Z08

Distributed Systems Security

Prof. Steve Wilbur

Room G03, Ext. 1397
s.wilbur@cs.ucl.ac.uk

1 Introduction

In a single computer system, many of the threats to security can be countered by providing special hardware to prevent processes interfering with each other. In communications systems such hardware protection mechanisms cannot be used and *cryptographic* techniques generally form the basis of any security system. In these notes we shall first look at the threats in a distributed system, then look at the basic encryption mechanisms. The way in which these are used is crucial to the security of the system and a number of techniques have been developed which will be discussed. Cryptographic systems depend on keys, and keys need to be distributed to participants. This, coupled with key storage, is perhaps one of the hardest problems in communications security, and will also be discussed. Finally, many systems involve multiple administrations which are autonomous. Coping with them, whilst being sceptical about the trustworthiness of all parts of the system is of growing concern, and has been the subject of papers including [Birrel86].

2 Aspects of Security

There are four broad areas of security in distributed systems:

- Access Control
- Security Threats
- Authentication and
- Notarisation

2.1 Access Control

In a simple client-server interaction the server may wish to limit access to resources. The standard techniques for doing this are:

- Access Control Matrix
- Capabilities
- Labels

The access control matrix is familiar from many systems, usually splitting the world into the “owner” of the resource, “friends” and “the rest of the world”. The controls may be enforced on a limited number of operations e.g. Read, Write, Delete, Change Protection, etc. Typically each resource will hold its access matrix close by, and the client must first prove its identity and be classified before access can be granted. It is usually the client’s principal, i.e. the person who uses the client process, who is authenticated, but there may be more than one principal. Extensions to such schemes also exist, eg. UNIX where the “set UID” bit is used to allow the creator/installer of a program to exert their (superior) authority even when the client process is executed by a less privileged person.

Capabilities are unforgeable tokens which include indicators as to which operations can be performed on a resource. In a single machine environment such tokens are never passed to process, they are kept securely by the operating system; only references to them are passed out. In a distributed system they must be sent through the network, and it must not be possible to alter them without such changes being detected. Moreover, it is desirable that a process should be able to pass a reduced set of privileges to another process via a capability, preferably without having to consult the original issuer of the capability, i.e. it should be possible for ordinary processes to reduce access rights offered by capabilities but not to enhance them. Techniques to do this exist.

Labels are used in so-called *mandatory security* systems. The techniques above are used in *discretionary security* systems. Mandatory control is used in military and high security systems, where classifications such as low, medium, high and top security are used. Information derived from that of other high security information would be classified high security, and only people/processes of suitable rank could access it. A set of rules would exist to determine the classification of new documents derived from existing classified sources. In such systems there is a tendency for all new information to eventually migrate to the highest classification, so de-classification has to take place periodically.

2.2 Security Threats

Consider a client and server connected by a network link. There are a number of threats posed by such links:

- Passive Tap
- Active Tap
- Denial of Service
- Faking
- Replay
- Traffic Analysis
- Accidental Access

The *passive tap* allows the interloper to observe traffic passing on the link. Whilst it cannot change the observed messages, it may be able to use the information to gain improper access, e.g. if a password is observed it can later be used to log in as someone else. Passive tapping is extremely easy on some networks, e.g. shared Ethernet.

An *active tap* is where the interloper can interpose a process between the original client and server. This may be purely for obtaining information such as passwords ("Trojan Horse"), or it may be as a means of changing messages, usually to the benefit of the interloper.

An often forgotten threat is *denial of service*. In some cases preventing access by a legitimate user can be a serious threat, so very persistent attempts to access a server, although they may be badly formed messages, may jam up the protocol handler and prevent legitimate access. There was reputed to be such a case some years ago where bookings for a Bruce Springsteen concert jammed all the Washington telephone exchanges for several hours. We were assured that there was no threat to key US Government lines.

A process may *fake messages* and send them to a server. These may be modeled on previous legitimate messages, with key fields changed. If the fields are not changed, it is termed replay, and this may be valuable to the fraud if it can get, say, multiple payments into its bank account. *Audit trails* are usually used in secure systems to record all transfers, but although the perpetrator may be known, it may be too late to recover the loss. In computer systems, audit trails need to record details of people, processes and machines, although the latter is very difficult on LANs.

Traffic analysis is usually used to infer where and when a military engagement will take place. This is based on the assumption that message activity will increase before a battle, and there will be a high traffic density on links to the front-line forces. There are some analogous cases in commerce. The usual defence is to keep all links heavily loaded with random (information-free) messages, and when messages need to be sent to transmit them in place of this background traffic.

Accidental access can be caused by a number of failures, e.g. crossed lines, software faults etc. Perhaps the most common on dial-up computer connections is the failure of a call to be completely disconnected by software, and for a subsequent caller to be communicated directly to the previous caller's session without any formalities. Similar problems can occur in packet protocols.

2.3 Authentication

We have already identified the need for authentication, both of the client and the server. Client authentication usually involves authentication of the principal. Server authentication is important not only because of the threat of active tapping (Trojan Horse) but also because many RPC binders only offer hints as to where a service is located. If it crashes, it may be replaced by another, totally different server, but the client needs to know of this mismatch so that the binder may be consulted again. Cryptography can be used for such purposes.

A related term is *message integrity*, often called message authentication. It is often important to be sure that a received message has not been tampered with in transit. Sometimes there is no requirement for secrecy of the message in transit; sometimes there is. A form of strong sumcheck based on cryptographic techniques is appended to messages for such purposes. This is sometimes called a digital signature or a message authentication code (MAC).

2.4 Non-Repudiation

Sometimes an even stronger requirement exists, such that the sender cannot deny sending a message. In such cases a notary is used to register messages such that neither of the participants can back out of a transaction and disputes can be resolved by presenting relevant signatures or encrypted text.

3 Cryptography

3.1 Ciphers

The purpose of *encryption* or *encipherment* is to transform a message or *plaintext* into an apparently random pattern, the ***ciphertext***, such that *decryption* of the ciphertext is extremely difficult unless an appropriate algorithm is known. For a normal *two-way cipher* there will be encryption and decryption algorithms which are complementary; for a *one-way* ***cipher*** the encryption algorithm will have no known inverse, i.e. it is not possible to algorithmically convert ciphertext to plaintext.

An example of a simple two-way cipher is the *Caesar cipher*, where each letter of the message is shifted by a number of places, say 3; such shifts are considered to be circular. To decrypt, a three place shift is applied in the opposite direction, e.g.

VENI VIDI VICI

might become:

YHQL YLGL YLFL

To maintain secrecy it is generally necessary to change the encryption algorithm periodically. If this were done on a daily or hourly basis, a large number of algorithms might need to be agreed in advance. For this reason, algorithms can usually be varied by a *key*, e.g. in the above example the key was 3 (one of a total of 26 in this case). In a good system there will be a vast number of keys so as to make it difficult to break the code by *exhaustive search* of the key space, i.e. where the ciphertext is decrypted with every possible key until recognisable cleartext emerges.

In the most common systems, known as *Secret Key Cryptography* (SKC), identical or closely related keys are used for both encryption and decryption. Thus knowledge of both the algorithm and the key would breach security. For this reason, although the algorithm may be made public it is vital that the keys be kept secret. A typical key may be 56 bits long so even if the algorithm is public, exhaustive search will take 11,000 years if the encipherment process only took 5 microseconds!

An alternative is *Public Key Cryptography* (PKC). The aim here is to develop a two-key system where one key is used for encryption and another is used for decryption. Clearly the two keys are related, but the mechanism is designed such that knowing one key yields little clue to the other one. More specifically, to compute the other key with the best of known algorithms would take hundreds or thousands of years. If this is the case then one key can be published, the *public key*, and one key is kept by the owner, the *personal* or *secret* key. When a message is sent it is encrypted with the public key of the recipient, and only the recipient can decrypt the message. By an interesting twist this mechanism can also be used to prove the authenticity of the sender. If the message is encrypted with the sender's personal key then anyone can decrypt it by virtue of the published public key. If it decrypts properly, and the published keys are trustworthy then the originator has been authenticated. A combination of the two techniques can be used to provide secrecy and authentication.

3.2 Block and Stream Cipher Security

One of the important encryption algorithms of recent years has been the, NBS Data Encryption Standard [Diffie77, FIPS74]. This secret key algorithm encrypts 64-bit blocks of data using a series of transformations and substitutions, and has a 56-bit key. The 64-bit cleartext is thus transformed into a 64-bit block of cipher-text. This is the *electronic code-book* (ECB) [FIPS81] mode of operation, where a given pattern is transformed into another pattern equivalent to a table lookup. Under the same key, the same cleartext will always produce the same ciphertext. The DES algorithm is not usually used in ECB mode because many messages have a similar format, e.g. bank transactions may have a fixed pattern of bits at the start equivalent to “Pay to account ...”. If ECB is used, a known plaintext attack may be used by a fraud to decrypt the text and possibly manufacture new messages.

To overcome this *Cipher Block Chaining* (CBC) can be used. Besides the encryption device, a 64-bit buffer is introduced into the system. This is loaded with a previously agreed value, the *initialisation vector* (IV), before the message is sent. The first 64 bits of the message are XOR-ed with the contents of the buffer, and the result is encrypted, sent to the recipient, and stored in the buffer. The process is repeated for successive 64-bit blocks. Thus, each block depends on its predecessor (or the IV) and increases the difficulty of systematic analysis. The recipient has a complementary scheme for decrypting, and must know both the IV and the key.

There are two other modes of use of the DES algorithm. The first, *Cipher Feedback Mode* (CFM), is rather similar to CBC except that it aims to encipher character streams, i.e. 8-bit characters arriving at arbitrary times. Initially, the IV is encrypted, and this value is stored in a buffer (select buffer), which is selected 8 bits at a time to XOR with the character stream. The outgoing coded characters are sent to the recipient and to the buffer which held the IV. After the eighth character this buffer is re-encrypted. Thus, there might be a slight delay after every eighth character.

In the second stream cipher, termed *Output Feedback* (OFB), the output of the select buffer is fed back to the IV buffer directly. The selected value is also XOR-ed with the message characters, but this output is only fed to the recipient. Thus, the IV buffer, encryption, and select buffer form a random sequence generator, independent of the message contents. Such a system is much faster, and transmission errors do not propagate, so might be useful for broadcasting of video etc.

The above techniques are specified in the DES standards, but are applicable to other block ciphers.

3.3 Encryption Algorithms

3.3.1 DES Algorithm

The Data Encryption Standard (which is not an ISO standard) is a block cipher, dealing with 64-bit data blocks encrypted under a 56-bit key. The algorithm is sketched in the attached diagrams.

3.3.2 RSA Algorithm

The Rivest, Shamir and Adleman algorithm (RSA) [Rivest,78] belongs to a family of exponentiation ciphers. They rely on the comparative ease of computing exponentials and the difficulty of computing logarithms. They also use modular arithmetic, where the modulus is usually a very large number. Thus, a message M may be encrypted to ciphertext C and decrypted back to M by application of:

$$C = (M^e) \bmod N \qquad \text{.... (1)}$$

and,

$$M = (C^d) \bmod N \qquad \text{.... (2)}$$

where e and d are the encryption and decryption keys respectively, and N is a large integer. N is usually of the order of 500-1,000 bits in length.

In order for us to appreciate the RSA approach we need a little mathematics first. *Fermat's Theorem* states that if p is prime and gcd(a,p)=l ("gcd" means greatest common divisor) then:

$$a^{p-1} \bmod p = 1$$

Thus if a and p have no factors in common, i.e. a is not a multiple of p, the above relationship holds.

Euler's Generalisation states that for every a and n such that gcd(a,n)=l then:

$$a^{\varphi(n)} \bmod n = 1 \qquad \text{.... (3)}$$

where φ(n) is the *Euler Totient Function*, i.e. the number of positive integers less than n which are relatively prime to n. For n=10, φ(n) = 4, the relative primes are (1, 3, 7, 9). If p is prime, then φ(p)=p-1.

For our encryption purposes we now take equations (1), (2), and (3), giving:

$$M^{\varphi(n)} \bmod n=1$$

from which it can be shown that:

$$e * d \bmod \varphi (n) = 1. \qquad \text{.... (4)}$$

We are now in a position to construct a simple exponentiation cipher using the *Pohlig-Hellman Scheme*. In this the modulus is a prime p, so that φ(p)=p-1. Using p=11 and choosing the decryption key d to be 7, we get:

$$7 * e \bmod 10 = 1.$$

Clearly, e=3 satisfies this. If we now encipher a message, say M=5, we get:

$$C = 5^2 \bmod 11 = 4$$

and decrypting it gives:

$$M = 4^2 \bmod 11 = 5.$$

However, in order to operate this scheme both the sender and recipient need to know the modulus p, one needs e and the other one needs d. However, knowing p means that φ (p) is also known (p-1), and given one key the other is then easily discovered using equation (4). This is clearly not suitable for a public key scheme.

The RSA approach is similar, but chooses the modulus n to be the product of two primes p and q. Thus:

$$n = p * q$$

and, $\varphi (n) = (p-1) * (q-1)$

If we take p=5 and q=7, we get n=35 and φ (n)=4*6=24. Choosing a key d=11, e can be computed using equation (4), i.e.:

$$e = \mathrm{inv}\,(11, 24) = 11.$$

For a message M=2:

$$C = 2 \bmod 35 = 2048 \bmod 35 = 18$$

and, $M = 18 \bmod 35 = 2$

Of course, e and d do not usually turn out to be identical. The security of this scheme compared to the Pohlig-Hellman one is as follows. Both parties need to know n (but definitely not its factors), one needs to know e and the other needs to know d. However, unless the factors of n are known it is extremely difficult to determine φ (n) and thus to determine the other key from the one in your possession. The difficulty of factorising integers is related to the time taken to compute discrete logarithms, for which the fastest known algorithm takes of the order:

$$T = O(\exp(\mathrm{sqrt}(\ln(n) . \ln(\ln(p)))))$$

For a 200-bit value of p at one step per microsecond factorisation might take about two to three days, while for a 200-decimal digit key (664 bits) it would take several billion years. It is, however, relatively easy to prove that a number is prime in a short space of time so constructing n from constituent primes is easy.

3.4 Breaking Ciphers

There is only one unbreakable form of encryption, known as the *one-time-pad***.** Both parties in the communication have an identical long list of random values which they use in sequence to encode the data units being sent. If the values are truly random, i.e. not generated by an algorithm, then the code is unbreakable.

Other codes are based on algorithms, and so are, in principle, breakable. They are usually designed to take either very long periods of time or large amounts of space to decipher. The "brute force" approach is an *exhaustive search*. Successive keys are tried on ciphertext until either the known plaintext or recognisable plaintext emerges. Even with fast encryption, a 56-bit key would take hundreds of years to break in this way. A US study in the early 1980s estimated that by 1990 it might be possible to have a machine available to find a key within a day, but it would be of enormous cost, consume enormous amounts of power, and there was only a 10-20% probability that it could be built.

If it is possible for the cryptanalyst to insert some plaintext into the system and observe the ciphertext, this may provide a basis for finding the key. Such attacks are known as *chosen-plaintext* attacks.

Other attacks can be made on the basis of the forms of communication. For example, if only two messages were ever transmitted e.g. financial shares rising or falling, it would not take much to detect their encodings. Clearly the key needs to be changed for every message, or some other form of *whitening* is needed.

4 Techniques

4.1 Passwords

Passwords, pass phrases and personal identification numbers (PINS) are the major means of authenticating users of computing equipment. Passwords illustrate the point that system security not only depends on good algorithms, but also good operational practice. However passwords are stored in a computer system it is unwise to choose:-

a) a short password
b) an easily predicted one (e.g. spouse's name)
c) one from a very small character set (e.g. telephone number).

NIST have produced guidelines identifying 10 characteristics which need to be carefully considered in designing a password system [FIPS112] which include:

- length
- character set

- lifetime
- source, i.e. user or administrator generated
- ownership, preferably individual
- distribution
- storage
- entry e.g. keyboard, display, possibility of being overlooked
- etc.

Storage and transmission can pose problems. Many systems now use a one-way encryption function for storage of passwords (e.g. UNIX), and the checking program takes the cleartext password and encrypts it before comparing with the stared value. The benefit of such an approach is that no special security needs to be applied to the password file because decryption is "impossible". Nonetheless, exhaustive search techniques can be used, hence the need for careful management of passwords.

Where passwords need to be transmitted on insecure media, neither the cleartext nor the one-way enciphered form is safe: in one case an observer could see the password, in the other the enciphered password is freely available in a file, so a system using it via a link is totally insecure. Transmission of passwords requires them to be further encrypted while in transit.

4.2 Capabilities

In an object-based system access control might be by capabilities which contain some form of object identification, a bit map of the allowed operations, and a random component which is denied from the bits in the capability. If this is to be secure, the algorithm for generating the random component must be kept secret. A public algorithm may be used if the random component (which acts as a means of detecting improper changes) is generated from the other components and a key kept by the server. Since the server generates and checks capabilities before using them there is no difficulty of key distribution. This solution is cumbersome in distributed systems, since whenever a client wishes to pass on a restricted capability to another process, it must make a request of the server to manufacture the new capability, involving a network request.

Mullender and Tanenbaum [Mullen84] have proposed an interesting scheme using only one-way cryptography to overcome this. In the simplest case where all rights are granted the scheme is similar to that above, i.e. the server key is used to encrypt the constant fields of the capability. This is stored in the check (random) field, and is also saved by the server for future reference. For each bit in the rights field, Ri, there is a corresponding one-way algorithm, Ai. In order to reduce the rights by Rj, the appropriate rights bit is cleared, and the algorithm is applied to the check field. Clearly applying the algorithms must be commutative, i.e.

$$Ai\ (Aj(X)) = Aj(Ai\ (X))$$

so that rights can be diminished in any order to yield the same value in the check field.

With this scheme it is possible for all machines to have an agreed set of Ai, and for clients to reduce the rights before passing the capabilities to others, without recourse to the issuing server.

4.3 Cryptographic Sealing

The capability scheme above is suitable where an object-oriented approach is being used, i.e. we are trying to prevent access to objects by limiting access to the legal functions. An alternative approach was suggested by Gifford [Gifford82] to deal with access control to data items. Rather than enforcing access control by active means close to where the item is stored, it is cryptographically sealed, and made freely available to anyone. However, keys are distributed only to legitimate users. Let us look at how this might be achieved. We shall use the notation:

$$\{x\}k$$

to mean that the plaintext x has been encrypted with the key k, this being our basic "sealing" operation. To unseal x the decryption algorithm is applied with key k. Thus in the simplest case the data is sealed into S_x, i.e.

$$S_x = \{x\}k$$

and only possessors of key k can access it.

An **indirect key** I can be made as follows:

$$I = \{ k_1 \} k_2$$
$$S_x = \{ x \}k_1$$

Thus, to unseal S_x it is first necessary to unseal I to obtain k_1. Only possessors of k_2 can do this. Indirect keys allow more powerful sealing to be performed. For example, if it is desired to seal x so that either k_1 or k_2 can unseal it then two indirect keys can be used:

$$I_1 = \{k\} k_1$$
$$I_2 = \{k\} k_2$$
$$S_x = \{x\}k$$

A logical AND functional can be achieved by double sealing of data (or an indirect key), e.g.

$$I = \{\{k\} k_1\} k_2$$
$$S_x = \{x\}k$$

In an ideal case the double encryption would be commutative so that sealing or unsealing could be applied in any order.

Clearly access control mechanisms can be built using this technique and it is ideal for read access. However, there are two problems which need to be dealt with:

- key distribution
- update access

Key distribution is a problem because keys may be compromised and it is necessary to distribute a new key to legitimate users. The free read-access that this system uses does not work so well with updates. If all users can freely overwrite files, interlopers can replace data items with garbage and at least deny others the legitimate service. With an asymmetric cipher such as a public key system it would be possible to give some users both read and write access by giving them both keys, and to give others only read access by providing them with only one of the keys. However, an additional mechanism is needed to prevent unauthorised overwriting of the data by non-key holders.

4.4 Authentication

4.4.1 SKC Approach

Let us suppose that a client, A, wishes to prove its authenticity to a server, B, and also wishes to satisfy itself that B is not a Trojan Horse. The normal scheme for this is due to Needham and Schroeder [Needha78] and includes a trusted *authentication server*, AS. Before data is transferred between A and B a dialogue of the following form is needed; we shall deal with SKC first.

1.	A → AS:	A, B, Ia
2.	AS → B:	{Ia, B, Kc, {Kc, A} Kb} Ka
3.	A → B:	(Kc, A) Kb
4.	B → A:	(Ib) Kc
5.	A → B:	{f(Ib)} Kc

The client sends message 1 to the AS in clear text. This message can be intercepted and replayed, and Ia is included to prevent frauds. Ia and Ib are called *nonces* and are integers which are only used once, so replay of previous copies of message 2 will serve no purpose. The components A and B in the message are essentially the identifiers of the relevant principals at the client and server, so A may be the name of the client's user, and B may be the server's name.

The authentication server is trusted and contains the keys of all principals. Thus it can generate messages sealed with such keys, and only the relevant principal can unseal the message. Message 2 is sealed so that only the client can unwrap the outer seal to reveal Ia, B, and Kc. The remainder is sealed for "the eyes of B only". At this point the client checks that the values for Ia and B match those of the request and if so it picks up the *session* or *conversation key* Kc for later use. The remainder of the message is sent to the server as message 3.

When message 3 arrives the server can unwrap it and can obtain the name of A and the session key. The name of A is included in the wrapping in case a Trojan Horse diverted the message. By using the unwrapped return address rather than the one in the protocol packet header obvious frauds can be minimised.

The server now sends a message (4) back to the client containing a nonce encrypted with Kc. To prove that A is A it returns a message with a simple transformation of Ib, e.g. Ib-1, also encrypted. Only the possessor of Kc would be able to do this thus proving its identity. Client and server may now exchange data encrypted with Kc.

The reason for the session key is that there is potential for key discovery every time the key is used. Thus the personal keys of users are only used to distribute the session keys to each party. Thereafter the session key is used. For each interaction a different session key may be used for the same reason, i.e. session keys will be changed at irregular intervals.

The above scheme can be extended to multiple authentication servers, for cases where client and server may be in different administrations. In such cases one AS may know the key of the client, another may know the key of the server. We assume that the ASs have already established a secure channel using the session key, Kas. The protocol would be:

1.	A → AS1:	A, B, Ial
2.	AS1 → AS2:	{Kc, A, B, Ia2, A}Kas
3.	AS2 → AS1:	{ {Kc, A}Ka, Ia2, A}Kas
4.	AS1 → A:	{Ial, B, Kc, {Kc, A}Kb } ka

then, as before. Note that to the client and server this protocol is indistinguishable from the single authentication server case. Note also that AS1 chooses a conversation key and passes it to AS2 which seals it with B's key. Thus, AS2 does not need to trust AS1 with B's key. However, AS1 does have to trust AS2 not to disclose the conversation key. Different nonces are used for the interactions between A and AS1, and between ASI and AS2 to minimise the potential for faking.

4.4.2 PKC Approach

The scheme for a single AS using public keys is as follows:

1.	A → AS:	A, B
2.	AS → A:	{PKb, B}SKas
3.	A → B:	{Ia, A}PKb
4.	B → AS:	B, A
5.	AS → B:	{PKa, A)SKas
6.	B → A:	{Ia, Ib}PKa
7.	A → B:	{Ib}PKb

Data phase:

a)	A → B:	{ {Mab}SKa } Pkb
b)	B → A:	{ {Mba} SKb}PKa

Messages 1, 2, 4 and 5 are merely to obtain the public key of the other party. The reason for encryption is not for secrecy, since they are public keys, but for authentication, i.e. proving that they did actually come from the authentication server whose public key is widely and unforgeably published. Other than that the protocol is straightforward. Double encryption is used in the data phase to both authenticate and seal messages.

4.4.3 Asynchronous Communication

The above cases of authentication have been synchronous systems in that a dialogue was possible in order to establish to their mutual satisfaction that A and B were authentic. In some cases e.g. electronic mail systems, this is not possible because the delays may be very great or the parties may not be available simultaneously. The scheme suggested by Needham and Schroeder is not the only possible one, but it is workable. It uses *self-authenticating messages* and timestamps. When A wishes to send to B it asks its As to create a conversation key and wrap it up in B's key, i.e.:

Token = {Kc, A}Kb

The body of the message is then sent along with this token, but encrypted with Kc in the following way:

Seg1 = Token, {TS, S1, Mess1)Kc
Seg2 = {S2, Mess2}Kc
Seg3 = (S3, Mess3}Kc
etc .

where: TS is a unique timestamp added by A,
Mess1, Mess2 etc are the fragments of the message,
Seg1, Seg2, etc. are transport service packets, and, S1, S2, etc. are sequence numbers.

All distinct messages originating from the source are given a unique TS, and the recipient keeps a table of TS values and corresponding sources for a limited time related to the expected transit time of messages through the network. Any very old or replayed (duplicate) messages can thus be discarded.

4.5 Message Integrity

It is often required that a message be verified to be authentic, i.e. that it is intact and has come from the specified source. Depending on the application it may not be necessary to encrypt the whole message, i.e. secrecy is not always required. In order to prove authenticity of messages, a code or *signature* is attached to it, often known as a Message Authentication Code (MAC). Such a MAC can be computed by applying an encryption algorithm such as DES to the message with a key unique to the sender, and choosing the most significant k bits of the last block of the ciphertext. Such a check provides a probability of 2^{-k} that the message is authentic. k can be chosen to provide the required degree of integrity [Davies84, Juenem85]. If secrecy is also required the message may be encrypted again, with a different key.

It is possible that under some circumstances changes may be made to the ciphertext which are normally undetectable when the message is decrypted. To counter this a Manipulation Detection Code (MDC) is appended to the cleartext before encipherment. This may be a simple CRC or similar function. Changes to the ciphertext will produce an invalid MDC when the message is decrypted.

An alternative approach is given in [Needha78]. There a one-way function F is applied to the message M to produce F(M), with significantly fewer bits than the original message. A property of F must also be that it is difficult to find another message M', such that F(M') = F(M), i.e. it must be difficult to find another message with the same integrity code. Since both the client and server (and the world at large) must know F, it is necessary to enlist the aid of a trusted third party to wrap the integrity code up securely. The authentication server may be suitable, and the protocol might be:

1.	A:	Sig = F (Mess)
2.	A $\rightarrow$ AS:	A, {Sig}Ka
3.	AS $\rightarrow$ A:	{A, Sig}Kas
4.	A $\rightarrow$ B:	Mess, {A, Sig} Kas
5.	B:	Bsig = F{Mess}
6.	B $\rightarrow$ AS:	B, {A, Sig}Kas
7.	AS $\rightarrow$ B:	{A, Sig}Ka
8.	B:	Bsig = Sig ==> Authentic

Note that the signature is transported from A to B encapsulated in the authentication server's key (Kas), and this token also verifies the sender's authenticity as well as the message's. Kn and Ka are the private keys of the sender and recipient resp..

In the case of Public Key Cryptography the message from A to B would be encrypted thus:

A $\rightarrow$ B: { { Mess } Ksa } Kpb

where Ksa is the secret key of A, and Kpb is the public key of B. Thus, if B retains both *Mess* and *{Mess}Ksa* it can verify with the aid of a trusted third party that the message has not been tampered with and that it came from A. This is particularly useful where A might try to wriggle out of a contract made electronically.

5 Key Distribution

The previous section has dealt with some of the mechanisms for key distribution. Initial key distribution is often done by out-of-band means, e.g. using the postal service or face-to-face meetings. Once initial keys have been distributed, it may be possible to use the data distribution mechanism (encrypted) to transmit new keys. However, weaknesses are usually human, and it is best to prevent users from ever being able to see their keys, or they may be tempted to send the clear text via electronic mail!

Storage of keys must be similarly controlled, and it is wise to ensure that even inside a program keys are kept encrypted, so that if there is a program, machine, or file store crash, cleartext keys cannot be easily discovered in the ruins.

6 REFERENCES

[Bauer83] Bauer RK, Berson TA, Feiertag RJ, "*A Key Distribution protocol Using Event Markers*", ACM Trans on Computer Systems, 1(3), pp 249-255,Aug 1993

[Birrel85] Birrell AD, "*Secure Communication Using Remote Procedure Calls*", ACM Trans on Computer Systems, 3(l), pp 1-14, Feb 1985

[Birrel86] Birrell AD, Lampson BW, et al, "*A Global Authentication Service without Global Trust*",IEEE ?, pp 223-230, 1986

[Bottin86] Bottin86 R, "*Novel Security Techniques for On-Line Systems*", Comm. ACM, 29(5), pp 416-417 May 1986

[Chaum85] Chaum D, "*Security without Identification:- Transaction Systems to Make Big Brother Obsolete*", Comm. ACM, 28(10), pp 1030-1044, Oct 1985

[Chehey8l] Cheheyl MH, et al, "*Verifying Security*", ACM Computing Surveys, 13(3), pp 279 340, Sept 198l

[Davies8l]	Davies DW, "*Protection*", in "*Distributed Systems - Architecture and Implementation, An Advanced Courses*", ed Lampson BW, et al, Springer-Verlag 1981, ISBN 3-540-12116-1
[Davies84]	Davies DW, Price WL, "*Security for Computer Networks*", John Wiley, ISBN 0-471-90063-X, 1984
[Dennin79]	Denning DE, and Denning PJ, "*Data Security*", ACM Computing Surveys, 11(3), pp 227-250, Sept 1979
[Dennin81]	Denning DE, Sacco MS, "*Timestamps in Key Distribution Protacols*", Comm ACM, 24(7), pp 533-536, Aug 1981
[Dennin83]	Denning DE, "*Cryptography and Data Security*", Addison-Wesley, ISBN 0-201-10150-5, 1983
[Diffie76]	Diffie W, Hellman M, "*New Directions in Cryptography*2, IEEE Trans on Information Theory, IT-22(6), pp644-654, Nov 1976
[Diffie77]	Diffie W, Hellman M, "*Exhaustive Cryptanalysis of the NBS Data Encryption Standard*", IEEE Computer, pp 74-84, Jun 1977
[Diffie79]	Diffie W, Hellman M, "*Privacy and Authentication: An Introduction Cryptagraphy*", Proc. IEEE, 67(3), pp 397-427, Mar 1979
[Ehrsam78]	Ehrsam WF, Matyas SM, et al, "*A Cryptographic Key Maragement Scheme for Implementing the Data Encryption Standard*", IBM Syvstems Journal, 17(2), pp 106-125, 1978
[FIPS46]	"*Data Encryption Standard*", National Bureau of Standards, FIPS Pub 46, Jan 1977
[FIPS74]	"*Guidelines for Implementing and Using the NBS Data Encryption Standard*", National Bureau of Standards, FIPS Pub 74, Apr 1981
[FIPS81]	"*DES Modes of Operation*", National Bureau of Standards, FIPS Pub 81, Dec 1980
[FIPS112]	"*Password Usage*", National Bureau of Standards, FIPS Pub 112, May 1985
[FIPS113]	"*Computer Data Authentication*", National Bureau of Standards, FIPS Pub 113, May 1985
[Guiffor82]	Gifford DK, "*Cryptographic Sealing for Information Secrecy and Authentication*", Comm ACM, 25(4), pp 274-286, Apr 1982
[Jones78]	Jones AK "*Protection Mechanisms and the Enforcement of Security Policies*", in "*Operating Systems, An Advanced Course*", ed. Bayer R, et al, Springer-Verlag, ISBN 3-540-098127, 1978
[Juenem85]	Jueneman RR, Matyas SM, Meyer CH, "*Message Authentication*", IEEE Communications Magazine, 23(9), pp 29-40, Sep 1985
[Landwe81]	Landwehr CE, "*Formal Methods for Computer Security*", ACM Computing Surveys, 13(3), pp 247-278, Sept 1981
[Matyas78]	Matyas SM, Meyer CH, "*Generation, Distribution, and Installation of Cryptographic Keys*", IBM Systems Journal, 17(2), pp 126-137, 1978
[Meyer82]	Meyer CH, Matyas SM, "*Cryptography: A New Dimension in Computer Data Security*", John Wiley, ISBN 0-471-04892-5, 1982
[Mullen84]	Mullender SJ, Tanenbaum AS, "*Protection and Resource Control in Distributed Operating Systems*", Computer Networks, Vol 8, pp 421-432, 1984
[Needha78]	Needham RM, Schroeder MD, "*Using Encryption for Authentication in Large Networks of Computers*", Comm ACM, 21(12), pp 993-999, Dec 1978
[Popek79]	Popek GJ, Kline CS, "*Encryption and Secure Computer Networks*", ACM Computing Surveys, 11(4), pp 331-355, Dec 1979
[Rivest78]	Rivest RL, Shamir A, Adleman L, "*A Methad for Obtaining Digital Signatures and Public Key Cryptosystems*", Comm. ACM, 21(2), pp 120-126, Feb 1978, reprinted in Comm. ACM, 26(l), pp 96-99, Jan 1983
[Suhler86]	Suhler PA, et al, "*Software Authorisation Systems*", IEEE Software, 3(5), pp 34-41, Sept 1986; (Piracy protection of software)
[Summer84]	Summers RC, "*An Overview of Computer Security*", IBM Systems Journal, 23(4), pp 309-325, 1984
[Voydoc83]	Voydock VL, Kent ST, "*Security Mechanisms in High Level Network Protocols*", ACM Computing Surveys, 15(2), pp 135-171, June 1983
[Zimmer86]	Zimmerman P, "*A Proposed Standard Format for RSA Cryptosystems*", IEEE Computer, 19(9), pp 21-34, Sept 1986