *B*; the use of such a counter is discussed further below. Note that such a counter may be regarded as a 'logical' time-stamp.

Before proceeding note that a problem could arise in either of the protocols so far described of the identifier of the message recipient (ID_B) were removed from the inputs to *f*. Suppose that the same protocol (as in Figure 1, but without use of the identifier ID_B) is used in identifying *A* to *B* and *B* to *A*; suppose moreover that the same key *K* is used for authentication in both directions. We also suppose that *A* and *B* are computers capable of maintaining more than one communications channel simultaneously. Then, if *C* is impersonating *A*, when *C* receives the challenge R_B from *B*, *C* simply starts another authentication protocol with *B*, and sends *B* the challenge R_B. *B* responds with a check-value computed using *K* and R_B, which *C* simply 'reflects' back to *B*. *B* now falsely believes itself to be talking to *A*. The inclusion of the identifier avoids this problem by making the message sent from *A* to *B* in response to a challenge different from the message sent from *B* to *A* in response to the same challenge. The protocol in Figure 2 includes an identifier for similar reasons; for further details see [4].

[4] C.J. Mitchell, 'Limitations of challenge-response entity authentication'. *Electronics Letters* **25** (1989) 1195-1196.

Note also that once we consider mutual authentication other problems can arise. One of these concerns the linking of messages from a particular instance of a protocol. In circumstances where more than one instance of a protocol may be in progress at a particular point in time, it is sometimes important that the recipient of a message knows which protocol instance the message belongs to, and that the links between protocol messages cannot be undetectably tampered with by a malicious interceptor. This and other factors make designing mutual authentication protocols a non-trivial business.

4. Components of an authentication protocol

Having seen what an authentication protocol is for, we now consider in a little more detail what we need to use to build such a protocol. It should be clear that, if we ignore simple password schemes, authentication protocols require the use of a combination of either shared secrets (keys or passwords) or signature/verification key pairs, and accompanying cryptographic mechanisms. These are used to ensure that the recipient of a protocol message knows:

- where it has come from (origin checking),
- that it has not been interfered with (integrity checking).

Note that cryptographic mechanisms (by themselves) cannot provide *freshness-checking*, i.e. the verification that a protocol message is not simply a replay of a previously transmitted (valid) protocol message, protected using a currently valid key. We consider the provision of freshness verification later.

4.1 Cryptographic mechanisms

A variety of different types of cryptographic mechanism can be used to provide integrity and origin checking for individual protocol messages. We consider three main possibilities:

- encipherment,
- integrity mechanism (MAC or Cryptographic Check Function),
- digital signature.

We consider each in turn in some detail.

4.1.1 Use of encipherment

To protect a message in a protocol, the sender enciphers it with a secret key shared with the recipient. The recipient can then verify the origin of the message using the following process. The recipient first deciphers the message and checks that it ‘makes sense’; if this is the case then the recipient reasons that it must therefore have been enciphered using the correct secret key, and since only the genuine sender knows this key, it must therefore have been sent by the claimed originator.

This reasoning makes a number of assumptions about the nature of the encipherment algorithm and the capabilities of the recipient. First and foremost, if this process is to be performed automatically by a computer (as we would expect), then we need to define what ‘makes sense’ means for a computer, especially as the contents of the message might include random session keys and random ‘challenges’. We are also assuming that an interceptor cannot manipulate an enciphered message (without knowledge of the key used to encipher it) in such a way that it still ‘makes sense’ after decipherment. This constrains the type of encipherment algorithm that is suitable for use in this application; for example, stream ciphers are usually unsuitable for use as part of an authentication protocol.

The usual solution to this problem is the addition of deliberate 'redundancy' (according to some agreed formula) to the message prior to encipherment. The recipient of the message can then automatically check the presence of this redundancy (after decipherment). One common method of adding redundancy to a message is to calculate a *Manipulation Detection Code (MDC),* a sort of checksum dependent on the entire message, and append it to the message prior to encipherment. The MDC calculation function will typically be a public function.

4.1.2 Use of integrity mechanisms

There is a variety of possible types of data integrity mechanism. Two well-known examples are as follows.

- A *Message Authentication Code (MAC)* function. A MAC (or a Cryptographic Check Function) is a function which takes as inputs a secret key and a message, and gives as output a fixed length MAC which is appended to the message as a 'cryptographic check value'. The 'standard' way of computing a MAC is by using a block cipher (e.g. DES) in Cipher Block Chaining (CBC) mode. The result of enciphering the last block is typically subjected to some additional processing, and the output forms the MAC.
- A *one-way function*. A one-way function is a function that is simple to compute yet computationally infeasible to invert, i.e. given an input it is easy to compute the corresponding output, but given an arbitrary output it is computationally infeasible to find an input which gives this output. To use such a function as a data integrity mechanism involves concatenating the message to be protected with a secret key and using this as the input to the one-way function. The resulting output can then be appended to the message as a 'cryptographic check value'. Note that, for this approach to be secure, a slightly more powerful property than 'one-wayness' is required; however we omit the details here.

The recipient of a protocol message checks it by recomputing the cryptographic check value, using a shared secret key, and comparing it with the value appended to the message. The appended check value provides both:

- *origin checking*, because the originator and verifier are assumed to be the only possessors of the secret key, and hence are the only ones capable of computing a valid check value for a message, and
- *integrity checking*, because, without the secret key, the new check value corresponding to a changed message cannot be calculated.

4.1.3 Use of digital signatures

A digital signature function is an example of a public key scheme (i.e. an asymmetric cryptographic technique). The fundamental idea of asymmetric cryptography is that keys come in matching pairs made up of a *public key* and a *private key*, and that knowledge of the public key of a pair does not reveal knowledge of the private key.

The key pair for a digital signature algorithm consists of:

- a private 'signing key' (which defines the signature transformation), and
- a public 'verification key' (which defines the verification transformation).

A signature will typically function somewhat like a MAC, in that the possessor of the private key can use it to derive the *digital signature* of a message, which is then appended to the message. The recipient then uses the public verification key to check the signature, thereby enabling the recipient to check the origin and integrity of a message.

4.1.4 Classifying protocols

One way of classifying authentication protocols is by the type of cryptographic mechanism they use. This is the approach followed by ISO/IEC 9798, which is divided into a number of parts, where each part contains authentication protocols based on a different type of cryptographic mechanism. We discuss these standards further in Section 6 below.

4.2 Freshness mechanisms

As we have already briefly noted, providing origin and integrity checking for protocol messages is not all that is required. We also need a means of checking the 'freshness' of protocol messages to protect against replays of messages from previous valid exchanges.

As we have already seen, there are two main methods of providing freshness checking:

- the use of *time-stamps* (either clock-based or 'logical' time-stamps),
- the use of *nonces* or challenges (as in the challenge-response protocol in Figure 1).

We consider these two approaches in turn.

4.2.1 Use of time-stamps

Clearly, the inclusion of a date/time stamp in a message enables the recipient of a message to check it for freshness, as long as the time-stamp is protected by cryptographic means. However, in order for this to operate successfully all entities must be equipped with **securely** synchronised clocks. It is non-trivial to provide such clocks (n.b. the clock drift of a typical workstation can be 1-2 seconds/day). Moreover, every entity receiving protocol messages will need to define a time acceptance 'window' on either side of their current clock value. A received message will then be accepted as 'fresh' if and only if it falls within this window. This acceptance window is needed for two main reasons:

- clocks vary continuously, and hence no two clocks will be precisely synchronised, except perhaps at some instant in time, and
- messages take time to propagate from one machine to another, and this time will vary unpredictably.

The use of an acceptance window is itself a possible security weakness, since it allows for undetectable replays of messages for a period of time up to the length of the window. To avert this threat requires each entity to store a 'log' of all recently received messages, specifically all messages received within the last t seconds, where t is the length of the acceptance window. Any newly received message is then compared with all the entries in the log, and if it is the same as any of them then it is rejected as a replay.

Another problem associated with the use of time-stamps is the question of how synchronised clocks should be provided. One solution is to use an authentication protocol **not** based on time-stamps (e.g. nonce-based) at regular intervals to distribute a master clock value that is then used to update each entity's individual clock. Another solution is for all entities to have reliable access to an accurate time source (e.g. a national radio broadcast time such as the Rugby time signal).

One alternative to the use of clocks is for every pair of communicating entities to store a pair of *sequence numbers*, which are used only in communications between that pair. For example, for communications between A and B, A must maintain two counters: N_{AB} and N_{BA} (B will also need to maintain two counters for A). Every time A sends B a message, the value of N_{AB} is included in the message, and at the same time A's stored value of N_{AB} is incremented.

Every time *A* receives a message from *B*, then the sequence number put into the message by *B* (*N* say) is compared with N_{BA} (as stored by *A*), and:

- if $N > N_{BA}$ then
    - the message is accepted as fresh, and
    - N_{BA} is reset to equal *N*,
- if $N \leq N_{BA}$ then
    - the message is rejected as an 'old' message.

These sequence numbers take the role of what are known as *logical time-stamps*, a well-known concept in the theory of Distributed Systems (following Lamport, [5]).

4.2.2 Use of nonces

Nonce-based (or *challenge-response*) protocols use a quite different mechanism to provide freshness checking. One party, *A* say, sends the other party, *B* say, a *nonce* (*Number used ONCE*) as a challenge. *B* then includes this nonce in the response to *A*. Because the nonce has never been used before, at least within the lifetime of the current key, *A* can verify the 'freshness' of *B*'s response (given that message integrity is provided by some cryptographic mechanism). Note that it is always up to *A*, the nonce provider, to ensure that the choice of nonce is appropriate, i.e. that it has not been used before.

The main property required of a nonce is the 'one-time' property. Thus, if that is all that is ever required, *A* could ensure it by keeping a single counter and whenever a nonce is required, for use with any other party, the current counter value is used (and the counter is incremented). However, in order to prevent a special type of attack, many protocols also need nonces to be *unpredictable* to any third party. Hence, nonces are typically chosen at random from a set sufficiently large to mean that the probability of the same nonce being used twice is effectively zero.

5. Evaluating authentication protocols

As mentioned above, there have been many efforts in recent years to devise formal and semi-formal methods for evaluating authentication protocols. Probably the most influential work in this area has been that of Burrows, Abadi and Needham, [6], who devised a special logic to reason about authentication protocols. This has spawned an enormous literature on similar topics, and remains an active area for current research. It seems that the answer to the question 'What is an authentication protocol?' remains to be fully answered.

[5] L. Lamport, 'Time, clocks, and the ordering of events in a distributed system'. *Communications of the ACM* **21** (1978) 558-565.

[6] M. Burrows, M. Abadi and R. Needham, 'A logic of authentication'. *Proceedings of the Royal Society of London, Series A* **426** (1989) 233-271.

6. Standards for authentication protocols

In recent years ISO/IEC JTC1/SC27 has been working on a multi-part standard, ISO/IEC 9798, [7], specifying a general-purpose set of authentication protocols. The four parts published so far have the following coverage.

- *Part 1 - General* (this part has recently been revised with much more explanatory text),
- *Part 2 – Mechanisms using symmetric encipherment algorithms*,
- *Part 3 – Entity authentication using a public key algorithm* (this part, covering the use of digital signatures, is currently being revised under the new title *Mechanisms using asymmetric signature techniques*),
- *Part 4 – Mechanisms using a cryptographic check function*.

Note that the example protocols given in Figure 1 and Figure 2 are taken from ISO/IEC 9798-4. A fifth part of ISO/IEC 9798 is currently at Draft International Standard (DIS) stage:

- *Part 5 – Mechanisms using zero knowledge techniques*.

The protocols specified in these standards have been designed for use in a variety of application domains. As such, they have been designed to be as 'robust' as possible, i.e. they have been designed to resist all known attacks (as long as they are used in the way specified). The reader is recommended to consult these standards before adopting an authentication protocol for use in a new application domain.

[7] ISO/IEC 9798, *Information technology – Security techniques – Entity authentication mechanisms*.