Developments in security mechanism standards

Chris Mitchell
Information Security Group, Royal Holloway, University of London

1 Introduction

1.1 International standardisation

Over the last ten years, in parallel with the enormous growth in the use of cryptography, major efforts have been devoted to assembling a set of internationally agreed standards for cryptographic mechanisms. These standards have been prepared by ISO/IEC JTC1/SC27, a committee devoted entirely to security standardisation. Of course, the standards produced by SC27 are not the only cryptographic standards in existence. A number of important standards have been produced by a variety of other bodies, including the following.

- From the early 1980s onwards, the US banking community has produced a range of US (ANSI) standards covering the use of cryptography in retail and wholesale banking. The standards have had a very strong influence on subsequent international banking standards on cryptography and its use. In turn these banking standards have motivated some of the general purpose standards developed by SC27.
- The Internet community has produced a number of RFCs covering a range of cryptographic algorithms. These RFCs have been primarily aimed at providing algorithms for use in specific secure Internet protocols (e.g. for secure email and secure IP). Nevertheless, some of the schemes adopted as RFCs have become widely used in many applications outside of the Internet sphere.
- A variety of national, regional and industry bodies have proposed standards for cryptographic techniques. Examples include the pioneering US standards for the DES block cipher, the DSA signature algorithm, and the SHA-1 hash algorithm, and the

European ETSI standards for cryptographic algorithms for use in mobile telecommunications (some of which remain confidential).

1.2 Scope of this chapter

However, despite this wide range of standardisation activity, the ISO/IEC JTC1/SC27 work is unique in being both truly international and also aimed at general applications. As such, while we mention the relevant work of other standards bodies, the main focus of this chapter is the work of ISO/IEC JTC1/SC27. The main purpose of this chapter is to bring the international standards for cryptographic techniques to the widest possible audience. Adoption of these standards, which have received detailed scrutiny from experts world-wide, can only help to improve the quality of products incorporating security features.

Note that much of the work described in this chapter is based on recent research. For brevity, references to research papers are not included here. For further information the interested reader should consult the bibliographies in the quoted standards, or the excellent encyclopaedic work (Menezes, van Oorschot and Vanstone, 1997).

1.3 Contents

The chapter will consider the full spectrum of international standardisation of cryptographic techniques. Thus will involve covering the following areas:

- Encryption algorithms,
- Modes of operation for block ciphers,
- Message Authentication Codes (MACs),
- Digital signatures and Hash-functions,
- Entity authentication,
- Non-repudiation, and
- Key management.

The main technical contents of the international standards covering these topics are outlined, and some motivation for their contents, as well as indications on their application, will be provided. This discussion of international standards for cryptographic techniques is prefaced by a short introduction to the main standards-making bodies, and an overview of some of the most significant parts of ISO/IEC 7498-2, the OSI Security Architecture. This latter standard is strictly outside the scope of this chapter, but it is useful in that it provides a standardised framework for the description of security services and mechanisms.

2 Standardisation bodies

2.1 Overview

The main international standards bodies relevant to information security are:

- International Organization for Standardization (ISO),
- International Electrotechnical Commission (IEC),
- International Telecommunications Union (ITU), the successor to CCITT and CCIR.

There is some collaboration between these bodies; for example, in the area of IT, ISO and IEC have formed a Joint Technical Committee (JTC1). At the European level, the 3 standards bodies roughly corresponding to ISO, IEC and ITU are respectively:

- Comité Européen de Normalisation (CEN),
- Comité Européen de Normalisation Eléctrotechnique (CENELEC),
- European Telecommunications Standards Institute (ETSI).

The National Standards bodies (members of ISO and IEC) also produce their own national standards. In Europe, ECMA (the European Computer Manufacturers Association) have produced standards for distributed system security. In North America, the IEEE (Institute of Electrical and Electronics Engineers) and NIST (the National Institute for Standards and Technology) produce IT security standards of international importance. IEEE work includes

LAN security and POSIX standards. NIST (the successor to NBS) produces standards for use by Federal Government bodies. Also of global significance is the work of ANSI (the American National Standards Institute), particularly for its banking security standards.

2.2 Internet standards

The *Internet* is the result of interconnecting a worldwide community of government, academic and private computer networks. It started as a project sponsored by the U.S. government, and it grew organically based largely on academic and research institutions. In recent years the Internet has expanded to include many private organisations wishing to make use of the communications facilities the Internet provides. The operation of the Internet relies on interconnection standards, primarily those designed specifically for Internet operation. The Internet is managed by the *Internet Activities Board (IAB)*, which delegates the main responsibility for the development and review of its standards to the *Internet Engineering Task Force (IETF)*. Final decisions on Internet standards are made by the IAB.

2.3 ISO standards

2.3.1 Overview

ISO, founded in 1946, is a worldwide federation of national standards bodies. A *Member Body* of ISO is the national body 'most representative of standardisation in that country' (e.g. BSI in the UK). ISO (and IEC) assigns responsibility for the development of standards in particular areas to *Technical Committees (TCs)*. A technical committee determines its own programme of work, within the scope specified by its parent body (ISO or IEC). The main TCs responsible for security relevant standards are:

- ISO/IEC JTC1: *Information technology*, and
- ISO TC68: *Banking and related financial services*.

TCs establish *Sub-Committees (SCs)* to cover different aspects of their work. SCs, in turn, establish *Working Groups (WGs)*, to deal with specific topics. While the structures of TCs, SCs and WGs evolve over time, the evolution is not rapid, and these groups typically exist for several years. An ISO standard moves through the following phases in its development:

- New Work Item (NWI) Proposal and TC Ballot,
- Appointment of editor,
- Series of Working Drafts (WDs),
- Committee Draft (CD), Final CD (FCD), and associated ballots,
- Draft International Standard (DIS), Final DIS (FDIS), and associated ballots,
- International Standard status,
- 5-yearly review.

Provisions are also made for standards to be revised earlier than the five-yearly review cycle if defects are found. This operates via a 'defect report' system.

2.3.2 ISO security standards

As previously mentioned, the main ISO TCs responsible for security are ISO/IEC JTC1 and ISO TC68. ISO TC68, responsible for banking standards, has produced a wide variety of security standards (including ISO 8730, ISO 8731-1, and ISO 8731-2 specifying integrity mechanisms, and ISO 8732, ISO 11166-1 and ISO 11166-2 specifying key management methods).

The main security-relevant SCs within JTC1 (responsible for Information Technology) are as follows:

- SC6: *Telecommunications and information exchange between systems*,
- SC17: *Identification cards and related devices*,
- SC18: *Document processing and related communication*,
- SC21: *OSI, data management and Open Distributed Processing*.

- SC27: *IT Security Techniques*.

The security techniques standards produced by SC27 is the main focus of this chapter. The work of SC27 is divided into three working groups.

- WG1 is responsible for security management and liaison with other standards groups.
- WG2 is responsible for security mechanism standards.
- WG3 is concerned with computer security (evaluation criteria, etc.).

We are primarily concerned here with the work of WG2, although we do consider two standards (ISO/IEC 9979 and ISO/IEC 11770-2) developed by WG1.

3 The OSI Security Architecture

We start our discussion of security standards by considering ISO 7498-2 (ISO, 1989) the *OSI security architecture* developed by JTC1 SC21. ISO 7498-2 is intended to serve as a security-specific addition to ISO 7498, the OSI reference model. In doing so it defines many security-related terms and ideas which are of importance to a variety of application areas, including many not covered by the OSI model. Of particular importance is the terminology it introduces for the description of security services and mechanisms.

3.1 Security model

The underlying model, implicit to the discussion in ISO 7498-2, is that there is a generic *security life-cycle*, containing the following steps:

- definition of a security policy, containing a rather abstract series of security requirements for the system,
- a security requirements analysis, including a risk analysis, possibly using a tool such as CRAMM, and an analysis of governmental, legal and standards requirements,
- definition of the security services necessary to meet the identified security requirements,

- system design and implementation, including selection of security mechanisms to provide the chosen security services, and
- continuing security management.

In the context of this model, a *security threat* is something that poses a danger to a system's security. A *security service* is selected to meet an identified threat, and a *security mechanism* is the means by which a service is provided. It is important to note the distinction between a security service, i.e. what is provided for a system, and a security mechanism, i.e. the means by which a service is provided. Hence *confidentiality* is a service, whereas *encryption* is a mechanism which can be used to provide confidentiality. In fact encryption can be used to provide other services, and data confidentiality can also be provided by means other than encryption (e.g. by physical protection of data).

When designing a secure system, the *scope* of the system and the set of rules governing the security behaviour of the system are of fundamental importance; these are the *security domain* and the *security policy* respectively. A security policy is defined in ISO 7498-2 as 'the set of criteria for the provision of security services'. A security domain can be regarded as the scope of a single security policy. It is possible to have nested or overlapping security domains, and thus nested or overlapping scopes for security policies.

ISO 7498-2 gives the following statement as an example of a possible generic security policy statement regarding *authorisation*:

> Information may not be given to, accessed by, or permitted to be inferred by, nor may any resource be used by, those not appropriately authorised.

An initial generic policy of this type can then be refined, in conjunction with the results of a requirements analysis, into a detailed set of rules governing the operation and management of the system. Note that this generic policy only deals with preventing unauthorised access, i.e. it does not make any statement about guaranteeing access to legitimate users. Thus it does not deal with *Availability*, and hence does not address denial of service threats.

ISO 7498-2 distinguishes between two types of security policy: *identity-based* and *rule-based*, depending on how authorisation is granted. Identity-based policies authorise system access on the basis of the identity of the client and the identity of the resource which the client wishes to make use of. Rule-based policies rely on global rules imposed on all users, with access decisions typically made using a comparison of the sensitivity of the resources with the user attributes (e.g. the 'clearance' of the user).

3.2 Security services

ISO 7498-2 defines five main categories of security service:

- *authentication*, including entity authentication and origin authentication,
- *access control*,
- *data confidentiality*,
- *data integrity*,
- *non-repudiation*.

Parts 2-6 of the 7-part *Security framework* standard (ISO, 1996a) give a much more detailed discussion of the general ways in which these services can be provided.

- *Entity authentication* provides corroboration to one entity that another entity is as claimed. This service may be used at the establishment of (or during) a connection, to confirm the identities of one or more of the connected entities. This service provides confidence, *at the time of usage only*, that an entity is not attempting a masquerade or an unauthorised replay of a previous connection.

  Origin authentication provides corroboration to an entity that the source of received data is as claimed. However, the service does *not*, in itself, provide protection against duplication or modification of data units.

- The *access control* service provides protection against unauthorised use of resources. This protection may be applied to various types of access to a resource, e.g. the use of a communications resource, the reading, writing, or deletion of an information resource, the execution of a processing resource.
- ISO 7498-2 defines four types of *data confidentiality* service; all these services provide for the protection of data against unauthorised disclosure. The four types are *Connection confidentiality* – which provides for the confidentiality of all user data transferred using a connection, *Connectionless confidentiality* – which provides for the confidentiality of all user data transferred in a single connectionless data unit, i.e. a packet, *Selective field confidentiality* – which provides for the confidentiality of selected fields within user data transferred in either a connection or a single connectionless data unit, and *Traffic flow confidentiality* – which provides for the confidentiality of information which might be derived from observation of traffic flows.
- ISO 7498-2 defines five types of *data integrity* service; all these services counter active threats to the validity of transferred data. The five types are *Connection integrity with recovery* – which provides for the integrity of all user data on a connection, and detects any modification, insertion, deletion or replay of data within an entire data unit sequence, with recovery attempted, *Connection integrity without recovery* – as previously but with no recovery attempted, *Selective field connection integrity* – which provides for the integrity of selected fields within the user data of a data unit transferred over a connection, *Connectionless integrity* – which provides integrity assurance to the recipient of a data unit, and *Selective field connectionless integrity* – which provides for the integrity of selective fields within a single connectionless data unit.
- ISO 7498-2 defines two types of *non-repudiation* service: *Non-repudiation with proof of origin*, where the recipient of data is provided with protection against any subsequent attempt by the sender to falsely deny sending the data, and *Non-repudiation with proof of*

delivery, where the sender of data is protected against any subsequent attempt by the recipient to falsely deny receiving the data.

3.3 Security mechanisms

Security mechanisms exist to provide and support security services. ISO 7498-2 divides mechanisms into two types: *Specific security mechanisms*, i.e. those specific to providing certain security services, and *Pervasive security mechanisms*, i.e. those not specific to the provision of individual security services, including trusted functionality and event detection (we do not discuss these further here).

Eight types of specific security mechanism are listed, namely *Encipherment*, *Digital signature mechanisms*, *Access control mechanisms*, *Data integrity mechanisms*, which include MACs, *Authentication exchange mechanisms*, *Traffic padding mechanisms*, *Routing control mechanisms*, and *Notarisation mechanisms*. We now consider each of these eight classes in a little more detail. This lays the ground work for the detailed consideration of standards for various types of security mechanism.

- *Encipherment* mechanisms, commonly known as encryption or cipher algorithms, can help provide confidentiality of either data or traffic flow information. They also provide the basis for some authentication and key management techniques.

- A *digital signature* mechanism consists of two procedures: a signing procedure, and a verifying procedure. Such mechanisms can be used to provide non-repudiation, origin authentication and/or integrity services, as well as being an integral part of some mechanisms to provide entity authentication. Signature mechanisms can be divided into two types: Digital signatures 'with message recovery', and Digital signatures 'with appendix'. *One-way hash functions* are an essential part of the computation of digital signatures 'with appendix'.

- *Access control* mechanisms can be thought of as a means for using information associated with a client entity and a server entity to decide whether access to the server's resource is granted to the client. Examples of types of access control mechanisms include: access control lists, capabilities and security labels. A general framework for access control mechanisms can be found in ISO/IEC 10181-3, the *Access Control Framework* (ISO, 1996a).

- Two types of *data integrity* mechanism exist: those concerned with the integrity of a single data unit, and those concerned with protecting the integrity of an entire sequence of data units. The first type of mechanism, e.g. a MAC, can be used to help provide both data origin authentication and data integrity (as well as being an integral part of some authentication exchange and key management mechanisms). Mechanisms of the second type, which must be used in conjunction with mechanisms of the first type, can be used to provide full connection-oriented integrity services. These mechanisms include sequence numbers and time stamps. These mechanisms are necessary since use of a MAC alone will not enable a recipient of data to detect replays of single data units, and, more generally, manipulation of a sequence of data units (including replay, selective deletion and re-ordering).
- *Authentication exchange* mechanisms, otherwise known as authentication protocols, can be used to provide entity authentication (as well as being the basis of some key management mechanisms).
- The term *traffic padding* describes the addition of 'bogus' data to conceal the volumes of real data traffic. It can be used to help provide traffic flow confidentiality. This mechanism can only be effective if the added padding is enciphered (or otherwise provided with confidentiality).
- *Routing control* mechanisms can be used to prevent sensitive data using insecure communications paths. For example, depending on the data's sensitivity, routes can be chosen to use only secure network components (sub-networks, relays or links). Data carrying certain security labels may be forbidden to enter certain network components.

- The integrity, origin and/or destination of transferred data can be guaranteed by the use of a *notarisation* mechanism. A third party *notary*, which must be trusted by the communicating entities, will provide the guarantee (typically by applying a cryptographic transformation to the transferred data).

4 Encryption algorithms

In the late 1970s the DES block cipher algorithm was adopted by the NBS as a U.S. Federal Standard (FIPS, 1993); it was subsequently made into a U.S. Standard by ANSI (ANSI, 1981). Efforts to make DES an ISO standard nearly succeeded in the mid 1980s, but work was stopped for political reasons (work also stopped on efforts to standardise RSA). Instead, all efforts to standardise encryption techniques were abandoned, and work focussed instead on creating an international register of algorithms. The form of register entries is standardised in ISO/IEC 9979 (ISO, 1999e) and the register itself is actually held by NCC, Manchester. The register enables communicating entities to identify and negotiate an agreed algorithm.

The Registration Authority maintains the register and ensures that 'register entries conform to the registration procedures' in ISO/IEC 9979. It 'does not evaluate or make any judgement of quality' of registered algorithms. A registered algorithm may be an algorithm for which a complete description is contained in the register, an algorithm for which a complete description is defined in an ISO document or in a standard maintained by an ISO member body or by a liaison organisation, or an algorithm not completely described in the public domain.

Submission of entries to the register may be originated by an ISO member body, e.g. BSI, AFNOR, ANSI, DIN, etc., an ISO Technical Committee, or a liaison organisation. For each registered algorithm the corresponding register entry must contain the following details.

a. Formal algorithm name.

b. Proprietary name(s) of algorithm.

c. Intended range of applications.

d. Cryptographic interface parameters.

e. Set of test values.

f. Organisation identity that requested registration.

g. Dates of registration and modifications.

h. Whether the algorithm is the subject of a national standard.

i. Patent licence restriction information.

It may also, optionally, contain the following information.

j. List of references to associated algorithms.

k. Algorithm description.

l. Modes of operation.

m. Other information.

5 Modes of operation for block ciphers

Modes of operation for the DES block cipher algorithm were standardised in the U.S. in 1980 (FIPS, 1980) and 1983 (ANSI, 1983). These modes of operation are recommended ways in which to use DES to encipher strings of data bits.

Work initially started within ISO to provide corresponding international standards for DES modes of operation. When the ISO work on DES ceased, the modes of operation work continued, but now directed towards any block cipher algorithm, resulting in two standards: ISO 8372 (ISO, 1987a) (modes of operation for a 64-bit block cipher algorithm) and ISO/IEC 10116 (ISO, 1997c) (modes of operation for an *n*-bit block cipher algorithm for any *n*).

All these standards (NBS, ANSI and ISO) contain four modes of operation:

- *ECB* (Electronic Code Book) Mode,

- *CBC* (Cipher Block Chaining) Mode,
- *OFB* (Output FeedBack) Mode, and
- *CFB* (Ciphertext FeedBack) Mode.

We now describe each of these modes in a little more detail. Note that we base all our descriptions on the text in ISO/IEC 10116 (ISO, 1997c) since ISO 8372 is just a special case of ISO/IEC 10116. Throughout we suppose e is the encryption operation for an n-bit block cipher (where n is the number of bits in a plaintext and a ciphertext block) and d is the decryption operation for the same block cipher. We write

$$C = eK(P)$$

where C is an n-bit ciphertext block, K is a secret key for the block cipher, and P is an n-bit plaintext block. Similarly we write

$$P = dK(C),$$

and hence $P = dK(eK(P))$.

5.1 Electronic Code Book (ECB) Mode

The plaintext must be in the form of a sequence of blocks P_1, P_2, ... , P_q where P_i is an n-bit block. The ciphertext is then defined to be the sequence of blocks C_1, C_2, ... , C_q where

$$C_i = eK(P_i)$$

for every i ($1 \leq i \leq q$). Decipherment is achieved as:

$$P_i = dK(C_i)$$

for every i ($1 \leq i \leq q$).

5.2 Cipher Block Chaining (CBC) Mode

As for ECB mode, the plaintext must be made into a series of n-bit blocks: $P_1, P_2, \ldots, P_q$. In addition let SV be a 'starting variable'. Then compute the sequence of ciphertext blocks $C_1, C_2, \ldots, C_q$, as follows:

$$C_1 = eK(P_1 \oplus SV), \text{ and } C_i = eK(P_i \oplus C_{i-1}) \quad (i>1)$$

where $\oplus$ denotes bit-wise exclusive-or of blocks. Decipherment operates as follows:

$$P_1 = dK(C_1) \oplus SV \text{ and } P_i = dK(C_i) \oplus C_{i-1} \quad (i>1).$$

5.3 Ciphertext FeedBack (CFB) Mode

We start by describing CFB mode as it appeared in the first, 1991, edition of ISO/IEC 10116. To use this mode it is first necessary to choose two parameters:

- k $(1 \le k \le n)$, the size of the *Feedback Variable,*
- j $(1 \le j \le k)$, the size of the *Plaintext Variable*.

Divide the plaintext into a series of j-bit blocks: $P_1, P_2, \ldots, P_q$. Let SV be an n-bit 'starting variable'. We will also use the following variables to denote 'intermediate results':

- $X_1, X_2, \ldots, X_q, Y_1, Y_2, \ldots, Y_q$, each of n bits,
- $E_1, E_2, \ldots, E_q$, each of j bits,
- $F_1, F_2, \ldots, F_{q-1}$, each of k bits.

Given an m-bit block $X = (x_1, x_2, \ldots, x_m)$ and a k-bit block $F = (f_1, f_2, \ldots, f_k)$ (where $k \le m$), we will use the notation $S_k(X|F)$ to denote the m-bit block

$$(x_{k+1}, x_{k+2}, \ldots, x_m, f_1, f_2, \ldots, f_k).$$

The effect is to shift X left by k places, shifting in the k elements of F on the right.

Encipherment operates as follows. First let $X_1=SV$. Then, for $i = 1, 2, \ldots, q$ calculate:

$Y_i = eK(X_i)$

$E_i = Y_i \sim j$

$C_i = P_i \oplus E_i$

$F_i = S_j(I(k)|C_i)$

$X_{i+1} = S_k(X_i|F_i)$

where $Y_i \sim j$ denotes the left-most j bits of Y_i, and $I(k)$ denotes a block of k ones. Note that the last two steps are not performed when $i = q$. Decipherment operates as follows. First let $X_1 = SV$. Then, for $i = 1, 2, ..., q$ calculate:

$Y_i = eK(X_i)$

$E_i = Y_i \sim j$

$P_i = C_i \oplus E_i$

$F_i = S_j(I(k)|C_i)$

$X_{i+1} = S_k(X_i|F_i)$

As for encipherment, the last two steps are not performed when $i = q$.

5.4 Output FeedBack (OFB) Mode

To use this mode it is first necessary to choose j $(1 \leq j \leq n)$, the size of the *Plaintext Variable*.

Divide the plaintext into a series of j-bit blocks: $P_1, P_2, ..., P_q$. Let SV be an n-bit 'starting variable'. We will also use the following variables to denote 'intermediate results':

- $X_1, X_2, ..., X_q, Y_1, Y_2, ..., Y_q$, each of n bits,
- $E_1, E_2, ..., E_q$, each of j bits.

Encipherment operates as follows. First let $X_1 = SV$. Then, for $i = 1, 2, ..., q$ calculate:

$Y_i = eK(X_i)$

$E_i = Y_i \sim j$

$C_i = P_i \oplus E_i$

$X_{i+1} = Y_i$

where $Y_i \sim j$ denotes the left-most j bits of Y_i. Note that the last step is not performed when $i = q$. Decipherment operates as follows. First let $X_1 = SV$. Then, for $i = 1, 2, ..., q$ calculate:

$Y_i = eK(X_i)$

$E_i = Y_i \sim j$

$P_i = C_i \oplus E_i$

$X_{i+1} = Y_i$

As for encipherment, the last step is not performed when $i = q$.

5.5 Padding

All four modes of operation require the plaintext to be ‘padded’ to the right length. Annex A to (ISO, 1997c) describes the following two methods for avoiding message extension for CBC mode. First suppose that the ‘unpadded’ plaintext results in a final block P_q of j bits (where $j < n$).

- Method 1 modifies the encipherment of the last ‘short’ block. The encipherment (and decipherment) methods for this block are as follows:

  $C_q = P_q \oplus (eK(C_{q-1}) \sim \mathrm{j})$

  $P_q = C_q \oplus (eK(C_{q-1}) \sim \mathrm{j})$.

- Method 2 (also known as *Ciphertext Stealing*) modifies the encipherment of the last block as follows:

  $C_q = eK(S_j(C_{q-1}|P_q))$

and the last two ciphertext blocks are then $C_{q\text{-}1} \sim j$ and C_q. It is necessary to decipher the final block C_q before $C_{q\text{-}1}$. Deciphering C_q enables the recovery of the last $n\text{-}j$ bits of $C_{q\text{-}1}$, and then $C_{q\text{-}1}$ can be deciphered.

Method 1 is subject to a possible 'chosen plaintext' attack if the *SV* (starting variable) is not secret or has been used more than once with the same key.

5.6 Generalised CFB Mode

In the 2nd edition of ISO/IEC 10116 (ISO, 1997c) a generalised version of the CFB mode has been included. This method allows 'pipelining' to take place. In the original version of CFB mode, the result of enciphering one block of plaintext is needed as input to the enciphering of the next block, and thus it is impossible to 'pipeline' calculations, i.e. start enciphering one block before the processing of the previous block is complete. To avoid this problem, in the new version of CFB mode an *r*-bit *feedback buffer* (*FB*) is introduced, where $2n \geq r \geq n$. An *r*-bit SV is now needed, and after setting $FB_1 = SV$, the encipherment process becomes:

$X_i = FB_i \sim n,$

$Y_i = eK(X_i),$

$E_i = Y_i \sim j,$

$C_i = P_i \oplus E_i,$

$F_i = S_j(I(k)|C_i),$

$FB_{i+1} = S_k(FB_i|F_i),$

where FB_1, FB_2, ..., FB_q are *r*-bit variables representing the successive contents of the Feedback Buffer. Note also that (ISO, 1997c) recommends choosing $j = k$ for CFB mode.

6 Message Authentication Codes (MACs)

The purpose of a Message Authentication Code (MAC), when applied to a message, is to enable the recipient of that message to check both where it comes from and that it has not been changed in transit. Standards for MACs date back to the early 1980s, when ANSI in the U.S. published MAC standards exclusively for banking use (ANSI, 1986a) and (ANSI, 1986b). The corresponding international banking standard, released by ISO in 1987, is (ISO, 1987b). All these standards specify use of the DES block cipher algorithm in CBC mode to produce what has become known as a CBC-MAC. Further international (banking only) MAC standards are (ISO, 1986) which gives general requirements for such mechanisms, and (ISO, 1992), which standardises a completely different and now discredited mechanism, called the *Message Authenticator Algorithm* (*MAA*).

Following on from this banking work, ISO produced a general purpose MAC standard, ISO/IEC 9797, in 1989. This standard also uses a block cipher in CBC mode, i.e. it specifies a CBC-MAC. Unfortunately, the 1989 version was ambiguously phrased in its description of how padding operates, and a revised version (ISO, 1994a) was published in 1994.

In 1997, a major revision of the ISO/IEC MAC standard commenced. The existing 1994 standard is being replaced by ISO/IEC 9797-1 (ISO, 1999c) containing an enlarged set of CBC-MAC mechanisms. A further part ISO/IEC 9797-2 (ISO, 1998a) is also under development, which contains a series of hash-function based MAC mechanisms, including the HMAC technique.

6.1 CBC-MACs

We start by considering the CBC-MACs defined in the (1994) second edition of ISO/IEC 9797, (ISO, 1994a). We then consider what is added in ISO/IEC 9797-1, and also briefly consider the hash-function based mechanisms in ISO/IEC 9797-2.

ISO/IEC 9797 ‘specifies a method of using a key and an n-bit block cipher algorithm to calculate an m-bit cryptographic check value that can be used as a data integrity mechanism’ to detect unauthorised changes to data. Note that m is user-selectable subject to the constraint $m \leq n$. Essentially the data is processed as follows.

- The data is padded to form a sequence of n-bit blocks.

- The data is enciphered using CBC mode with a secret key.
- The final ciphertext block becomes the MAC, after optional processing and optional truncation (which will only be necessary if $m < n$).

More specifically, if the n-bit data blocks are denoted D_1, D_2, ..., D_q, then the MAC is computed by first setting $I_1 = D_1$ and $O_1 = eK(I_1)$, and then performing the following calculations for $i = 2, 3, ..., q$:

$$I_i = D_i \oplus O_{i-1}$$

$$O_i = eK(I_i)$$

The output O_q from these calculations is then subjected to an ‘optional process’ and finally truncated to m bits to produce the MAC.

6.2 Padding methods

ISO/IEC 9797 specifies two possible padding methods.

- **Method 1**: add as many zeros (possibly none) as are necessary to obtain a data string whose length is an integer multiple of n (old method).
- **Method 2**: add a single one and then as many zeros as are necessary (this method may involve creating an entire extra block).

It is important to note that the padding does not need to be transmitted/stored with the integrity-protected data string. If the length of the data is not reliably known by the verifier then Method 2 should be used, since it allows the detection of malicious addition/deletion of trailing zeros (unlike Method 1, which is retained for backwards compatibility with (ANSI, 1986a) and (ANSI, 1986b).

6.3 An attack on CBC-MACs

Suppose a CBC-MAC is computed with no optional process and no truncation. Then, given two messages with valid MACs (computed using the same secret key K), we can compute a third ‘composite’ bogus message with a valid MAC without knowing the key.

To see how this works we illustrate the attack in the case where MACs are known for two single block messages. Suppose $MAC_1 = eK(D_1)$, and $MAC_2 = eK(D_2)$. Then MAC_2 is a valid MAC on the two block message with first block D_1 and second block: $D_2 \oplus MAC_1$. To avoid such attacks, known sometimes as ‘cut and paste’ attacks, we either need to use one of the optional processes, or use padding method 3 from (ISO, 1999c). Note that even if two MACs are never computed with the same key, this attack still applies, since we can take $D_1=D_2$.

6.4 Optional processes

ISO/IEC 9797 specifies two optional processes which can be applied to the final block O_q obtained from the CBC encipherment of the padded data string.

The two optional processes are as follows (where O_q is the n-bit output from the CBC process and K is the key used with the CBC encipherment).

- **Optional process 1**: choose a key K_1 and compute:

$O_q'' = eK(dK_1(O_q))$.

- **Optional process 2**: choose a key K_1 (which may be derived from K) and compute:

$O_q' = eK_1(O_q)$.

Following the optional process, the resulting n-bit block can be truncated to m bits (if $m < n$). Note that one of the main reasons for using an optional process is to avoid ‘cut and paste’ attacks of the type just described.

6.5 New CBC-MAC methods

The motivation for the new CBC-MAC methods in ISO/IEC 9797-1 is provided by some new attacks on CBC-MACs, described in detail in Annex A of (ISO, 1999c). The enhancements include the following.

- A new (3rd) padding method has been introduced.
- A new algorithm has been introduced with special processing for the first block (as well as the last block). This makes exhaustive key search more difficult.
- Two new 'parallel' variants have been introduced.

The new 'Padding Method 3' operates as follows.

- The data string *D* shall be right-padded with as few (possibly zero) 0 bits as are necessary to obtain a data string whose length is a multiple of *n* bits.
- The resulting string shall be left-padded with a single *n*-bit block *L*, consisting of the binary representation of the length in bits of the unpadded data string *D* (left-padded as necessary with zeros).

In summary, the six MAC algorithms in the new version of ISO/IEC 9797-1 are as follows. Note that the first three algorithms were in the 1994 version of the standard.

- MAC Algorithm 1 is simply CBC-MAC with no optional process.
- MAC Algorithm 2 is CBC-MAC with optional process equal to an additional encryption of the last block.
- MAC Algorithm 3 is CBC-MAC with optional process equal to an extra decryption and encryption. This means that, effectively, the last block is 'triple encrypted'.
- MAC Algorithm 4. In this algorithm the first and last blocks are both 'double encrypted'.
- MAC Algorithm 5 is equal to two parallel instances of MAC algorithm 1 (with different keys). The two outputs are ex-ored together to give the MAC.

- MAC Algorithm 6 is equal to two parallel instances of MAC algorithm 4 (with different keys). The two outputs are ex-ored together to give the MAC.

6.6 MACs from hash-functions

ISO/IEC 9797-2 (ISO, 1998a) contains a total of three different methods for deriving a MAC function from a hash-function. In each case it recommends use of one of the three hash-functions from ISO/IEC 10118-3 (1998c), described below. Thus ISO/IEC 9797-2 defines a total of nine different MAC functions.

Superficially it is possible to derive a MAC from a hash-function by simply concatenating a secret key with the data to be MACed, and then applying the hash. That is we could put

$$\text{MAC} = h(K||D)$$

where h is a hash-function, K is a secret key, and D is the data to be MACed.

This is insecure because of the iterative nature of popular hash-functions. To see why this is the case, we first need to consider what it means for a hash-function to be iterative. Essentially it means that the hash-function is constructed from use of a special type of function called a *round-function*. To compute a hash-code, the data is first divided into blocks. The round-function is then applied repeatedly, and at each application it combines a data block with the previous output of the round-function. The first input to the round-function is a fixed IV, and the last output is the hash-code. This means that, if h is the hash-function, then knowledge of $\mathrm{h}(X)$ for secret X, enables $h(X||Y)$ to be computed for any chosen Y. This is because the hash-code is simply the output of the last iteration of the hash-function. Thus, if the MAC is computed as suggested above, then given a MAC on data string D, a valid MAC can be computed on a data string $D||D'$, where D' is chosen by the attacker.

The three methods described in ISO/IEC 9797-2 are as follows.

- MDx-MAC is the first scheme in ISO/IEC 9797-2. It involves modifying the hash-function in a small way. It only works with the three hash-functions from ISO/IEC

10118-3, all of which involve iterative use of a *round-function* (as described previously, the data string to be hashed is divided into blocks, and the round-function combines a block with the previous round-function output). The round-function of the underlying hash-function is first modified in a key-dependent way. An 'intermediate value' is then obtained by concatenating some key-derived information with the data string to be MACed, and then applying the (modified) hash-function. This intermediate value is then input to the round-function one more time, with the other input being further key-dependent information. The output is the MAC value.

- HMAC, as defined in Internet RFC 2104 (RFC, 1997), is the second scheme included in ISO/IEC 9797-2. The basic idea of the HMAC scheme is to compute

  $$MAC = h(\ K \,||\, h(\ K' \,||\, m\ ))$$

  where h is a hash-function and $K \neq K'$. More specifically K and K' are two variants of a single secret key (and steps are taken to ensure that K and K' are distinct).

- The third scheme in ISO/IEC 9797-2 is a modified version of MDx-MAC applying only to short messages (at most 256 bits). It has been optimised to minimise the amount of computation required.

7 Digital signatures

A digital signature mechanism is a function which, when applied to a message, produces a result which enables the recipient to verify the origin and integrity of a message. Moreover, it has the property that only the originator of the message can produce a valid signature (i.e. being able to verify the correctness of a signature generated by entity *A*, does not provide the means to compute *A*'s signature on another message). Digital signatures can be used to provide non-repudiation of origin for a message, i.e. the recipient of a message with entity *A*'s signature on it has evidence that *A* did originate the message, which even *A* cannot repudiate. This emulates the properties we expect of a conventional signature.

A digital signature mechanism requires every user to have a pair of keys, a *private key* for signing messages (which must be kept secret) and a *public key* for verifying signatures (which is widely distributed).

Signature mechanisms can be divided into two types:

- *Digital signatures with message recovery*, i.e. where all or part of the message can be recovered from the signature itself, and mechanisms of which type are standardised in the multi-part standard ISO/IEC 9796, and
- *Digital signatures with appendix,* i.e. where the entire message needs to be sent or stored with the signature, as covered by the multi-part international standard ISO/IEC 14888.

7.1 Signatures with message recovery

Signatures with message recovery operate in the following general way:

1. the message to be signed is lengthened by the addition of ‘redundancy’ according to an agreed formula, and
2. the lengthened message is then subjected to the signing process.

The verification process reveals the lengthened message, from which the original message can be recovered. Hence, with such a signature scheme, the message is contained in the signature, and thus the message does not need to be sent or stored independently of the signature itself. Because of this property, signatures of this type can only be applied to short messages.

ISO/IEC 9796 currently has three parts:

- ISO/IEC 9796 (ISO, 1991), currently being transformed into ISO/IEC 9796-1. The signature scheme is based on a generalised version of RSA.
- ISO/IEC 9796-2 (ISO, 1997a). This scheme uses the same signature transformation as (ISO, 1991). However the method for adding redundancy is completely different, being

based on use of a hash-function. It also provides for *partial message recovery*, and hence this scheme can be used to sign arbitrarily long messages.

- ISO/IEC FCD 9796-3 (ISO, 1999b). This scheme uses the same redundancy method as (ISO, 1997a), i.e. it allows for partial message recovery, will work with arbitrarily long messages, and is based on a hash-function. However the signature function is different, being based on discrete logarithms rather than being RSA-like.

7.2 ISO/IEC 9796 scheme

7.2.1 Overview

The ISO/IEC 9796 standard for a 'signature with recovery' mechanism operates in the following general way.

Messages to be signed are subject to a sequence of five processes (note that use of the scheme requires choice of a parameter k_s):

1. **Padding**. This ensures the padded message contains a whole number of 8-bit bytes.
2. **Extension**. This ensures the extended message contains the 'correct' number of bytes.
3. **Redundancy adding**. This doubles the length of the extended message by interleaving it with special 'redundancy' bytes.
4. **Truncation and forcing**. This involves discarding a few of the most significant bits (if necessary) of the redundancy-added message to get a string of k_s-1 bits, then prefixing the result with a single 1 (to get a string of k_s bits), and finally changing the least significant byte according to a specified formula. The purpose of this, seemingly rather bizarre, operation on the least significant byte is to prevent certain types of cryptographic attack.
5. **Signature production**. The truncated and forced message is input to a mathematical signature algorithm which operates on strings of k_s bits (e.g. the modified RSA signature scheme described in Annex A of the standard), to obtain the signature on the message.

Although the signature production function is not specified in ISO/IEC 9796, the other four processes (which *are* specified) have been designed specifically for use with the signature production function given in Annex A of the standard, and are probably inappropriate for any other signature production function. The signature production function is in Annex A and not in the body of the standard for political and not technical reasons.

Note that Annex A of ISO/IEC 9796 is informative and not normative (i.e. the RSA-type scheme is not officially part of the standard), although this situation has been changed in ISO/IEC 9796-1, where Annex A has been made normative.

7.2.2 The five signature generation processes

We now examine each of the five processes in a little more detail. We assume that the signature production function operates on strings of k_s bits and produces signatures also containing k_s bits. Because of the various processes applied to the message before it is input to the signature production function, and because of the mathematical properties of the signature production function in Annex A of ISO/IEC 9796-1, this means that messages to be signed must contain a little less than $k_s/2$ bits.

1. The bit string to be signed is first padded with between 0 and 7 zeros at the 'most significant end', to get a whole number (denoted z) of bytes. The *Index r*, is defined to be the number of added zeros plus one (i.e. r will satisfy $1 \leq r \leq 8$). The output of the padding process is denoted *MP* (for *Padded Message*). The following must hold for the signature computation to be possible:

   $$16z \leq k_s+3.$$

2. Define t to be the smallest integer such that $16t \geq k_s$-1 (and hence a string of $2t$ bytes will contain between k_s-1 and k_s+14 bits). The *Extended Message*, *ME,* is now obtained by repeating the z bytes of *MP* as many times as are necessary to get a string with exactly t bytes in it.

3. The third step involves producing a *Redundancy-added Message MR*, which will contain precisely $2t$ bytes. It is obtained by interleaving the t bytes of *ME* (in odd positions) with t bytes of redundancy (in even positions). Hence, if m_1, m_2, ..., m_t are the bytes of *ME*, then Byte $2i$-1 of $MR = m_i$, and Byte $2i$ of $MR = S(m_i)$, for every i ($1 \leq \mathrm{i} \leq t$), where S is a function specified in ISO/IEC 9796. More precisely, ISO/IEC 9796 specifies a permutation Π which acts on 4-bit 'nibbles', and if $m = \mu_2 \| \mu_1$ is a byte, where $\|$ denotes concatenation, then

   $$S(m) = \Pi(\mu_2) \| \Pi(\mu_1).$$

   Finally, byte number $2z$ of *MR* (denoted mr_{2z}) is modified by $mr_{2z} = r \oplus mr_{2z}$ where $\oplus$ denotes bit-wise exclusive-or and r is the index (defined in the padding step).

4. As a result of the fourth step a string *IR* is produced, which will contain exactly k_s bits, from *MR*. This is done by setting the most significant bit to a one, and then setting the other k_s-1 bits to the least significant k_s-1 bits of *MR* (which contains between k_s-1 and k_s+14 bits), i.e. between 0 and 14 bits are discarded. Finally the least significant byte is replaced using the following method. If $\mu_2\|\mu_1$ is the least significant byte of *MR* (where μ_1 and μ_2 are 4-bit 'nibbles'), then the least significant byte of *IR* is set to $\mu_1\|6$.

5. The signature Σ is obtained as a string of k_s bits by applying the signature function to *IR* under the control of the secret signature key. Hence

   $$\Sigma = \mathrm{Sign}(IR)$$

   where 'Sign' is the signature function. As already stated, the function 'Sign' will take as input a string of k_s bits and give as output another string of k_s bits. The details of this function are not specified in the main body of ISO/IEC 9796, but Annex A gives exact details of a function for which the whole process has been designed. This function is based on modular exponentiation.

7.2.3 Key generation

To perform the signature function specified in Annex A of (ISO, 1991) it is first necessary for the signer to generate a key pair. To do this the signer must first choose:

- a *verification exponent* $v>1$,
- two primes p and q, where,
  - if v is odd, then p-1 and q-1 shall be coprime to v (where two integers are *coprime* if they have highest common factor 1), and
  - if v is even then (p-1)/2 and (q-1)/2 shall be coprime to v, and p shall not be congruent to q modulo 8.

The signer's *public modulus* is then $n = pq$. The length of the modulus is denoted by k, and the choice of k also fixes k_s so that $k = k_s+1$. Finally the signer's *secret signature exponent*, denoted s, is then set equal to the least positive integer such that

- $sv \equiv 1 \pmod{\mathrm{lcm}(p\text{-}1,q\text{-}1)}$ if v is odd,
- $sv \equiv 1 \pmod{\mathrm{lcm}(p\text{-}1,q\text{-}1)/2}$ if v is even.

This is equivalent to 'standard' RSA if the exponent v is odd.

7.2.4 The ISO/IEC 9796 Signature Function

The signature function contains two steps. The first converts the 'Intermediate Integer' IR to a 'Representative Element' RR. The second computes the signature Σ from RR.

To compute RR from IR:

- if v is odd: $RR = IR$,
- if v is even and $(IR|n) = +1$: $RR = IR$, and
- if v is even and $(IR|n) = -1$: $RR = IR/2$,

where $(a|n)$ denotes the *Jacobi symbol*, and in this case $(a|n) = (a^{(p-1)/2} \bmod p).(a^{(q-1)/2} \bmod q)$ where $n = pq$.

To compute Σ from RR, compute: $RR^s \bmod n$, $\Sigma = \min(RR^s \bmod n,\ n-(RR^s \bmod n))$.

7.2.5 Signature verification

Signatures to be verified are subject to a sequence of three processes:

1. **Signature opening**. This is essentially the inverse to the signature function (step 5 of the signature production process).
2. **Message recovery**. This step yields the original message.
3. **Redundancy checking**. This final step is present to complete the checks that the signature is correct.

At each of these three steps it is possible that the signature may be rejected as invalid if certain checks fail (in which case there is no point in performing any further processing). The signature opening function is not specified in ISO/IEC 9796, although the other two processes are. Annex A to (ISO, 1991) does contain a precise specification of a signature opening function to go with the signature production function also specified there.

7.2.6 The three signature verification processes

We now examine each of the three processes in a little more detail.

1. The *Signature opening* step involves transforming the signature to be verified Σ into a string *IR*', the *recovered intermediate integer*. Hence

   $IR' = \text{Verif}(\Sigma)$.

   The signature Σ is rejected if *IR*' is not a string of k_s bits with most significant bit one and least significant nibble equal to 6. (If all is correct, then *IR*' should be equal to the string *IR* produced as a result of the fourth step of the signature generation procedure.)

2. The *Message recovery* step involves producing a $2t$-byte string MR' (the *recovered message with redundancy*) from IR'. Firstly the least significant k_s-1 bits of MR' are set to equal the corresponding bits of IR', with the most significant $16t$-k_s+1 bits of MR' being set to zeros. The least significant byte of MR' is now replaced using the following method. If $\mu_4||\mu_3||\mu_1||6$ are the four least significant nibbles of IR' then the least significant byte of MR' is made equal to

   $$\Pi^{-1}(\mu_4)||\mu_1.$$

   If all is correct, then MR' should be equal to the string MR produced as a result of the fourth step of the signature generation procedure, with the possible exception of the most significant $16t$-k_s+1 bits, which in MR' are set to all zeros. A series of t checks are now performed to see which of the even bytes 'match' the odd bytes, i.e. if we label the bytes of MR': $m_{2t}, m_{2t-1}, ..., m_1$, a check is performed for successive values of i ($i = 1, 2, ..., t$) to see whether or not $m_{2i} \oplus S(m_{2i-1}) = 0$. If this equation holds for every i ($1 \leq i \leq t$) then the signature is rejected.

   Let z be the smallest positive integer for which $m_{2z} \oplus S(m_{2z-1}) \neq 0$. Set r equal to the least significant nibble of $m_{2z} \oplus S(m_{2z-1})$. The signature is rejected if $1 \leq r \leq 8$ does not hold. The *Recovered padded message MP'* is then put equal to the z least significant bytes in odd positions in MR'; MP' should now be equal to the padded message MP, produced as a result of the first step of the signature procedure. Finally the message is recovered from MP' by deleting the most significant r-1 bits (the signature is rejected if these deleted bits are not all zeros).

3. As a final step in verifying the signature, the recovered padded message MP' is subjected to the second and third steps of the signature generation process (Extension and Redundancy). The least significant k_s-1 bits of the result are compared with the least significant k_s-1 bits of MR' (generated during the previous step). If they disagree then the signature is rejected.

7.2.7 Concluding remarks

The scheme described in ISO/IEC 9796 can be adapted to produce digital signatures 'with appendix' for messages of arbitrary length. This can be achieved by using a *One-way Collision-free Hash-function*, *h*.

- The *one-way* property means that, given an arbitrary output string y, it is computationally infeasible to *find* a binary string x such that $h(x) = y$ (although many such strings x will typically exist).
- The *collision-free* property means that it is computationally infeasible to find two binary strings x and x' ($x \neq x'$) such that $h(x) = h(x')$, although many such pairs will exist.

A message of arbitrary length, m say, is signed by first computing $h(m)$ and then inputting $h(m)$ to the five-part ISO/IEC 9796 signature process.

7.3 ISO/IEC 9796-2

ISO/IEC 9796-2 was published in 1997 (ISO, 1997a). This mechanism has two main properties:

- The system allows for 'partial message recovery' for messages of arbitrary length. I.e., if the message is sufficiently short then all the message can be recovered from the signature, whereas if the message is too long to 'fit' then part of the message can be recovered from the signature and the rest will need to be conveyed to the verifier by some other means.
- The redundancy scheme of ISO/IEC 9796 is a little 'heavy' in that it requires half of the available space in the signature block to be used for redundancy. Thus, if the signature function is 768-bit RSA, then, with the scheme in (ISO, 1991), only 384 bits are available for conveying data bits. With the scheme specified in ISO/IEC 9796-2, around 600 bits out of the 768 could be available for data; such a gain could be critically important in certain practical applications.

The basic idea of the scheme is as follows.

1. The entire message m to be signed is input to a hash-function h to obtain a hash-code H, i.e. $H = h(m)$.
2. If the message is too long to be totally included in the signature then some portion of the message is selected to be ‘recoverable’ from the signature.
3. A flag bit (called a ‘more-data’ bit) is added to the recoverable portion of the message to indicate whether it is all or part of the message.
4. The recoverable portion of the message, the flag and the hash-value, together with other ‘formatting’ bits including an optional hash-function identifier, are concatenated and input to the signature function to derive the signature Σ.

Like ISO/IEC 9796, no signature function is specified in the body of 9796-2; instead exactly the same RSA-based function is included in an informative (non-normative) annex.

7.4 ISO/IEC 9796-3

The ISO/IEC 9796-3 signature function is based on discrete logarithms (the scheme is known as Nyberg-Rueppel, after the inventors). There are two basic versions of the scheme specified in the standard: one based on the group of integers modulo a prime p, and the other based on working within an elliptic curve group. Both versions make use of the same basic idea, so we describe the first version only.

The following domain parameters must be agreed by any community of users of the scheme:

- two large prime numbers p and q, where q is a factor of p-1,
- an element g of multiplicative order q modulo p, i.e. a number g satisfying $g^q \equiv 1 \pmod{p}$ and $g \neq 1$.

Then a user’s private signature key is a number x, where $1 < x < q$. The corresponding public verification key is: $y = g^x \bmod p$.

To use the scheme a signer needs a method for generating secret random numbers k ($1 < k < q$), one per signature; these numbers must be different and unpredictable for each signature. To generate a signature it is necessary to first compute the integer Π , where $\Pi = g^k \bmod p$, and second compute the integer R, where $R = \Pi + D \bmod q$, and where D is the concatenation of the recoverable part of the message and a hash-code (computed on the entire message) – as in ISO/IEC 9796-2. Finally compute the integer S as $S = k - xR \bmod q$. The signature is then the pair (R, S).

To verify a signature first compute the integer Π', where: $\Pi' = g^S y^R \bmod p$, second compute the value D', where: $D' = R - \Pi' \bmod q$, and third reconstruct the message from the recoverable part (embedded within D') and the non-recoverable part (which must be sent with the signature). The reconstructed message is then used to recompute the hash-code. Finally it is necessary to compare the recomputed hash-code with the value embedded in D'.

7.5 ISO/IEC 14888 – signatures 'with appendix'

7.5.1 Introduction

ISO/IEC 14888 is a multi-part standard containing a variety of 'digital signature with appendix' mechanisms. The three parts are as follows:

- **ISO/IEC 14888-1: 1998**: *General*, (ISO, 1998f),
- **ISO/IEC DIS 14888-2**: *Identity-based mechanisms*, including the Guillou-Quisquater scheme, (ISO, 1998g),
- **ISO/IEC 14888-3: 1998**: *Certificate-based mechanisms*, including NIST's *Digital Signature Algorithm (DSA)* – a version of the El Gamal signature algorithm, (ISO, 1998h).

Since this standard covers a large variety of mechanisms, we will not discuss all parts in detail. All 'signature with appendix' schemes operate roughly in the following way:

1. the message to be signed is input to a collision-free one-way hash-function,
2. the output of the hash-function (the *hash-code*) is subjected to the signing process, and
3. the signed hash-code constitutes the signature (*appendix*).

The verification process needs to take as input both the signature and the message, i.e. the message cannot be recovered from the signature.

ISO/IEC 14888-1 provides a general model for all the signature schemes specified in ISO/IEC 14888 parts 2 and 3. This general model covers both *deterministic* and *randomised* signatures. In a deterministic signature scheme, the signature of a fixed string will always be the same. In a randomised signature scheme, a random number is used as part of signing process. This means that, if the same data string is signed twice, different signatures will result. In such schemes it is always important to ensure that the randomised number is different every time, and that guessing the random number is not possible.

7.5.2 Identity-based mechanisms

ISO/IEC 14888-2 (ISO, 1998g) specifies identity-based signature techniques (with appendix). In identity-based schemes, each entity's public signature verification key is derived from that entity's identity. Thus there is no need for public key certificates. To make such a scheme work a Trusted Third Party (TTP) is needed to generate private keys (users cannot generate their own). Hence in such a scheme the TTP has access to all private keys. This means that such schemes are not suitable in all applications. However, such schemes may be suitable for certain closed domains (e.g. within a large company) where there is a 'natural' TTP.

ISO/IEC 14888-2 contains three different signature schemes, all of which are of the *randomised* type. All three schemes are, in fact, different variants of the *Guillou-Quisquater* signature scheme. We only describe the first ('basic') variant here. The scheme is essentially a variant of RSA, closely analogous to the scheme used in ISO/IEC 9796-1 and 9796-2. The TTP chooses (and makes public):

- the domain verification exponent v, and
- the domain modulus $n = pq$, where p and q are large primes (which the TTP ***does not*** make public), and p-1 and q-1 are both coprime to v.

The TTP calculates (and keeps secret) the key generation exponent d, where d is the multiplicative inverse of v (mod $(p\text{-}1)(q\text{-}1)$). Hence we have that $u^{dv} \bmod n = u$ for all non-zero u. This is just like the key generation process for RSA.

To participate in this scheme, each entity must have unique 'identification data' I (a string of bits). To generate the key pair for a user with identification data I, the TTP computes the user's public verification key y as $y = f(I)$ where f is the redundancy-adding function specified in ISO/IEC 9796-1. The private signature key for this user is then $x = y^{-d} \bmod n$.

To generate a signature, the signer first generates the *randomiser* k, and then computes $\Pi = k^v \bmod n$ (where v and n are the domain parameters). The signer next computes $R = h(\Pi \parallel M)$, where M is the message to be signed, and h is an agreed collision-resistant hash-function. Finally the signer computes $S = k.x^R \bmod n$, (where R is converted from a bit string to an integer), and the signature is the pair (R, S).

To verify a signature, the verifier first computes $\Pi' = y^R.S^v \bmod n$. The verifier next computes $R' = h(\Pi' \parallel M)$, where M is message. Finally the verifier compares R and R'. If they agree then the signature is accepted (otherwise it is rejected).

7.5.3 Certificate-based mechanisms

ISO/IEC 14888-3 (ISO, 1998h) describes two general models for signatures with appendix, one discrete logarithm based, and the other factorisation based. A number of examples of each type of scheme are specified in the standard:

- **Discrete logarithm based schemes**: DSA, Pointcheval-Vaudenay (a DSA variant), ECDSA (Elliptic Curve DSA).
- **Factorisation based schemes**: ISO/IEC 9796 with hash, and ESIGN.

The most important of these is probably DSA, and we now describe this scheme.

7.5.4 The Digital Signature Algorithm

The *Digital Signature Algorithm (DSA)* is a version of the ElGamal signature algorithm, which depends for its security on the *discrete logarithm* problem (just like the Diffie-Hellman key exchange mechanism). DSA was adopted as a U.S. Federal Standard in 1993, in the *Digital Signature Standard (DSS)* (FIPS, 1994). The FIPS standard specifies which hash-function should be used with the DSA algorithm, namely the *Secure Hash Algorithm (SHA-1)*, which is itself specified in a separate U.S. Federal Standard (FIPS, 1995).

The generation of a key pair for the Digital Signature Algorithm is a two-stage process. The first stage corresponds to the selection of a triple of underlying parameters (P, Q, G) which may be common to a group of users. It involves the following steps.

- A parameter l is selected which determines the size of the modulus; l is chosen subject to the constraint that $0 \leq l \leq 8$. This determines the value of the 'modulus length parameter' L, where $L = 512 + 64l$.
- A prime P is selected, where $2^{L-1} < P < 2^L$, i.e. P has L bits in its binary representation. The prime P is chosen in such a way that P-1 possesses a prime factor Q, where $2^{159} < Q < 2^{160}$, i.e. Q has 160 bits in its binary representation. An algorithm for generating P and Q is specified in FIPS 186.
- Choose a number G ($1 < G < P$-1) of multiplicative order Q (when working modulo P). To do this choose a random T ($1 < T < P$-1) and check that $T^{(P-1)/Q} \neq 1$. If this check fails then choose another T, and repeat as necessary. Finally put $G = T^{(P-1)/Q}$.

The triple (P, Q, G) is made public, and could be common for a group of users. The second stage involves selecting the private/public key pair.

- The private signature key X is randomly chosen, where $0 < X < Q$.
- The public verification key Y is calculated using $Y = G^X \bmod P$.

The signature for the message M is calculated using the following steps.

- M is subjected to the specified hash-function (SHA-1) which we denote by h, i.e. $h(M)$ is computed. For the purposes of this signature scheme $h(M)$ needs to be treated as an integer; a rule for converting the bit string $h(M)$ into an integer is given in FIPS 186.
- A random value K is selected, where $0 < K < Q$. A different and unpredictable value of K must be chosen for every signature computed (note that DSA is a *randomised* signature scheme).
- A value R is computed, where $R = (G^K \bmod P) \bmod Q$.
- A value S is computed where $S = (K^{-1}(h(M) + XR)) \bmod Q$. Note that K^{-1} is the inverse of K modulo Q. The signature on the message M is the pair (R, S), which contains only 320 bits (since R and S are both 160 bits long).

The verification process takes as input the message M, and the signature pair (R, S). The following steps are performed.

- The verifier first checks that $0 < R < Q$ and $0 < S < Q$; if not then the signature is rejected.
- The verifier next computes:

$$W = S^{-1} \bmod Q,$$

$$U1 = h(M)W \bmod Q,$$

$$U2 = RW \bmod Q, \text{ and}$$

$$V = (G^{U1}Y^{U2} \bmod P) \bmod Q.$$

If $V = R$ then the signature is verified; if not then the signature is rejected. Note that signing the message M using DSA involves calculating the two values:

$$R = (G^K \bmod P) \bmod Q, \text{ and}$$

$$S = (K^{-1}(h(M) + XR)) \bmod Q,$$

where K is a random value. Given that K is message-independent, it can be selected in advance. Moreover, R is a function only of K, i.e. it is message-independent, and thus R can also be pre-computed, as can K^{-1} and XR. Thus signing can be made very fast, at least in situations where pre-computations can be performed. All that is required to compute a signature is to hash the message (i.e. compute $h(M)$), add $h(M)$ to XR (mod Q), and multiply the result of the previous step by K^{-1} (mod Q).

However, verification includes calculating two exponentiations mod P, where both exponents will be 160 bits long. This is a non-trivial calculation. This is reverse of situation for RSA, where use of low exponent for public key can make verification very simple.

8 Hash-functions

8.1 Introduction

One-way hash functions form an integral part of any digital signature with appendix. However, ISO/IEC 14888 not specify any particular hash function – the choice is left to the user. A separate multi-part standard, ISO/IEC 10118, specifying one-way hash functions has been developed. Such cryptographic hash functions also have uses for file protection and data integrity purposes.

- **ISO/IEC 10118-1**, *General* (ISO, 1994c) provides general definitions and background for the other parts of the standard.
- **ISO/IEC 10118-2**, *Hash-functions using an n-bit block cipher algorithm* (ISO, 1994d) describes two methods for deriving a hash function from an *n*-bit block cipher.
- **ISO/IEC 10118-3**, *Dedicated hash-functions* (ISO, 1998c) describes three hash-functions designed specifically for the purpose (namely SHA-1, RIPEMD-128 and RIPEMD-160).
- **ISO/IEC 10118-4**, *Hash-functions using modular arithmetic* (ISO, 1998d) describes two hash-functions (MASH-1 and 2) using modular exponentiation to construct a hash-value.

All the hash-functions specified in ISO/IEC 10118 parts 2, 3 and 4 conform to the same general model (a simplified version of which is given in ISO/IEC 10118-3). The model requires choice of two parameters m (the ‘block length’) and s (the length of the ‘iteration value’, which determines the maximum possible length for the derived hash-code), the choice of an s-bit Initialising Value (IV), the choice of the length for the hash-code L_H (where $L_H \leq s$), and the use of a *round-function* ϕ which takes as input two strings (of lengths m and s bits), and gives as output an s-bit string. Hence if X is an m-bit string and Y is an s-bit string then $\phi(X,Y)$ is an s-bit string.

The model involves four steps in the processing of a data string D.

1. *Padding*. D is padded to ensure that its length is a multiple of m bits.
2. *Splitting*. The padded version of D is split into m-bit blocks $D_1, D_2, ..., D_q$.
3. *Iteration*. The s-bit blocks $H_1, H_2, ..., H_q$ are calculated iteratively in the following way:

   $$H_i = \phi\ (D_i, H_{i-1})$$

   where $H_0 = IV$.
4. *Truncation*. The hash-code H is derived by taking L_H of the s bits from H_q.

8.2 Block cipher based hash-functions

ISO/IEC 10118-2 contains two methods for deriving a hash-function from an n-bit block cipher. Method 1 produces hash-codes of length L_H bits, where $L_H \leq n$. Method 2 produces hash-codes of length L_H bits, where $L_H \leq 2n$. The padding techniques for these two methods are not specified in ISO/IEC 10118-2, although examples are given in an annex to the standard.

8.2.1 Method 1 (single length hash-codes)

For Method 1, the block length (m) is equal to n, the plaintext/ciphertext length for the block cipher. Hence the data string to be hashed is padded and split into a sequence of n-bit blocks

$D_1, D_2, ..., D_q$.

The parameter s is also set to n, the block cipher plaintext/ciphertext length. If encipherment of block M using key K is denoted $e_K(M)$, then the round-function ϕ is defined so that

$$\phi(X,Y) = e_{u(Y)}(X) \oplus X$$

where u is a function which maps n-bit blocks into blocks suitable for use as keys in the chosen block cipher. Hence $H_i = \phi(D_i, H_{i-1}) = e_U(D_i) \oplus D_i$ where $U = u(H_{i-1})$. The truncation function involves taking the left-most L_H bits of H_q.

8.2.2 Method 2 (double length hash-codes)

To define the round-function for ISO/IEC 10118-2 Method 2, we first need to define three special functions.

- $\boldsymbol{I}$, which takes as input a $2n$-bit block and gives as output a $2n$-bit block. Suppose n is even and $X = X_1 \,||\, X_2 \,||\, X_3 \,||\, X_4$ is a $2n$-bit block, where X_i (i = 1, 2, 3, 4) are $n/2$ bit sub-blocks. Then $\boldsymbol{I}(X) = X_1 \,||\, X_4 \,||\, X_3 \,||\, X_2$, i.e. sub-blocks X_2 and X_4 are interchanged.
- $\boldsymbol{L}$, which takes as input a $2n$-bit block and gives as output an n-bit block containing the n left-most bits of the input.
- $\boldsymbol{R}$, which takes as input a $2n$-bit block and gives as output an n-bit block containing the n right-most bits of the input.

For Method 2 we have $m = n$ (as for Method 1), and hence the data string to be hashed is padded and split into a sequence of n-bit blocks $D_1, D_2, ..., D_q$. We set s to $2n$, i.e. twice the block cipher plaintext/ciphertext length. The round-function ϕ is now defined so that

$$\phi(X,Y) = \boldsymbol{I}(\ e_{u(\boldsymbol{L}(Y))}(X) \oplus X \,||\, e_{u'(\boldsymbol{R}(Y))}(X) \oplus X\ )$$

where u and u' are functions which map n-bit blocks into blocks suitable for use as keys in the chosen block cipher. Hence

$$H_i = \phi(D_i, H_{i-1}) = \boldsymbol{I}(\ e_{u(L(i))}(D_i) \oplus D_i \,||\, e_{u'(R(i))}(D_i) \oplus D_i\ ).$$

where $L(i) = \boldsymbol{L}(H_{i\text{-}1})$ and $R(i) = \boldsymbol{R}(H_{i\text{-}1})$.

In Annex A to ISO/IEC 10118-2, choices for the Initialising Value (*IV*) and transformation *u* are suggested which are appropriate for method 1 when the block cipher in use is DES. In the same annex, choices for the Initialising Value (*IV*) and transformations *u*, *u*' are suggested which are appropriate for method 2 when the block cipher in use is DES. Worked examples of these choices are given in a further annex.

8.3 Dedicated hash-functions

ISO/IEC 10118-3 (*Dedicated hash functions*) contains three functions specifically designed for use as hash-functions. In all cases the Initialising Values are specified in the standard, as are the padding methods. Two of the hash-functions, *Dedicated Hash-functions 1* and *2*, are identical to RIPEMD-128 and RIPEMD-160 respectively, European algorithms developed as part of the EC-funded RIPE project. In the first case the round-function has m = 512 and s = 128, i.e. it can generate hash-codes of up to 128 bits in length, and in the second case the round-function has m = 512 and s = 160, i.e. it can generate hash-codes of length up to 160 bits. The third function, *Dedicated Hash-function 3*, is NIST's *Secure Hash Algorithm* (*SHA-1*), already a U.S. Federal Standard (FIPS, 1995). In this case, like RIPEMD-160, the round-function has m = 512 and s = 160, i.e. it can generate hash-codes of up to 160 bits in length.

8.4 Modular arithmetic based hash-functions

ISO/IEC 10118-4 (*Modular arithmetic based hash-functions*) contains a pair of hash-functions, *MASH*-1 and *MASH*-2, based on modular exponentiation. They are improved variants of the function given in the original 1988 version of X.509 (CCITT, 1988) which was found to be prone to attack and hence it has been removed from later versions of the standard.

The initialising values and padding methods for these functions are specified in the standard. The values of m and s will depend on the modulus for the arithmetic operations. The roundfunctions for both versions of MASH are based on exponentiation using a fixed exponent; this fixed exponent is 2 for MASH-1 and 257 for MASH-2.

9 Entity authentication

9.1 Introduction

Authentication forms the basis of the provision of other security services in the majority of network security systems. The OSI Security Architecture (ISO 7498-2) distinguishes between *data origin authentication* (i.e. verifying the origin of received data - a connectionless operation), and *(peer) entity authentication* (i.e. verifying the identity of one entity by another - a connection-oriented operation).

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 term 'protocol' should strictly be 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. However, here we abuse the OSI notation slightly and follow generally accepted practice and call them authentication protocols.

ISO/IEC JTC1/SC27 has produced a multi-part standard, ISO/IEC 9798, specifying a general-purpose set of authentication protocols. The five parts published so far are as follows.

- ISO/IEC 9798-1 – General model, (ISO, 1997b).
- ISO/IEC 9798-2 – Protocols based on symmetric encipherment, (ISO, 1994b).
- ISO/IEC 9798-3 – Protocols based on digital signatures, (ISO, 1998b).
- ISO/IEC 9798-4 – Protocols based on data integrity mechanisms, (ISO, 1995).
- ISO/IEC 9798-5 – Zero knowledge protocols, (ISO, 1999d).

The protocols specified in these standards have been specified 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).

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 *mutual authentication*. Unilateral authentication is 'entity authentication which provides one entity with assurance of the other's identity but not vice versa' and Mutual authentication is '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 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, and so one application of entity authentication is 'authenticated session key establishment'. Other applications exists which are not directly related to session key exchange, including secure clock synchronisation, secure RPC (remote procedure call), and secure transactions.

9.2 Mechanisms underlying authentication protocols

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.

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), and digital signature. The use of MACs and digital signatures and MACs for integrity protection of messages is standard practice; however the use of encipherment for this purpose is much less straightforward, and hence we discuss this a little more before proceeding.

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 presence of this redundancy can then be automatically checked by the recipient of the message (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.

9.3 Classifying authentication 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. However, it is also possible to classify authentication protocols by the 'freshness checking' mechanism they use. 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. 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.

9.3.1 Timestamp-based protocols

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, since the clock drift of a typical work-station can be 1-2 seconds/day.

Every entity receiving protocol messages will need to define a time acceptance 'window' 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 which 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 N_{AB} is incremented by *A*. 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, 1978).

9.3.2 Nonce-based protocols

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.

9.4 Example protocols

We now consider a variety of examples of authentication protocols taken from parts 2, 3 and 4 of ISO/IEC 9798. We give examples based on both types of freshness mechanism.

9.4.1 A unilateral authentication protocol using timestamps and encipherment

The first example can be found in clause 5.1.1 of ISO/IEC 9798-2. It is based on the use of time-stamps (for freshness) and encipherment (for origin and integrity checking). It provides *unilateral authentication* (*B* can check *A*’s identity, but not vice versa). In the message description (here and subsequently) we use the following notation:

- $x \| y$ denotes the concatenation of data items x and y,

- Text1 and Text2 are data strings, whose use will depend on the application of the protocol,
- K_{AB} denotes a secret key shared by *A* and *B*,
- eK_{AB} denotes encryption using the shared secret key K_{AB}, and
- T_A denotes a time-stamp (or sequence number) generated by *A*.

The mechanism has one message pass, as follows:

$A \rightarrow B$: Text2 || e_{KAB}(T_A || B || Text1)

When *B* receives the message from *A*, *B* deciphers the enciphered string, and checks that the deciphered message 'makes sense' (has the appropriate redundancy), that the time-stamp is within its current window (and, using its 'log', that a similar message has not recently been received), and that *B*'s name is correctly included. If all three checks are correct, then *B* accepts *A* as valid. Use of the data strings 'Text1' and 'Text2' will depend on the application domain ('Text1' might, for example, be used for session key transfer). Either or both of these strings may be omitted.

9.4.2 A unilateral authentication protocol using nonces and MACs

This example can be found in clause 5.1.2 of ISO/IEC 9798-4. It is based on the use of nonces (for freshness) and a data integrity mechanism (for origin and integrity checking). It provides *unilateral authentication* (*B* can check *A*'s identity, but not vice versa). In the message descriptions we use the following notation (in addition to that defined for the first example):

- Text1, Text2 and Text3 are data strings, whose use will depend on the application of the protocol,
- fK_{AB} denotes a cryptographic check value (the output of a data integrity mechanism) computed using the shared secret key K_{AB},

- R_B denotes a random nonce generated by *B*.

The mechanism has two message passes, as follows:

$B \rightarrow A$: R_B || Text1

$A \rightarrow B$: Text3 || f_{KAB}(R_B || B || Text2)

When *B* sends the first message, *B* stores the nonce R_B. When *B* receives the second message, *B* first assembles the string R_B||B||Text2, then computes fK_{AB}(R_B||B||Text2) using the shared secret K_{AB}, and finally checks that the newly computed value agrees with the one in the message. If the check is correct, then *B* accepts *A* as valid. Note that, in order for *B* to perform the desired check, *B* must have the means to obtain the data string 'Text2'. One possibility is that Text3 contains a copy of Text2, perhaps in an enciphered form.

9.4.3 A mutual authentication protocol using nonces and encipherment

This example can be found in clause 5.2.2 of ISO/IEC 9798-2. It is based on the use of nonces (for freshness) and encipherment (for origin and integrity checking). It provides *mutual authentication* (*B* can check *A*'s identity and vice versa). In the message descriptions we use the following notation (in addition to that defined for previous examples):

- Text1-Text5 are data strings, whose use will depend on the application of the protocol,
- R_A and R_B denote random nonces generated by *A* and *B* respectively.

The mechanism has three message passes, as follows:

$B \rightarrow A$: R_B || Text1

$A \rightarrow B$: Text3 || e_{KAB}(R_A || R_B || B || Text2)

$B \rightarrow A$: Text5 || e_{KAB}(R_B || R_A || Text4)

When *B* sends the first message, *B* stores the nonce R_B. When *A* sends the second message, *A* stores the nonces R_A and R_B. When *B* receives the third message, *B* deciphers the enciphered string and checks that the deciphered message 'makes sense' (has the appropriate redundancy), that the nonce it includes is the one B sent in the first message, and that B's name is correctly included. If all checks are correct, then B accepts A as valid, and sends the third message. When A receives the third message, A deciphers the enciphered string and checks that the deciphered message 'makes sense' (has the appropriate redundancy), and that the nonces it includes are the expected ones. If both checks are correct, then A accepts B as valid.

9.4.4 A mutual authentication protocol using timestamps and MACs

This example can be found in clause 5.2.1 of ISO/IEC 9798-4. It is based on the use of time-stamps (for freshness) and an integrity mechanism (for origin and integrity checking). It provides *mutual authentication* (B can check A's identity and vice versa). In the message descriptions we use the following notation (in addition to that defined for previous examples):

- Text1-Text4 are data strings, whose use will depend on the application,
- T_A and T_B denote time-stamps (or sequence numbers) generated by A and B respectively.

The mechanism has two message passes, as follows:

$$A \rightarrow B\text{:}\ \ T_A \,||\, \text{Text2} \,||\, f_{KAB}(\ T_A \,||\, B \,||\, \text{Text1})$$

$$B \rightarrow A\text{:}\ \ T_B \,||\, \text{Text4} \,||\, f_{KAB}(\ T_B \,||\, A \,||\, \text{Text3})$$

When B receives the first message, B first assembles the string $T_A||B||\text{Text1}$ and then computes $fK_{AB}(T_A||B||\text{Text1})$, using the shared secret K_{AB}. B checks that the time-stamp T_A is within its current window (and, using its 'log', that a similar message has not recently been received), and that the newly computed check value agrees with the one in the message. If the checks are correct, then B accepts A as valid and sends the second message. When A receives it, A first assembles the string $T_B||A||\text{Text3}$ and then computes $fK_{AB}(T_B||A||\text{Text3})$, using the shared secret K_{AB}. A checks that the time-stamp T_B is within its current window (and, using its 'log', that a similar message has not recently been received), and that the newly computed check value agrees with the one in the message. If the checks are correct, then A accepts B as valid.

Note that, in order for A and B to perform their checks, A and B must have the means to obtain the data strings Text3 and Text1 respectively. One possibility is that Text4 (Text2) contains a copy of Text3 (Text1), perhaps in enciphered form.

9.4.5 A mutual authentication protocol using timestamps and signatures

This example can be found in clause 5.2.1 of ISO/IEC 9798-3. It is based on the use of time-stamps (for freshness) and digital signature (for origin and integrity checking). It provides *mutual authentication* (B can check A's identity and vice versa). In the message descriptions we use the following notation (in addition to that defined for previous examples):

- S_A and S_B are the private signature keys of A and B respectively.
- sS_A denotes the signature function computed using private key S_A.

The mechanism has two message passes, as follows:

$$A \rightarrow B\text{:}\ \ T_A \,\|\, B \,\|\, \text{Text2} \,\|\, sS_A(\, T_A \,\|\, B \,\|\, \text{Text1})$$

$$B \rightarrow A\text{:}\ \ T_B \,\|\, A \,\|\, \text{Text4} \,\|\, sS_B(\, T_B \,\|\, A \,\|\, \text{Text3})$$

When B receives the first message, B first checks that the time-stamp T_A is within its current window (and, using its 'log', that a similar message has not recently been received). B then assembles the string $T_A\|B\|$Text1 and checks that the signature is a valid signature on this string, using a copy of A's public verification key. If the checks are correct, then B accepts A as valid and sends the second message. When A receives it, A first checks that T_B is within its current window (and, using its 'log', that a similar message has not recently been received), and then assembles the string $T_B\|A\|$Text3 and checks that the signature is a valid signature on this string. If the checks are correct, then A accepts B as valid.

Note that, in order for A and B to perform their checks, A and B must have the means to obtain the data strings Text3 and Text1 respectively. One possibility is that Text4 (Text2) contains a copy of Text3 (Text1), perhaps in enciphered form.

9.4.6 A mutual authentication protocol using nonces and signatures

This example can be found in clause 5.2.2 of ISO/IEC 9798-3. It is based on the use of nonces (for freshness) and digital signature (for origin and integrity checking). It provides *mutual authentication* (*B* can check *A*'s identity and vice versa). We use identical notation to the previous examples. The mechanism has three message passes, as follows:

$B \to A$: $R_B \,||\, \text{Text1}$

$A \to B$: $R_A \,||\, R_B \,||\, B \,||\text{Text3} \,||\, sS_A(R_A \,||\, R_B \,||\, B \,||\, \text{Text2})$

$B \to A$: $R_B \,||\, R_A \,||\, A \,||\text{Text5} \,||\, sS_B(R_B \,||\, R_A \,||\, A \,||\, \text{Text4})$

When *B* sends the first message, *B* stores the nonce R_B. When *A* sends the second message, *A* stores the nonces R_A and R_B. When *B* receives the second message, *B* first assembles the string $R_A||R_B||B||$Text2, and then checks that the signature is a valid signature on this string (using a copy of *A*'s public verification key). If the check is correct, then *B* accepts *A* as valid and sends the third message. When *A* receives it, *A* assembles the string $R_B||R_A||A||$Text4 and checks that the signature is a valid signature on this string. If the check is correct, then *A* accepts *B* as valid.

Note that, in order for *A* and *B* to perform their checks, *A* and *B* must have the means to obtain the data strings Text4 and Text2 respectively. One possibility is that Text5 (Text3) contains a copy of Text4 (Text2), perhaps in enciphered form.

9.5 Comparing different approaches

We now briefly consider the relative merits of time-stamps and nonces for freshness checking. Time-stamps have the following advantages with respect to nonces:

- time-stamp based protocols typically contain less messages then nonce-based protocols (typically one less),
- time-stamp based protocols fit well to the client-server model of computing (e.g. RPC).

The main disadvantages of time-stamp based protocols are as follows:

- there is a need to maintain either synchronised clocks (and a log of recently received messages) or sequence number pairs (if logical time-stamps are used),
- problems arise in securely linking the messages of the protocol together.

The need for this latter property depends on the application of the authentication protocol. If the protocol is used for time synchronisation, or database query protection, then linking of a 'request' message to a 'response' message is needed (to prevent a malicious interceptor 'shuffling' responses to requests issued within a short time of one another). To address this problem, time-stamp protocols can use a 'transaction ID' to securely link a request to a reply.

Note that, because of the many problems that have been encountered with authentication protocols in the past, a variety of various 'logics of authentication' have been proposed. The purposes of these logics is to provide a framework to reason formally about the 'soundness' (or otherwise) of candidate protocols. The most celebrated example is the *BAN Logic* (names after its inventors: Burrows, Abadi and Needham). The BAN logic actually makes it possible to reason about one particular application of authentication, namely key distribution.

9.6 Keying requirements for authentication protocols

As we have already noted, almost all authentication protocols use either shared secret keys, or public/private key pairs (for digital signatures). More specifically, protocols based on symmetric cryptography (either 'symmetric' encipherment or data integrity mechanism) make use of a shared secret between *A* and *B*. Digital signature based protocols need *A* and *B* to have a trusted copy of each other's verification key.

We start by considering the keying requirements for symmetric (secret key) cryptography based protocols, i.e. where *A* and *B* need to share a secret key. Of course, if *A* and *B* already share a secret key, then there is no problem. We therefore suppose that *A* and *B* want to engage in an authentication protocol but they do not yet share a secret key. To provide the required shared secret key we assume that there is a trusted third party (TTP) with whom both

A and *B* share a secret. The (on-line) TTP co-operates to enable *A* and *B* to authenticate one another. This process requires more elaborate protocols. Two examples of such protocols can be found in ISO/IEC 9798-2 (ISO, 1994b) although we do not explore them further here. Further examples are provided in ISO/IEC 11770-2 (ISO, 1996c).

When using public key cryptographic techniques such as digital signatures, there is a need for a means to distribute *trusted* copies of user public keys instead of shared secrets. Public verification keys can be *certified* by applying the digital signature of a Trusted Third Party (TTP). The result (i.e. a public key, an entity name, an expiry date, and the signature of a TTP on these three items) is called a public key *certificate*. In order to obtain a verified copy of a user's public key, one first obtains a copy of their public key certificate. To verify a certificate signed by a TTP requires a trusted copy of TTP's public verification key (this could typically be obtained by a user at the time the user's own certificate is generated).

If two entities have certificates signed by different TTPs, then a *cross-certificate* is needed (i.e. one a copy of one TTP's public verification key signed by the other TTP). This leads to the notion of *certification paths*, i.e. sequences of cross-certificates with the subject of one certificate being the signer of the next certificate in the sequence.

9.7 Applications

One very important application of authentication protocols is during connection establishment. An authentication protocol can be used to set up session key(s) to protect data which is transferred during the lifetime of the connection. Keys can be transferred by inclusion in the data string elements of protocol messages. Parts 2 and 3 of the key management standard, ISO/IEC 11770 (ISO, 1996c) and (ISO, 1999f) contain examples of how this can be achieved.

10 Non-repudiation

10.1 Introduction

A multi-part standard (ISO/IEC 13888) on mechanisms for the provision of non-repudiation services has recently been completed. This overlaps to some extent with the work on digital signatures, since digital signatures can be used to provide non-repudiation services.

The non-repudiation standards seek a rather wider scope, with ISO/IEC 13888-1 (ISO, 1997d) giving a general model for the provision of non-repudiation, including a discussion of the role of the trusted third party. ISO/IEC 13888-2 (ISO, 1998e) discusses the provision of non-repudiation services using symmetric cryptographic techniques. Such schemes require the on-line involvement of a trusted third party or *Notary*. ISO/IEC 13888-3 (ISO, 1997e) covering asymmetric cryptography, is concerned with how digital signature techniques can be used to provide these types of service.

10.2 ISO/IEC 13888-1

ISO/IEC 13888-1 (ISO, 1997d) provides a high-level discussion of the ways in which non-repudiation services can be provided. Amongst other topics, the roles of TTPs and the use of tokens are discussed. A total of eight non-repudiation services are defined; amongst them are the following four services which are likely to be the most important.

- *Non-repudiation of origin*, protects against the message originator falsely denying having sent the message.
- *Non-repudiation of submission*, protects against a message delivery authority falsely denying acceptance of the message from the originator.
- *Non-repudiation of transport*, protects against a message delivery authority falsely denying delivery of the message to the recipient.

- *Non-repudiation of delivery*, protects against the message recipient falsely denying receipt of the message.

Each of these services is provided by giving evidence to the party being given protection. For the above four services, evidence is provided to the message recipient for service 1, and to the message originator for services 2, 3 and 4.

10.3 ISO/IEC 13888-2

ISO/IEC 13888-2 (ISO, 1998e) describes a set of mechanisms for providing a variety of non-repudiation services using a combination of symmetric cryptography and a Trusted Third Party. Possible non-repudiation services covered by these mechanisms include:

- *non-repudiation of origin* - a service which protects against an originator's false denial of being the originator of the message, and
- *non-repudiation of delivery* - a service which protects against a recipient's false denial of having received the message.

We describe one TTP-based mechanism for providing non-repudiation of origin.

Suppose entity A is to send a message m to entity B, and suppose also that A and B both trust a Trusted Third Party TTP. Suppose also that A has identity ID_A, and B has identity ID_B. We also suppose that $f_k(D)$ is a MAC computed on data D using the key k, A and TTP share a secret key a, B and TTP share a secret key b, the TTP possesses a secret key x, h is a hash-function, and z denotes a string of data items including ID_A, ID_B, ID_{TTP}, a timestamp, and $h(m)$.

The mechanism has five message passes, as follows:

1. $A \rightarrow TTP$: $z \,||\, f_a(z)$
2. $TTP \rightarrow A$: $z \,||\, f_x(z) \,||\, f_a(z \,||\, f_x(z))$
3. $A \rightarrow B$: $m \,||\, z \,||\, f_x(z)$

4. $B \rightarrow TTP$: $z \parallel f_x(z) \parallel f_b(z \parallel f_x(z))$

5. $TTP \rightarrow B$: $PON \parallel z \parallel f_x(z) \parallel f_b(\text{ PON} \parallel z \parallel f_x(z))$

where *PON* is one bit (*Positive or Negative*) indicating whether or not the non-repudiation information is valid.

After receiving the final message, *B* retains the string $z \parallel f_x(z)$ as evidence that *A* really did send message *m* to *B*. This evidence can be verified by the TTP at any later stage, using the TTPs' secret key x (and the TTP does not need to retain a record of the transaction).

10.4 ISO/IEC 13888-3

ISO/IEC 13888-3 (ISO, 1997e) describes how to construct and use digitally signed tokens to provide various non-repudiation services. For example, a non-repudiation of delivery token is defined as the recipient's signature on a data string containing the following data items:

$ID_{\text{originator}}$, $ID_{\text{recipient}}$, a timestamp, and a hash of the message.

To provide the non-repudiation of delivery service, the message recipient will be required to provide a non-repudiation of delivery token upon request by the message originator.

An informative annex to ISO/IEC 13888-3 describes the use of a TTP to provide a time-stamping service. Such a service involves a TTP adding a timestamp and its signature to data provided by a requester. This data could be a previously signed non-repudiation token. The use of such a time-stamping service is vital if signatures, and hence non-repudiation tokens, are to have long term validity. The addition of a Trusted Third Party timestamp protects against subsequent revocation and/or expiry of the private key used to sign the non-repudiation token.

11 Key management

11.1 Introduction

We now consider the multi-part ISO/IEC standard concerned with key management, namely ISO/IEC 11770. We divide our discussion into three parts, corresponding to the three parts of ISO/IEC 11770:

- *Part 1* - Key management framework (ISO, 1996b) under which heading we consider basic definitions and concepts,
- *Part 2* - Mechanisms using symmetric techniques (ISO, 1996c) i.e. mechanisms for distributing keys using symmetric cryptographic techniques,
- *Part 3* - Mechanisms using symmetric techniques (ISO, 1999f) i.e. mechanisms for distributing keys (for both symmetric and asymmetric algorithms) using asymmetric cryptography.

The earliest key management standards work was started in the early 1980s by the ANSI banking standards community. It has resulted in a series of important banking key management standards (e.g. X9.17-1985, X9.24, X9.28, X9.30 and X9.31). This work was then taken up by ISO TC68, the banking standards committee for ISO, and has resulted in a series of parallel ISO standards, e.g. ISO 8732 for wholesale key management (based on X9.17), ISO 11568 for retail key management, ISO 11649 (based on X9.28), and ISO 11166 (a multi-part standard covering key management using asymmetric algorithms completed in 1994, and related to X9.30 and X9.31). More recently SC27 has developed a generic key management multi-part standard: ISO/IEC 11770.

The ISO/IEC JTC1/SC27 work has primarily focussed on key establishment mechanisms, although 11770-1 is the *Key Management Framework*, containing general advice and good practice on key management, and which is distantly related to ISO/IEC 10181, the multi-part security frameworks standard (ISO, 1996a). ISO/IEC 11770-2 contains key distribution mechanisms based on the use of symmetric (conventional) cryptography, and ISO/IEC 11770-3 contains key distribution/agreement mechanisms based on asymmetric cryptography.

11.2 Key management framework

ISO/IEC 11770-1 (ISO, 1996b) covers the following main topics.

- A list of definitions relevant to key management.
- Methods for key protection and a definition of the key 'lifecycle'.
- Key management 'concepts', covering: key generation, registration, certification, distribution, installation, storage, derivation, archiving, revocation, de-registration, and destruction.
- Models for key distribution.
- A series of appendices covering: Threats to key management, Key Management Information Objects (an ASN.1 definition for a data structure containing key(s) and associated information), Types of keys, and Certificate lifecycle management.

Some of the most important ISO/IEC 11770 definitions (mostly but not exclusively contained in Part 1) are as follows.

- *certification authority (CA)* – a centre trusted to create and assign public key certificates. Optionally, the CA may create and assign keys to the entities.
- *implicit key authentication to A* – the assurance for one entity *A* that only another identified entity can possibly be in possession of the correct key.
- *key* – a sequence of symbols that controls the operation of a cryptographic transformation.
- *key agreement* – the process of establishing a shared secret key between entities in such a way that neither of them can predetermine the value of that key. [This means that neither entity has key control.]

- *key confirmation* – the assurance for one entity that another identified entity is in possession of the correct key.
- *key control* – the ability to choose the key, or the parameters used in the key computation.
- *key distribution centre (KDC)* – an entity trusted to generate or acquire, and distribute keys to entities that share a key with the KDC.
- *key establishment* – the process of making available a shared secret key to one or more entities. Key establishment includes key agreement and key transport.
- *key translation centre (KTC)* – an entity trusted to translate keys between entities that each share a key with the KTC.
- *key transport* – the process of transferring a key from one entity to another entity, suitably protected.
- *private key* – that key of an entity's asymmetric key pair which should only be used by that entity. [A private key should not normally be disclosed.]
- *public key* – that key of an entity's asymmetric key pair which can be made public.
- *secret key* – a key used with symmetric cryptographic techniques and usable only by a set of specified entities.

Keys are typically organised in *key hierarchies*. Keys in one level of the hierarchy may only be used to protect keys in the next level down in the hierarchy. Only keys in the lowest level of the hierarchy are used directly to provide data security services. This hierarchical approach allows the use of each key to be limited, thus limiting exposure and making attacks more difficult. For example, the compromise of a single session key (i.e. a key at the lowest level of the hierarchy) only compromises the information protected by that key. The key at the top level of the hierarchy is referred to as the *master key*. Disclosure of a master key will potentially enable the possessor to discover or manipulate all other keys protected by it (i.e.

all keys in that particular hierarchy). It is therefore desirable to minimise access to this key, perhaps by arranging that no single user has access to its value.

11.3 Certificate management

Annex D of ISO/IEC 11770-2 (ISO, 1996c) contains a detailed discussion of certificate lifecycle management. This discussion covers the role of the Certification Authority (CA), the ‘certification process’ (covering the relationships between the main entities involved in the generation and management of certificates), distribution and use of certificates, certification paths, and certificate revocation.

A *public key certificate* is a list of data items associated with a particular user, including the public key(s) of that user, all signed by a Certification Authority. Every user will subscribe to a particular CA, and possess a (trusted) copy of the verification key for that CA; they are thus able to verify certificates generated by that CA. Information in a certificate will typically include:

- the name of the user,
- an expiry date (or, more generally, a period of validity),
- a serial number,
- one or more public key(s) belonging to the user,
- the algorithm identifier(s) for the public key(s),
- information regarding the security policy under which this certificate has been created.

Various standards exist for the structure of a certificate. Most important is ITU-T recommendation X.509 (ITU, 1997) and the corresponding ISO/IEC standard: ISO/IEC 9594-8 (ISO, 1999a). Recent revisions to these standards (which enable policy information to be included in these ‘standard’ certificates), have resulted in the ‘Version 3’ X.509 certificate format.

The CA is trusted by its subscribers for the purposes of certificate generation. The CA is responsible for identifying the entities whose public key information is to be incorporated into a certificate, ensuring the quality of the CA’s own key pair used for generating certificates, and securing the certificate generation process and the private key used in the certificate generation process.

One issue of major importance not addressed in ISO/IEC 11770-1 concerns the situation where a user generates his/her own asymmetric key pair, and then requests the CA to generate a certificate for his/her public key. It is generally considered good practice for the CA to ask the user to provide assurance that the user possesses the private key corresponding to the public key offered for signature (e.g. in the case of a signature key by signing a date-stamped statement to this effect, which the CA can then verify using the offered public key). Such a procedure can avoid one user claiming to possess another user’s key pair, with undesirable consequences in certain situations.

Certificates may be revoked before their scheduled date of expiry by the issuing CA. Possible reasons include: key compromise, request for cancellation by an entity, termination of the entity, etc. Thus there needs to be a means to inform all relevant users that an apparently valid certificate is no longer valid. This is typically done by means of a *Certificate Revocation List (CRL)*. A CRL is a time-stamped list of serial numbers or other certificate identifiers for those certificates which have been revoked by a particular CA. The CRL is signed by the relevant CA. Updates should be issued at regular intervals, even if the list has not changed (thus enabling users possessing a CRL to check that it is the current one). Means need to be provided for the effective and timely distribution of CRLs.

11.4 Key establishment using symmetric techniques

ISO/IEC 11770-2 (ISO, 1996c) defines key establishment mechanisms using symmetric cryptographic techniques (mainly using symmetric encipherment but also using cryptographic check functions). The text is primarily, but not exclusively, based on using authentication protocols from ISO/IEC 9798-2 (ISO, 1994b) for key distribution.

ISO/IEC 11770-2 includes 13 'key establishment mechanisms' for:

- session key distribution between a pair of entities with a pre-established shared 'master key',
- key distribution between a pair of parties employing a trusted third party acting as a Key Distribution Centre, and
- key distribution between a pair of parties employing a trusted third party acting as a Key Translation Centre.

We consider four representative examples. In these examples we use the following notation.

- A and B are the two entities wishing to establish a new secret key.
- $e_K(X)$ denotes encipherment of data block X using secret key K (note that the encipherment technique is not specified). Note that the encipherment technique is assumed to provide data integrity and origin authentication. Hence it is implicit that an MDC or MAC will be computed on the data and appended to the data prior to encryption.
- $X \| Y$ denotes the concatenation of data items X and Y, *in the order specified*.

11.4.1 Authenticated key establishment using timestamps and encipherment

For this mechanism to be usable, entities *A* and *B* must already share a secret key *KAB*. *A* and *B* must also maintain synchronised clocks or sequence numbers. This mechanism is based on the one-pass (unilateral) authentication mechanism given in ISO/IEC 9798-2, Clause 5.1.1. It provides *unilateral authentication* of *A* to *B*, and *implicit key authentication* to *A*. *A* chooses the key and therefore has *key control*.

The mechanism has one message pass:

$$A \rightarrow B\text{:}\ \ e_{KAB}(\,T/N \,\|\, B \,\|\, F \,\|\, \text{Text1})$$

T/*N* denotes either a timestamp *T* or a sequence number *N*, *B* denotes the distinguishing name of *B*, and *F* contains keying material. On receipt of the message, *B* deciphers the enciphered part, and then checks for the presence of its identifier and the correctness of the timestamp/sequence number. The key established between *A* and *B* is contained in *F*.

11.4.2 Authenticated key establishment using nonces and encipherment

Mechanism 6 (a nonce-based key distribution mechanism) is derived from the 3-pass authentication protocol in Clause 5.2.2 of ISO/IEC 9798-2. To use this mechanism, *A* and *B* must share a secret key *KAB*. It provides *mutual authentication* between *A* and *B*. In the most general version of its use, no individual entity has *key control*.

The mechanism has three message passes:

$B \rightarrow A$: R_B

$A \rightarrow B$: e_{KAB} ($R_A \,||\, R_B \,||\, B \,||\, F_A \,||$ Text1)

$B \rightarrow A$: e_{KAB} ($R_B \,||\, R_A \,||\, F_B \,||$ Text2)

R_A and R_B are (unpredictable) nonces. F_A and F_B contain keying material. *A* and *B* calculate their new shared key as a function of the keying material F_A and F_B. The standard permits either F_A or F_B to be null; however if both F_A and F_B are used, then the properties required for the function used to combine them mean that neither entity has key control.

11.4.3 TTP-aided authenticated key establishment using nonces

Mechanism 9 (a nonce-based key distribution mechanism) is based on the 5-pass authentication protocol in clause 6.2 of ISO/IEC 9798-2. Note that, in this protocol, *T* is a third party (a KDC) trusted by both *A* and *B*. Moreover *T* shares the secret keys *KAT* and *KBT* with *A* and *B* respectively. The mechanism provides *mutual authentication* between *A* and *B*. The KDC has *key control*.

The mechanism has five message passes:

$B \rightarrow A$: R_B

$A \rightarrow T$: $R_A \,||\, R_B \,||\, B$

$T \rightarrow A$: $e_{KAT}(\, R_A \,||\, F \,||\, B \,||\, \text{Text1} \,) \,||\, e_{KBT}(\, R_B \,||\, F \,||\, A \,||\, \text{Text2} \,)$

$A \rightarrow B$: $e_{KBT}(\, R_B \,||\, F \,||\, A \,||\, \text{Text2} \,) \,||\, e_K(\, R'_A \,||\, R_B \,||\, \text{Text3} \,)$

$B \rightarrow A$: $e_K(\, R_B \,||\, R'_A \,||\, \text{Text4} \,)$

K is the new shared key generated by *T*, and it is contained in the keying material field *F*. R_A, R_B and R'_A are nonces.

11.4.4 TTP-aided authenticated key establishment using timestamps

Mechanism 12 (a timestamp-based key distribution mechanism) is based on, but is not fully compatible with, the 4-pass authentication protocol in clause 6.1 of ISO/IEC 9798-2. Note that, in this protocol, *T* is a third party (a KTC) trusted by both *A* and *B*. Moreover *T* shares the secret keys *KAT* and *KBT* with *A* and *B* respectively. *T*, *A* and *B* must also maintain synchronised clocks or sequence numbers. The mechanism provides *mutual authentication* between *A* and *B*. Entity *A* has *key control*.

The mechanism has four message passes:

$A \rightarrow T$: $e_{KAT}(\, TVP_A \,||\, B \,||\, F \,||\, \text{Text1} \,)$

$T \rightarrow A$: $e_{KAT}(\, TVP_A \,||\, B \,||\, \text{Text2} \,) \,||\, e_{KBT}(\, T_T/N_T \,||\, F \,||\, A \,||\, \text{Text3} \,)$

$A \rightarrow B$: $e_{KBT}(\, T_T/N_T \,||\, F \,||\, A \,||\, \text{Text3} \,) \,||\, e_K(\, T_A/N_A \,||\, B \,||\, \text{Text4} \,)$

$B \rightarrow A$: $e_K(\, T_B/N_B \,||\, A \,||\, \text{Text5} \,)$

TVP_A is a time variant parameter (random number, timestamp or sequence number) chosen by *A* and used by *A* to match the response from *T* with the request to *T* (it is not checked by *T*). T_X/N_X denotes either a timestamp T_X or a sequence number N_X generated by entity *X*. *K* is the new shared key generated by *A*, and it is contained in the keying material field *F*.

11.5 Key establishment using asymmetric techniques

ISO/IEC 11770-3 (ISO, 1999f) defines key establishment mechanisms based on asymmetric cryptographic techniques. It provides mechanisms using asymmetric techniques to agree a shared secret key between two entities (seven key agreement mechanisms), transport a secret key from one entity to another (six key transport mechanisms), and make an entity's public key available to other entities in a verifiable way (three public key transport mechanisms). The first two types of mechanism include uses of some of the protocols defined in ISO/IEC 9798-3 (ISO, 1998b) but also include a variety of other asymmetric techniques (e.g. Diffie-Hellman key exchange). The third type of mechanism includes the use of certificates.

The seven key agreement mechanisms specified in ISO/IEC 11770-3 all make use of a 'mathematical context', which essentially means the pre-agreement by the relevant parties (A and B) of a readily computed function F with certain very special properties. More specifically:

$$F\colon\ H \times G \to G,$$

where G and H are sets. F satisfies:

- $F(h, F(h', g)) = F(h', F(h, g))$ for every $h, h' \in H$ and every $g \in G$, and
- Given $F(h, g)$, $F(h', g)$ and g it is computationally infeasible to find $F(h, F(h', g))$. Amongst other things this implies that $F(\cdot, g)$ is one-way, for every g.

A and B must share a common element $g \in G$ (which may be public).

One example of a candidate for F provided in ISO/IEC 11770-3 is the discrete logarithm mechanism which underlies Diffie-Hellman key exchange, namely: $G = Z_p$ (the integers modulo p for some prime p), $H = \{ 1, 2, ... , p\text{-}2 \}$, g is a primitive element from Z_p, and $F(h, g) = g^h \bmod p$. Note that,, on this case, the prime number p needs to be chosen with care.

11.5.1 Key agreement with mutual implicit key authentication

This example is specified in ISO/IEC 11770-3 as ‘Key agreement mechanism 5’. If entities A and B wish to use this mechanism to establish a new shared secret key, then:

- entity A must have a private key $h_A \in H$ known only to A and a public key $p_A = F(h_A, \text{g})$ known to B,
- entity B must have a private key $h_B \in H$ known only to B and a public key $p_B = F(h_B, \text{g})$ known to A.

This mechanism provides mutual implicit key authentication, but does not, however, allow A and B to choose the value or form of K_{AB} in advance, i.e. neither A nor B has key control. Hence this mechanism is inappropriate for systems where the key needs to have a special form - in such a case a *key transport* mechanism needs to be used. Prior to starting the mechanism, A chooses a random (secret) $r_A \in H$, and B chooses a random (secret) $r_B \in H$. The protocol is simply:

$$A \rightarrow B\text{:}\ \ F(r_A, g) \,\|\, \text{Text1}$$

$$B \rightarrow A\text{:}\ \ F(r_B, g) \,\|\, \text{Text2}$$

After receipt of the first message, B computes the shared secret key as

$$K_{AB} = w(\ F(h_B, F(r_A, g)), F(r_B, p_A)\ ),$$

and after receipt of the second message A computes the shared secret key as

$$K_{AB} = w(\ F(r_A, p_B), F(h_A, F(r_B, g))\ ),$$

where w denotes an (unspecified) commonly agreed one-way function.

11.5.2 Key transport with mutual authentication

Transport Mechanism 5 (a nonce-based key transport mechanism) is based on the 3-pass authentication protocol in clause 5.2.2 of ISO/IEC 9798-3. Note that, in this protocol:

- *A*, *B* have signing/verification transform pairs (S_A,V_A) and (S_B,V_B) respectively. Both parties must have access to each other's public verification transformation.
- *A*, *B* have encrypt/decrypt transform pairs (E_A,D_A) and (E_B,D_B) respectively. Both parties must have access to each other's public encipherment transformation.

The mechanism provides *mutual authentication* and optional *key confirmation* to *B*. Two keys are established (one transported in each direction).

The mechanism has three message passes:

$A \to B$: r_A || Text1

$B \to A$: $S_B(\ r_B \mid\mid r_A \mid\mid A \mid\mid E_A(\ B \mid\mid K_B \mid\mid \text{Text2}\ ) \mid\mid \text{Text3}\ ) \mid\mid \text{Text4}$

$A \to B$: $S_A(\ r_A \mid\mid r_B \mid\mid B \mid\mid E_B(\ A \mid\mid K_A \mid\mid \text{Text5}\ ) \mid\mid \text{Text6}\ ) \mid\mid \text{Text7}$

r_A, r_B denote nonces. The keys K_A and K_B are established between *A* and *B*. For key confirmation to *B*, user *A* can include a check-value computed on K_B in Text6. Mutual key control can be achieved by combining the two keys K_A and K_B using a one-way function, yielding a key agreement mechanism.

12 Other standards

A multi-part standard specifying security mechanisms based on elliptic curves, ISO/IEC 15946, is at an early stage of development.

- *Part 1* contains mathematical background on elliptic curves.
- *Part 2* covers elliptic curve signatures.
- *Part 3* covers elliptic curve key establishment techniques.

A further new area for standardisation currently being investigated within SC27/WG2 concerns methods for key generation. Whilst this is potentially an extremely important area, no substantive document exists as yet.

References

ANSI. (1981). ANSI X3.92. American National Standard – Data Encryption Algorithm. American National Standards Institute.

ANSI. (1983). ANSI X3.106, American National Standard for Information Systems – Data Encryption Algorithm – Modes of Operation. American National Standards Institute.

ANSI. (1986a). ANSI X9.9 (revised), American National Standard – Financial institution message authentication (wholesale). American Bankers Association.

ANSI. (1986b). ANSI X9.19, American National Standard – Financial institution retail message authentication. American Bankers Association.

CCITT. (1988). X.509, The Directory – Authentication Framework. CCITT.

FIPS. (1980). FIPS 81, DES Modes of Operation. National Bureau of Standards.

FIPS. (1993). FIPS 46, Data Encryption Standard. National Bureau of Standards, (FIPS 46-2: 2nd revision).

FIPS. (1994). FIPS 186, Digital signature standard. National Institute of Standards and Technology.

FIPS. (1995). FIPS 180-1, Secure hash standard. National Institute of Standards and Technology, 1st revision.

ISO. (1986). ISO 8730, Banking – Requirements for message authentication (wholesale). International Organization for Standardization.

ISO. (1987a). ISO 8372, Information processing – Modes of operation for a 64-bit block cipher algorithm. International Organization for Standardization.

ISO. (1987b). ISO 8731-1, Banking – Approved algorithms for message authentication – Part 1: DEA. International Organization for Standardization.

ISO. (1989). ISO 7498-2, Information processing systems – Open Systems Interconnection – Basic reference model – Part 2: Security architecture. International Organization for Standardization.

ISO. (1991). ISO/IEC 9796, Information technology – Security techniques – Digital signature scheme giving message recovery. International Organization for Standardization.

ISO. (1992). ISO 8731-2, Banking – Approved algorithms for message authentication – Part 2: Message authenticator algorithm. International Organization for Standardization, 2nd edition.

ISO. (1994a). ISO/IEC 9797, Information technology – Security techniques – Data integrity mechanism using a cryptographic check function employing a block cipher algorithm. International Organization for Standardization, 2nd edition.

ISO. (1994b). ISO/IEC 9798-2, Information technology - Security techniques - Entity authentication - Part 2: Mechanisms using symmetric encipherment algorithms. International Organization for Standardization.

ISO. (1994c). ISO/IEC 10118-1, Information technology – Security techniques – Hash-functions – Part 1: General. International Organization for Standardization.

ISO. (1994d). ISO/IEC 10118-2, Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher algorithm. International Organization for Standardization.

ISO. (1995). ISO/IEC 9798-4, Information technology - Security techniques - Entity authentication - Part 4: Mechanisms using a cryptographic check function. International Organization for Standardization.

ISO. (1996a). ISO/IEC 10181 Parts 2 to 6, Information technology – Open Systems Interconnection – Security frameworks for open systems. International Organization for Standardization.

ISO. (1996b). ISO/IEC 11770-1, Information technology - Security techniques - Key management - Part 1: Framework. International Organization for Standardization.

ISO. (1996c). ISO/IEC 11770-2, Information technology - Security techniques - Key management - Part 2: Mechanisms using symmetric techniques. International Organization for Standardization.

ISO. (1997a). ISO/IEC 9796-2, Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: mechanisms using a hash-function. International Organization for Standardization.

ISO. (1997b). ISO/IEC 9798-1, Information technology - Security techniques - Entity authentication - Part 1: General. International Organization for Standardization, 2nd edition.

ISO. (1997c). ISO/IEC 10116, Information technology – Security techniques – Modes of operation for an n-bit block cipher. International Organization for Standardization, 2nd edition.

ISO. (1997d). ISO/IEC 13888-1, Information technology – Security techniques – Non-repudiation – Part 1: General. International Organization for Standardization.

ISO. (1997e). ISO/IEC 13888-3, Information technology – Security techniques – Non-repudiation – Part 3: Mechanisms using asymmetric techniques. International Organization for Standardization.

ISO. (1998a). ISO/IEC CD 9797-2, Information technology – Security techniques – Message Authentication Codes (MACs) – Part 2: Mechanisms using a hash-function. International Organization for Standardization.

ISO. (1998b). ISO/IEC 9798-3, Information technology - Security techniques - Entity authentication mechanisms - Part 3: Mechanisms using digital signature techniques. International Organization for Standardization, 2nd edition.

ISO. (1998c). ISO/IEC 10118-3, Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. International Organization for Standardization.

ISO. (1998d). ISO/IEC 10118-4, Information technology – Security techniques – Hash-functions – Part 4: Hash-functions using modular arithmetic. International Organization for Standardization.

ISO. (1998e). ISO/IEC 13888-2, Information technology – Security techniques – Non-repudiation – Part 2: Mechanisms using symmetric techniques. International Organization for Standardization.

ISO. (1998f). ISO/IEC 14888-1, Information technology – Security techniques – Digital signatures with appendix – Part 1: General. International Organization for Standardization.

ISO. (1998g). ISO/IEC FDIS 14888-2, Information technology – Security techniques – Digital signatures with appendix – Part 2: Identity-based mechanisms. International Organization for Standardization.

ISO. (1998h). ISO/IEC 14888-3, Information technology – Security techniques – Digital signatures with appendix – Part 3: Certificate-based mechanisms. International Organization for Standardization.

ISO. (1999a). ISO/IEC DIS 9594-8, Information technology – Open Systems Interconnection – The Directory – Part 8: Authentication framework. International Organization for Standardization.

ISO. (1999b). ISO/IEC FCD 9796-3, Information technology – Security techniques – Digital signature schemes giving message recovery – Part 3: Discrete logarithm based mechanisms. International Organization for Standardization.

ISO. (1999c). ISO/IEC FDIS 9797-1, Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher. International Organization for Standardization.

ISO. (1999d). ISO/IEC 9798-5, Information technology - Security techniques - Entity authentication - Part 5: mechanisms using zero knowledge techniques. International Organization for Standardization.

ISO. (1999e). ISO/IEC 9979, Information technology – Security techniques – Procedures for the registration of cryptographic algorithms. International Organization for Standardization, 2nd edition.

ISO. (1999f). ISO/IEC 11770-3, Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric techniques. International Organization for Standardization.

ITU. (1997). X.509, Information technology – Open Systems Interconnection – The Directory – Authentication Framework. ITU-T, 3rd edition.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. Communications of the ACM **21**:558–565.

Menezes, A.J., van Oorschot, P.C., and Vanstone, S.A. (1997). *Handbook of Applied Cryptography*. CRC Press.

RFC. (1997). RFC 2104, HMAC: Keyed hashing for message authentication. Internet Request for Comments 2104, H. Krawczyk, M. Bellare and R. Canetti.