A1 Security Mechanisms

This appendix is provided as a general introduction to cryptology for those not familiar with the topic.

Security mechanisms, as defined in [IS7498-2], are the means by which the OSI Security Functions can be provided. In the OSI Security Architecture mechanisms of quite different levels are all called security mechanisms. In fact these mechanisms form a hierarchy:

- *Higher level mechanisms*, such as security protocols and (semantic) message contents.

- *Lower level mechanisms*, such as cryptosystems, forming parts of the above-mentioned higher level mechanisms.

- *Physical mechanisms*, such as encryption chips and pieces of program code, implementing the above-mentioned mechanisms.

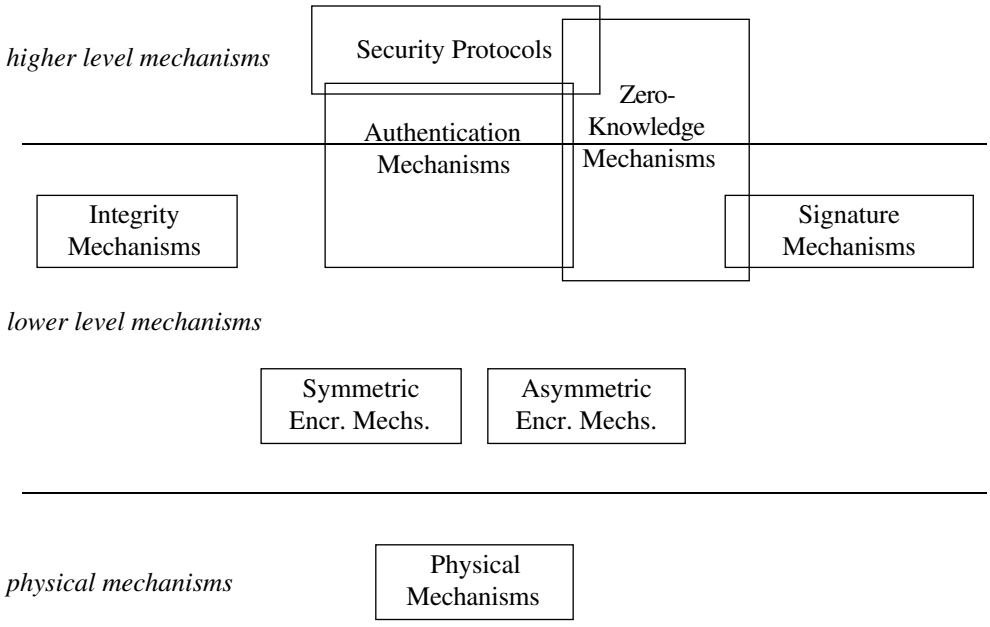

Figure A1-1: The hierarchy of security mechanisms.

The security of a local system can, to a great extent, be ensured by physical security arrangements. In a global untrusted network it is impossible to guarantee the security of communications by means of physical security but cryptographic techniques are needed. A lot of work has been done in the area of cryptology during the past couple of

 © Arto Karila, 1991

thousand years but the most significant progress has taken place after the advent of electronic computing. Today we have at our disposal a number of cryptographic mechanisms that can be used in connection with open communications architectures.

Among the basic cryptographic mechanisms are those used for data encryption, integrity and authentication. Data encryption mechanisms can be used for all these three purposes and they are usually divided into symmetric and asymmetric cryptosystems. There are cryptosystems, not based on asymmetric encryption, that can be used for purposes or authentication and non-repudiation. Here we shall discuss the four main classes of cryptographic mechanisms: *symmetric encryption mechanisms*, *asymmetric encryption mechanisms*, *signature mechanisms*, and *integrity mechanisms*, separately. The use of signature mechanisms for *authentication* is discussed and the important class of *zero-knowledge techniques* is briefly dealt with. Finally, the implementational aspects of cryptographic mechanisms and the *physical mechanisms* implementing them are touched upon.

While this chapter is outside the core of this study, it is necessary to discuss security mechanisms in some detail in order to demonstrate the implementability of the security architecture proposed and tie this work more firmly to reality.

A1.1 Data Encryption Mechanisms

An *encryption mechanism* is used to convert a cleartext message into a cryptogram, that is into a form that is unintelligible to anyone not knowing the secret decryption method. An encryption mechanism is based on a *public algorithm* and at least one *key* whose value is randomly chosen from a large set.

The need for separating the algorithm from the keys is obvious. In this way we can use the same (hardware or software) implementation of a cryptosystem over and over again with different key values. The algorithm remains the same and only a relatively compact key value needs to be changed.

The reason why the algorithm needs to be public is less obvious. The security of a cryptographic mechanism should never be based on the secrecy of an algorithm. This is a generally accepted principle known as the *Kerckhoff's principle* (named after a 19th century Flemish linguist). Only an algorithm that has been open for public criticism and cryptanalysis for a reasonable period of time can be trusted to be secure. Secret algorithms may (intentionally or not) contain trapdoors and other faults.

In fact, it can be claimed that not only the security of a cryptosystem but, in deed, the security of the entire open system should always be based on public mechanisms and secret keys. That is, everything else about a secure open system except the key values

should be public and open for analysis and criticism. Not only is this the only way to gain reasonable assurance of the security of an open system but it also seems to be a prerequisite for any legislation related to data security giving the security mechanisms used a legal standing (e.g. making digital signatures binding in the court of law).

An interesting current research area in cryptology is that of *elliptic curve cryptosystems* based on a group of points on an elliptic curve. Originally proposed in 1985 by V. Miller of IBM and further developed by N. Koblitz and others, these mechanisms seem to promise public key cryptosystems more secure than RSA with comparable block lengths in the near future. These cryptosystems are believed to be computationally intensive but the current fast progress in VLSI technology will help in their efficient implementation [BVA90].

This is not meant to be a study on new emerging cryptosystems or their principles. The only motivation here is to show, that we cannot expect anything of the nature of the keys of possible future cryptosystems. They may, for example, be sets of data items or functions. Therefore, it is vital that we leave enough flexibility in our current designs to accommodate for new cryptosystems which inevitably will emerge. In this study the word *key* is used to denote any (secret or public) reasonably compact piece of information which can be presented digitally (as a bit-string) and used in connection with a public algorithm to form a cryptographic function.

A1.1.1 Symmetric Encryption Mechanisms

In a *symmetric cryptosystem* we have two functions *encrypt* and *decrypt*. A message encrypted with key K can only be decrypted with the same key. Symmetric key cryptography, which dates back to ancient times, was developed into exact science in Shannon's famous paper written in 1945 as a secret report and declassified and published in 1949 [Sha49].

From here on we shall use the following notation for encrypting with a symmetric cryptosystem: c = eK(m), where m is the *cleartext message*, K is the *key*, e is the *encrypt function* and c is the *cryptogram*. Similarly, we use the following notation for decryption: m = dK(c), where d is the *decrypt function*.

In our notation, the key used is combined with the function rather than supplied as one of its arguments in order to point out the fact that the public part of the function and the key together form a unique function used for encryption or decryption. The key can be thought of as an index choosing one of a family of functions.

A1.1.2 Asymmetric Encryption Mechanisms

The first mention of *public key cryptography* is found in [DH76]. The first implemented *public key cryptosystems* (from here on simply *PKCs*) were the Rivest-Shamir-Adleman (RSA) scheme, presented in [RSA78], and the Merkle-Hellman *trapdoor knapsacks*, presented in [MH78]. For an introduction to the history and current status of public key cryptography see e.g. [Dif88] and [BVA90]. A short introduction is also found e.g. in [Den83]. Even though the original Merkle-Hellman knapsacks were soon broken, some of their improved versions are still believed to be quite secure. Some improved Merkle-Hellman knapsacks are presented e.g. in [LLH89].

In a (PKC) the key can be divided into two parts, the *encryption key* and the *decryption key*, in such a way that the encryption key specifies the *encryption transformation* and the decryption key determines its left inverse mapping *decryption*. If it is *computationally unfeasible* (for a more precise definition of the term see [DH79]) to determine the decryption key from the encryption key the PKC is called an *asymmetric* (or *public key*) *encryption mechanism*. If it is unfeasible to derive the encryption key from the decryption key the PKC is called a *public key signature mechanism*.

An asymmetric encryption mechanism provides complete confidentiality (only the legitimate recipient in possession of the secret key can decrypt the message) but no authentication of the sender (anybody with access to the recipient's public key could have generated the message).

From here on we shall use the following notation for encrypting with the public key: $c = eK_{P,X}(m)$, where m is the cleartext message, $K_{P,X}$ is the public key of X, e is the Encrypt function, and c is the cryptogram.

The following notation is used for decrypting with the secret key: $m = dK_{S,X}(c)$, where $K_{S,X}$ is the secret key of X and d is the *decrypt* function.

A1.1.3 On the Use of Encryption Mechanisms

For encrypting large amounts of data symmetric cryptosystems are preferable to asymmetric cryptosystems e.g. for the following reasons:

Firstly, symmetric encryption is much more efficient than asymmetric encryption. For example, both hardware and software implementations of DES are currently about 1000 times faster than corresponding RSA implementations (with a reasonable key length).

Secondly, when multicasting a large message it is much more efficient to use symmetric than asymmetric encryption. If the message is encrypted with the public key of the

recipient, a different key has to be used for each recipient, causing the same message to be encrypted and transferred several times. With symmetric encryption, only the symmetric key used needs to be encrypted separately and passed to each recipient, then this one key (now a part of the security context associated with this message) can be used to encrypt (and decrypt) the data which is multicast to all of the recipients. This scheme is adopted e.g. in the *Privacy Enhancement for Internet E-mail* [RFC89a,b,c].

A1.2 Signature Mechanisms

Digital signatures can be used to realize various security functions, including *authentication* (both of *peer entity* and *data origin*), *non-repudiation* (both of *origin* and *delivery*), and *integrity*.

The digital signature of a message is a redundant piece of information which depends on the entire contents of the message in such a way that only the party in possession of the secret signing function can produce it but its validity can be checked without this secret information. The signature (when associated with the message) proves both the origin and integrity of the message providing the function of Non-repudiation of Origin.

Just like with encryption mechanisms, a good digital signature scheme has the property that it is based on a public algorithm and a secret key. Public information is used for checking the signature. It should be computationally unfeasible to generate valid signatures without knowing the secret key as well as to find two different messages with the same signature value.

A PKC is a signature mechanism if the encryption or *signing*, as we may call it now, is secret and the decryption, i.e. the checking of the signature is a public algorithm. When using RSA, the most straight-forward way of signing a message is to encrypt the whole message with the secret key of the sender. In this case we need not send the cleartext message at all. If the message has sufficient redundancy, so that a randomly chosen cryptogram does not decrypt into an acceptable cleartext message with anybody's public key, we can be confident in, and show to an impartial judge, that the message was originated by the holder of the secret key. This kind of signature scheme is presented in [IS9796] *Signature Algorithm with Message Recovery*, which currently is a DIS version.

A more common practice is to send the cleartext message together with the signature. This makes it possible to read the contents of the message at intermediate nodes without having to decrypt the message.

For performance reasons we usually encrypt a hash value based on the message rather than the whole message. The hash function used in computing the hash value has to have certain properties described more closely in e.g. [DP84]. Obviously we must claim the hash function to be such, that it is computationally unfeasible to find two valid cleartext messages with the same hash value.

Hash functions are currently an area of very active research. A proposal for a hash function which is light to compute with software is presented in [Riv90]. Among other recent proposals are Merkle's Snefru function and Miyaguchis's N-hash [MOI90]. Biham and Shamir have shown in [BS90] that it is easy to find synonyms (i.e. different messages with the same hash value) for all of these three algorithms. Also in ISO standard hash functions for digital signatures are being defined [IS10118-1,2]. [Mer89] discusses the use of DES in computing one-way hash functions. DES-based hash functions still appear to be a safe choice for most commercial purposes.

A message is now signed in three steps:

- Calculate the hash value of the message.

- Encrypt the hash value with the secret key of the sender.

- Attach the encrypted hash value (signature) to the original message.

The notation $S_X\{m\}$ is used in this study to mean "message m signed by X". The notation does not stand for the signature but for the message together with its associated signature, hence the notation (use of curly brackets instead of parentheses). The notation $S_X(m)$ is used to denote the signature value of message m generated by X. This practice is consistent with that adopted in e.g. [X.509].

A1.3 Integrity Mechanisms

Data Integrity can be divided into two parts:

- *Message* (or *content*) *integrity* means that each individual message is intact (that is, received in exactly the same form as sent).

- *Sequence integrity* means that the sequence of messages (usually the sequence of messages sent across a connection during its life-time) is intact. This means that no messages have, undetected, been omitted or duplicated and that the original ordering of the messages is preserved.

In [ISO90d] integrity mechanisms are classified as those designed to protect against random modifications and those designed to protect against modifications deliberately engineered to defeat the integrity mechanism. In this study, we shall use the terms *weak* and *strong integrity*, respectively, for these two cases. Data integrity in the weak sense of the word means detecting all incidental (random) changes in the data being transferred, optionally recovering from the changes, when possible, and reporting the cases where recovery is not possible. Weak integrity is normally provided at the Data Link layer point-to-point (e.g. HDLC) and at the Transport Layer end-to-end (e.g. ISO TP4). Protocols of both of these layers are designed to detect and recover from transmission errors (but not planned attacks against data integrity).

For weak message integrity simple redundancy in the data is adequate. Usually a Checksum or, preferably, a Cyclic Redundancy Check (CRC) is used to detect chances in the data. In LANs a 32-bit CRC is normally used and found to be quite effective for this purpose.

Sequence integrity is achieved by labelling the messages with a running sequence number which can be used to detect omitted, duplicated, or out-of-sequence messages. To detect messages omitted at the end of a connection, a graceful disconnect procedure with the negotiation of the last sequence number sent is required before closing the connection.

For purposes of error recovery, retransmission of corrupted messages is used (Automatic Repeat Request, ARQ). In time-critical applications or with unidirectional transmission systems, error-correcting codes, such as Hamming codes, are sometimes used (Forward Error Correction).

Integrity in the strong sense of the word means detecting not only accidental errors but also those deliberately caused by an enemy by means of an active attack. Here simple redundancy added to the messages with public algorithms is not sufficient to guarantee message integrity and the use of sequence numbers is not sufficient to guarantee sequence integrity because both can easily be forged.

In the set-up illustrated in figure 4-2, X can easily modify the data stream between A and B, without this being detected by either communicating party, if only weak integrity is provided. A simple way of doing this is by letting an untrusted intermediate node in the network, such as a router, act as a transport layer gateway between A and B, using the standard transport protocol towards both A and B, relaying the information transferred from one connection to another, modifying the information at its will, and letting both A and B believe that they are communicating end-to-end at the transport layer. The attack is illustrated in figure 4-2 below.

 © Arto Karila, 1991

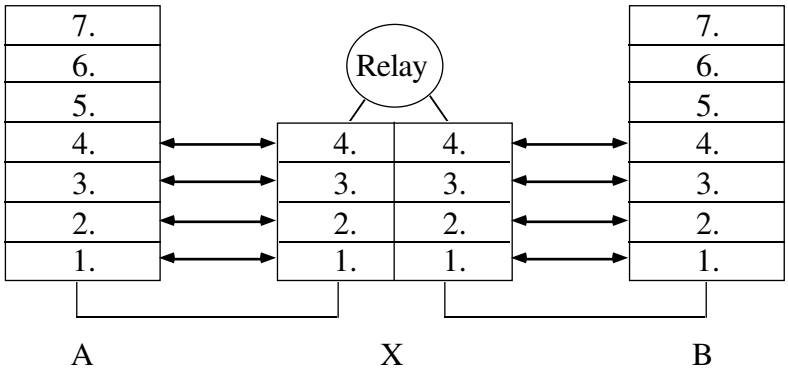

Figure A1-2: Attack against a weak integrity mechanism at the transport layer.

Strong Integrity can be efficiently achieved by using the orthogonal function of *confidentiality* in combination with a *weak integrity mechanism*. If the confidentiality function is placed below the weak integrity mechanism in the OSI Protocol Stack, then the result is strong integrity. This result can be justified with the following reasoning:

- Assuming that the encryption mechanism is an acceptable one and that the enemy does not possess the key used, it is impossible for the enemy to determine anything of the contents of the cleartext message by studying the cryptogram. It is equally impossible for him to generate a cryptogram that would decrypt into anything but a random cleartext message.

- Therefore, from the weak integrity mechanism's point-of-view, any changes introduced by the enemy to the stream of cryptograms are seen as completely random changes in the stream of cleartext messages. But these random changes are just what the weak integrity mechanism was designed to protect against.

It must be noted that there are a number of requirements for an "acceptable" encryption mechanism and weak integrity mechanism as well as for their combination. For example, when a block cipher (with the ECB mode of operation) is used, the weak integrity mechanism must be able to detect errors in the ordering of the encrypted blocks.

If, for example, all TPDUs of TP4 are encrypted, then the Integrity Check Value carried by each TPDU is also encrypted and the TPDUs, including their sequence numbers, cannot be modified by the enemy without this being detected. This makes all attempts to modify the content of individual messages or the sequence of messages rather random and enables the normal functions of TP4 to detect (and attempt to recover from) these modifications.

The situation is clarified in figure 4-3 below. In case a) TP4 is run on top of a noisy channel. This results in reliable transfer on the transport connection between the two Transport Service Access Points. In case b) the channel is not only noisy but hostile, meaning that it causes planned non-random changes in the data stream transferred (e.g. the situation depicted in figure 4-2). This results in unreliable transfer on the transport connection. If end-to-end encryption is introduced below the transport protocol, as depicted in case c), all changes caused to the data stream by enemies not in possession of the encryption key are random and result in the illusion of a noisy channel at the interface between the encryption layer and the actual transport layer. This reduces the situation to that of case a) resulting, again, in reliable transfer on the transport connection.

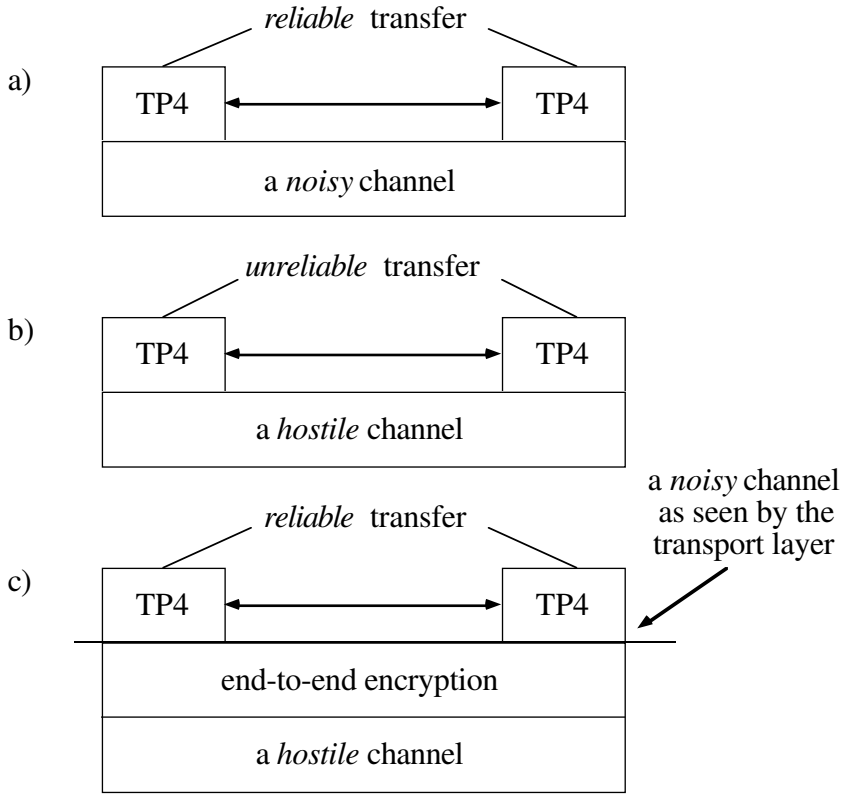

Figure A1-3: Weak integrity combined with encryption produces strong integrity.

In order to prove the reasoning presented above for a given transport protocol – data encryption scheme pair, detailed requirements of the transport protocol (e.g. TP4) to the underlying "channel" (network service) in order to guarantee reliable transport service must first be stated. Then it must be shown, that the hostile channel augmented with the proposed encryption scheme (e.g. DES) does fulfill these requirements. Because the purpose of this work is to create an architectural framework, not to analyze the properties of individual cryptosystems (such as DES) or protocols (such as TP4), the detailed analysis is left for further study.

This revelation gives us one more criterion for placing the functions and mechanisms of Integrity and Confidentiality in the layers of OSI. To avoid unnecessary duplication of functionality, it usually seems to be a good idea to place a confidentiality mechanism somewhere below an existing weak integrity mechanism and get strong integrity as a by-product of these two mechanisms.

S. Walker proposes, in [Wal89], that EESP-like protocols should be placed *above* rather than below the normal transport protocol in order to minimize the overhead caused by encryption and decryption in case of error recovery at the transport layer. It is my belief that efficiency under normal operation is by far more important than efficiency in the rather rare case of retransmission at the transport layer. Obviously this also depends on the quality of the underlying network service.

We can determine that, with some restrictions, a reliable *weak integrity mechanism* when used on top of a reliable *confidentiality mechanism* provides us with the function of *strong integrity*. The problem of which encryption mechanisms are "acceptable" and which weak integrity mechanisms they should be combined with in order to produce strong integrity is a current research topic of cryptography. One basic requirement appears to be that the integrity check value employed by the weak integrity mechanism should be non-linear. It is generally known that block cipher with the OFB mode of operation together with linear redundancy in the cleartext message (for example DES with the OFB mode of operation in connection with Hamming coding) does not produce strong integrity. Integrity mechanisms are also being standardized by ISO. [IS9797] is an example of a standardized strong integrity mechanism which is currently a draft proposal.

A1.4 Authentication

Authentication is divided into two classes:

- *Weak* (or *simple*) *authentication*, where a password but no encryption may (or may not) be used, and

- *Strong authentication*, based on cryptographic techniques.

Weak authentication offers no protection (if not even passwords are used) or inadequate protection against malicious attacks. Weak authentication is currently used in most systems (e.g. practically all operating systems) but, even with passwords, it is very vulnerable to replay. In strong authentication, the user (or service) can prove his identity without revealing his secret piece of information. The need to authenticate a user without him having to reveal his secret authentication information is pointed out

e.g. in [Sim89] where a simple identity verification scheme is presented providing unforgeable transaction receipts for later arbitration.

A requirement for any type of trust in a verifier by a claimant should not be a prerequisite for the correct operation of any authentication mechanism. If passwords are used, then the claimant always has to trust the verifier not to keep and reuse his password to impersonate as him to another entity [IS10181-2]. This observation alone is sufficient for ruling out the use of passwords as an acceptable general solution to authentication. It has also been shown (e.g. in [MT79]) that password-based authentication schemes can never be made completely reliable. Therefore, in this study, only strong authentication is discussed, since weak authentication clearly does not offer adequate security for our needs.

Strong authentication can be based on symmetric or public key cryptography. With symmetric cryptosystems a mutually agreed-upon pairwise key belonging to the appropriate security context is needed for authentication between any pair of parties A and B. Therefore, with n communicating parties n(n-1)/2 keys are needed. For large values of n this is clearly impractical. The number of keys required is too large to be securely generated, transferred and stored, every party having to be in possession of n-1 pairwise keys.

Public key signature mechanisms have several advantages over symmetric cryptosystems when used for authentication. Key management is greatly simplified by the fact that only public keys of the key pairs need to be shared and only one key pair is needed for each party. Also, since the secret key is only known to one party, public key signatures can be used for non-repudiation.

Even though symmetric encryption schemes can be used for authentication they cannot easily be used for the stronger service of non-repudiation. Because a symmetric communication key always has to be shared by at least two parties, it is impossible to prove to an impartial judge the origin of a message signed with an algorithm based on symmetric encryption without relying on a trusted server of some kind, such as an on-line notary. If A produces a message signed with a pairwise key between A and B, claiming that this message was created by B, B can always claim that the message was forged by A, A also being in possession of the same key. In a case such as this, only A and B know the truth but neither can convince the judge. Some signature schemes based on symmetric encryption are presented in e.g. [Akl83] and [Dav83] but they are rather artificial (i.e. they "consume" keys when used) and not very well suited for our purposes.

From the authentication point-of-view we simply need a trusted certification authority whose certificates can be used to verify the integrity of other parties' public keys. The way in which these certificates are distributed is a management issue. The X.500

Directory Service (more precisely X.509) provides us with a natural means of distributing these certificates. It is worth pointing out (again) that the directory itself need not be trusted, we only need to trust our CA.

Authentication Protocols

Authentication protocols have been widely studied (see e.g. [NS78, DS81, NS87, DH79, Moo88, IM90]). Often the need for three-way handshake for reliable mutual authentication has been pointed out. The main motivation for a three-way handshake mechanism is the need for each party to challenge the other.

The mechanisms included in the ACSE Security Addendum can be used to implement the authentication procedure describe above. This arrangement allows the client (initiator) to challenge the server (responder) by having him increment the time stamp by one. However, the server does not get to challenge the client. This is usually considered a shortcoming in ACSE's authentication mechanism, since three-way handshake was ruled out of ACSE in ISO.

The idea of exchanging challenges is that in this way each party can make sure that the authentication message received from the other party was not a replay. However, it is not the exchange of challenges per se but rather the *freshness* of the tokens that is required. With the freshness of a token we mean that it was generated for this purpose and has not been used before. It is shown below that two-way handshake is adequate for reliable mutual authentication when used with a time stamping mechanism. The principles presented here are nowadays widely known and accepted, the interested reader is referred to e.g. [IS9798-1,2,3] and [IS9799]. However, a brief summary of the principles of authentication based on a two-way handshake protocol is given here.

Assume that:

1) Each authentication message from A to B includes $S_A\{A, B, t\}$ (sender, recipient and time stamp, signed by the sender).

2) B is responsible for storing all authentication messages still valid (those, whose time stamps have not yet expired) and detecting attempted replay.

3) The signature scheme is reliable (only A can produce an acceptable message signed by A).

Conclusion: two-way authentication is adequate, because:

1) Message $S_A\{A, B, t\}$ is only accepted by B (the indicated recipient).

2) It is only accepted if it is fresh (generated by A for this purpose), i.e. the time stamp is still valid and B has not received the same message before.

3) B knows that the message was *for him*, *from A* wishing to authenticate *now*.

4) Likewise, A knows that the response from B: $S_B\{A, B, t+1\}$ was *for him*, *from B* and a reply to his request.

The fields for sender and recipient (A and B) included in the signed portion prevent the use of a captured message anywhere except between A and B. The time stamp values (t and t+1, where the time unit is the resolution of the time stamp used, i.e. one "tick") link the two authentication messages together and, when combined with the obligation of B to detect replay, make up for the lack of challenges chosen by A and B. The order of A and B (not implied with this notation) prevents the use of a response as a request later. These conditions appear to be sufficient for the conclusion that A and B unquestionably have mutually authenticated.

These conditions are thought to be sufficient but not necessarily minimal. As demonstrated by the error in X.509 [BAN89, IM90] the strive towards minimality can be dangerous, and it is by no means very useful. The conditions stated above are, in any case, reasonable and can be fulfilled without excessive over-head.

In case public key signature mechanisms are used for purposes of authentication (which most often is the case), it is necessary to include the recipient in the signed portion of the message to avoid the same message being used for another purpose later (referring, again, e.g. to the X.509 attack in [BAN89]). However, if symmetric cryptosystem and pairwise keys are used for authentication, it is possible to do without the Sender and Recipient fields in the signed portion of the message. That is because a given authentication key is only valid between a certain pair of communicating parties (such as A and B) and replay is impossible outside this Security Context. Here too, it must be assured that a reply cannot be used as a request later. With public key cryptosystems authentication message originally sent from A to B could later be used by B trying to impersonate as A to C if the recipient was not included. The necessity of the sender field can always be argued, it is included mainly for consistency.

The only question left open is whether, after the authentication exchange, both A and B know that a connection has been opened and a session key (completing the common Security Context) has been agreed upon. Obviously, after two-way handshake, B cannot know that A has received and accepted his last message. The problem is analogous to that found in establishing a transport connection. Also the solution is the same.

 © Arto Karila, 1991

We can require that A sends the first (encrypted) message on the newly established connection as soon as it receives the response from B. Authentication is only considered completed after one more message is passed from A to B. This third message, after the two previously mentioned authentication messages, concludes a kind of three-way handshake. If B does not receive this message within a reasonable time, a time-out occurs and the connection is aborted.

A more formal proof of the adequacy of this two-way authentication protocol could be given by using the "Logic for Authentication" presented in [BAN89]. However, this logic is still at its early stage (as pointed out e.g. in [Nes90]) and does not (yet) fulfill the requirements for a serious mathematical formalism. Substantial extension to the BAN logic are proposed in [GNY90]. It is also used in [CG90] to analyze a secure multi-party session protocol. While it is evident that this kind of formalism is needed in developing and analyzing security protocols, a ready-to-use formalism does not currently exist and developing one is a major undertaking and clearly beyond the scope of this study.

The authentication function per se does not protect against an untrusted party within the network intercepting the connection later, after authentication, and impersonating as A to B and vice versa. However, if a pairwise symmetric session key is securely exchanged between A and B during the authentication exchange and used in subsequent communication, e.g. in the way presented in figure 4-3, this threat can be protected against (an authentication protocol that enables us to securely exchange secret information in connection with the authentication exchange is called a *subliminal channel*). This session key can be generated by either of the communicating parties alone or both together.

The communicating parties can authenticate each other (both at the end system level and at the user level) in the following manner. The process is initiated by user U logging in a service. The initial log-in is omitted in the following description, which concentrates on the security related aspects of the communications sequence.

The application entity A_U serving user U forms token $T_U = \{U,A_U,S,A_S,TD\}$ and signs it with its public key. If authentication of the user is required, the message is handed over to be signed by U. A_U then sends the signed token over to A_S. A_S (and S) can now authenticate both U and A_U by checking that the signatures are intact and the time stamp is valid. This procedure can be performed without U giving its secret key to A_U or vice versa.

In a similar fashion, the server S and its application entity A_S can identify themselves to U and A_U without trusting each other with their secret keys. This makes it possible to use, for example, a smart card for authentication of the user.

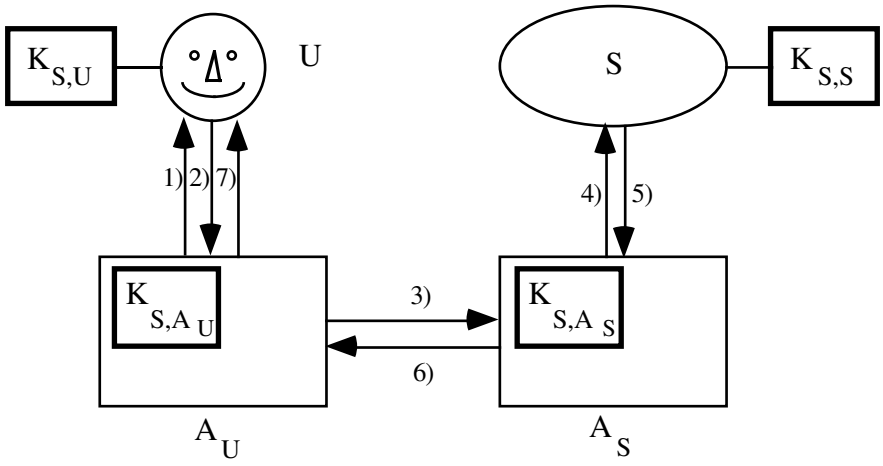

Figure 4-4: Strong two-way authentication.

Strong two-way authentication goes as follows:

A_U generates token T_U={U,A_U,S,A_S,TD}, where TD is a time stamp, and passes it on to U:

1) T_U

U signs T_U (with its secret key $K_{S,U}$) and responds with:

2) $S_U\{T_U\}$

A_U co-signs the result (with its secret key K_{S,A_U}) and sends the signed message over to A_S:

3) $S_{A_U}\{S_U\{T_U\}\}$

A_S checks the signature of A_U, thereby verifying its identity. A_S also checks the freshness of T_U by checking that the time stamp TD is still valid and (if it is) that the same token has not been received before. A_S now generates token: T_S={S,A_S,U,A_U,TD+1} and passes it on to S together with T_U signed by U:

4) $\{T_S, S_U\{T_U\}\}$

S checks the signature of U, signs T_S (with its secret key $K_{S,S}$). S can also generate a pair-wise fresh session key, encrypt it with the secret key of U, and include it in T_S before signing, yielding: T_S={S,A_S,U,A_U,TD+1,e$K_{P,U}$(K_{AB})}, as described in appendix 3. S now responds with:

5) $S_S\{T_S\}$

A_S co-signs the result (with its secret key $K_{S,AS}$) and sends the signed message over to A_U:

6) $S_{AS}\{S_S\{T_S\}\}$

A_U checks the signature of A_S, thereby verifying its identity. A_U also checks that the value of the time stamp in T_S is TD+1 (where TD is the time stamp of the corresponding T_U) and that the time stamp is still valid (not too much time has elapsed). A_U then passes T_S signed by S on to U for it to check the signature:

7) $S_S\{T_S\}$

U checks the signature of S completing the authentication cycle. If T_S includes the encrypted pair-wise session key $eK_{P,U}(K_{AB})$, U recovers the key by decrypting it with its secret key $K_{S,U}$.

Confidential encrypted information can be included in the signed portions of the authentication messages. One way of doing this is by using a PKC and the key pair of U (the calling party) in message 6), as describe above and in appendix 3. In this case the pairwise session key is generated by the service, which usually is a good approach when using a smart card with limited memory and processing capacity for user authentication.

It is also possible to have the calling party U generate the session key. In this case the PKC key pair of S (the called party) is used in message 3). In message 6) either a PKC and the key pair of U or a symmetric cryptosystem and the session key included in message 3) can be used for confidentiality. The latter approach has two benefits:

- By using the pairwise key included in message 3) S acknowledges this session key and U can now be sure that S agrees on the use of this key.

- Using a symmetric cryptosystem is computationally more efficient than using a PKC.

If a fresh, pairwise session key is securely exchanged in connection with the authentication procedure, we can be confident that only the two mutually authenticated parties are in possession of this key. This key can then be used for purposes of continued authentication, integrity and confidentiality over this session, e.g. in the way illustrated in figure 4-3.

 © Arto Karila, 1991

Two considerations regarding the exchange of a fresh session key in connection with the authentication exchange are worth pointing out:

- The session key must be signed by the sender, that is it must be included in the portion of the message affecting the value of the signature.

- Even though the session key is always sent in an encrypted form, the signature function should preferably be performed before the encryption, because signing encrypted data is always considered a bad practise as shown e.g. in [MWR89] or [IS9798-3].

The authentication protocol described above can be realized by using a symmetric block cipher (such as DES) and a PKC (such as RSA) in the following way:

- The symmetric cryptosystem is used to calculate the hash function h (e.g. with Cipher Block Chaining mode of operation).

- The PKC is used for encrypting and decrypting the hash values. For this, the encryption key is kept secret while the decryption key is made public.

The application entity A_U serving User U forms token $T_U = \{U,A_U,S,A_S,TD\}$ and signs it. A_U then sends the signed token to A_S. A_S and S can now authenticate U and A_U, respectively, by checking that both of the signatures match and the time stamp TD is valid. Mutual authentication is achieved by having A_S and S perform the symmetric operation.

Mutual strong two-way authentication now proceeds as follows (see also figure 4-3):

A_U generates token $T_U = \{U,A_U,S,A_S,TD\}$ and passes it on to U:

1) T_U

U signs T_U, by calculating $h(T_U)$, encrypting this with its secret key $K_{S,U}$, and appending this signature to T_U, and returns the signed token to A_S:

2) $\{T_U, eK_{S,U}(h(T_U))\}$

A_U co-signs T_U by encrypting the signature of U with its secret key $K_{S,AU}$: $X_U = eK_{S,AU}(eK_{S,U}(h(T_U)))$ and sends this signature X_U, together with T_U, over to A_S.

3) $\{T_U, X_U\}$

 © Arto Karila, 1991

A_S decrypts the signature with the public key of A_U recovering the signature of U. A_S also checks the freshness of T_U by checking that the time stamp TD is still valid and (if it is) that the same token has not been received before. A_S now generates token: T_S = {S,A_S,U,A_U,TD+1} and passes it on to S together with T_U and the signature of U:

4) $\{T_S, T_U, dK_{P,U}(X_U)\}$

S checks the signature of U by decrypting $dK_{P,U}(X_U)$ with the secret key of U and comparing the result with $h(T_U)$. S now signs T_S by encrypting $h(T_S)$ with its secret key $K_{S,S}$, appends this signature to T_S, and passes the signed token back to A_S. Optionally S can generate a pair-wise session key K_{AB}, encrypt it with the public key of U, and append it to T_S before signing (this seems to be one of the rare cases, where signing encrypted data is justified). This means substituting T_S with $\{T_S,eK_{P,U}(K_{AB})\}$ in the message below (the optional part is denoted with angle brackets):

5) $\{\{T_S\ [,eK_{P,U}(K_{AB})]\}, eK_{S,S}\{h(T_S\ [,eK_{P,U}(K_{AB})])\}\}$

A_S co-signs T_S by encrypting $eK_{S,S}(h\{T_S[,eK_{P,U}(K_{AB})]\})$ with its secret key $K_{S,AS}$: X_S=$eK_{S,AS}(eK_{S,S}(h\{T_S[,eK_{P,U}(K_{AB})]\}))$ and sends the resulting signature X_S, together with T_S, over to A_U:

6) $\{\{T_S\ [,eK_{P,U}(K_{AB})]\}, X_S\}$

A_U checks the signature of A_S, by decrypting X_S with the public key of A_S and recovering the signature of S: $eK_{S,S}(h\{T_S[,eK_{P,U}(K_{AB})]\})$. A_S also checks that the value of time stamp in T_S is TD+1 (where TD is the time stamp of the corresponding T_U) and that the time stamp is still valid (not too much time has elapsed). A_S now passes T_S (together with the optional encrypted pair-wise session key) and the signature of S and on to U:

7) $\{\{T_S\ [,eK_{P,U}(K_{AB})]\}, dK_{P,AS}(X_S)\}$

U checks the signature of S by decrypting it with the public key of S and verifying that $h(T_S[,eK_{P,U}(K_{AB})]) = dK_{P,S}(dK_{P,AS}(X_S))$, completing the authentication cycle.

The key exchange scheme presented here is by no means the only possible way of securely exchanging a fresh, pair-wise session key in connection with the authentication exchange. It is, for example, possible to have U (instead of S) generate the key or exchange the key on application (rather than user) level.

However, the scheme presented here appears to be well suited for user identification based on a smart card. Here the burden of key generation is left entirely to the service

 © Arto Karila, 1991

which has better facilities for this purpose. Also, if the smart card has an integrated symmetric encryption mechanism, the key can easily be loaded onto the smart card and decrypted there without it ever leaving the card in clear-text form.

A1.5 Zero-Knowledge Techniques

A rapidly growing area is that of zero-knowledge techniques. In these techniques, the secret authentication information of each party plays very much the same role as the secret key in PKCs but it cannot be used for data encryption, only for authentication (and possibly digital signatures). In some zero-knowledge schemes, unlike in PKCs, the "public key" needed to verify an entity can be the same for all entities belonging to the domain of the same CA. Another typical characteristic of current zero-knowledge schemes is, that they often use several iteration cycles of challenges and replies to them to achieve the required level of confidence in the identity of the other party.

A class of authentication schemes, where the the *claimant* can prove his identity to the *verifier* without revealing a single bit of his secret authentication information, is known as Zero-knowledge techniques. Perhaps the nicest property of Zero-knowledge techniques is that they eliminate the need to periodically change the authentication information in order to protect against cryptanalysis, because the verifier (or a casual eavesdropper) cannot learn anything in the process that would help him masquerade as the claimant later. The first zero-knowledge technique was proposed by Shamir in 1984 [Sha85]. Another important scheme is presented in [FFS88]. [QG90] gives an easy-to-read introduction to zero-knowledge protocols.

In [IS10181-2] zero-knowledge techniques are (falsely) defined as a class of procedures adequate for authentication of single principal or for signature, using a single value of verification authentication information for a set of principals. This property of zero-knowledge techniques makes key distribution much easier than with authentication based on PKCs, where a separate piece of verification authentication information (namely the public key) is needed for each principal. Unfortunately, this nice property is not shared by all zero-knowledge schemes.

Zero-knowledge techniques can be used for purposes of authentication or non-repudiation but usually not for data encryption. Zero-knowledge techniques are most often computationally less complex than PKCs which makes them attractive for user authentication with a smart card. Typical properties of zero-knowledge techniques, according to [FP90], are the following:

- The security level of the verifier depends on the number of iteration cycles of the basic protocol.

- The claimant protects his secret key by giving the verifier at every iteration cycle access only to a randomly selected part of the key.

- One-way functions are used to minimize information flow.

- The verifier does not learn anything during the authentication process that would enable him to impersonate as the claimant later.

Some existing zero-knowledge techniques make key management very simple by completely abolishing the need for user-dependent public keys. The draw-back of these techniques (in comparison with PKCs) is, that they require the secret keys to be generated by a trusted third party (the CA) and they cannot be used for data confidentiality. An example of such an authentication technique is the Fiat-Shamir technique first presented in 1986 [FS87].

A1.6 Physical Mechanisms

In the case of cryptography, the physical mechanisms at the bottom of the hierarchy that are needed to actually perform the cryptographic functions employed can be pieces of software (running on a piece of hardware) or hardware.

Usually the two lowest layers of OSI are implemented in hardware (the Physical layer and the Medium Access Control sublayer of the Data Link layer) whereas the upper layers (from the Logical Link Control sublayer of the Data Link layer up) are implemented in software. However, dedicated hardware can be used at any layer of the OSI model to perform computation-intensive functions, such as cryptography.

The physical mechanisms used have an impact on the over-all security of the system because an implementation of an open system cannot be more secure than the weakest physical mechanism, or the weakest path between these mechanisms, used in the trusted parts of the system. A more detailed analysis of the security of physical mechanisms falls in the area of local systems security and is beyond the scope of this study.

With the transmission speeds offered by current data networks, the efficiency of the physical mechanisms used can become a major issue of systems design. The choice between various physical mechanisms is a trade-off between economy, flexibility, and performance. Therefore, different solutions will be used in different applications. The choice of these mechanisms does usually not affect the external behavior of the system except performance-wise.

However, new portable trusted pieces of software open up new opportunities for the application of secure open systems. The smart card, or its less compact calculator-like version with an integrated keyboard and display, is the only reliable way for strong authentication of the user. There is a very large potential for the use of such devices.

 © Arto Karila, 1991

A2 Some Commonly Used Cryptosystems

This work is independent of any individual cryptosystems or other low-level mechanisms. In the preceding chapters we have simply assumed that we have at our disposal symmetric and asymmetric cryptosystems with certain properties. In a constant struggle between cryptographers and cryptanalysts to make and break new cryptosystems existing systems will die and new systems be born. It is important that this work shall not be tied to any currently existing schemes.

However, since we are interested in actual implementations of secure open systems it is important that we be able to specify some current cryptosystems fulfilling the requirements set before. In this chapter we will have a brief look at some cryptosystems which, for the time being, are considered suitable for our purposes.

Here we shall not present the details of these cryptosystems but rather concentrate on their applicability to our purposes. Any good tutorial on cryptology, such as [DH79], [DP84], [Mas88] or [Pat87], will discuss these algorithms in detail.

The schemes presented here are all well known and well established. They have all resisted cryptanalytic attacks for a number of years and are still considered reasonably safe although many of the later cryptosystems have been broken. See e.g. [BO88] for an overview of cryptanalytic attacks against these three cryptosystems as well as some others.

A2.1 DES

Those interested in the history, current use and future prospects of DES are advised to read [SB88] for an overview of the topic.

In 1973 the US National Bureau of Standards (NBS) initiated the development of the Data Encryption Standard (DES) by publishing a solicitation for encryption algorithms for computer data protection. The US National Security Agency (NSA) was requested to assist in the evaluation of proposed encryption schemes.

IBM had started work on developing cryptosystems in the late 60's which had already resulted in commercial applications. In the early 70's the work was continued and it led to a number of patents and products. IBM submitted a cryptographic algorithm to NBS in response to the solicitation and this algorithm was to become DES.

From the very beginning there was a lot of concern about the security of DES. The main grounds for criticism were the following:

 © Arto Karila, 1991

Firstly, the 56-bit key-length of DES was not considered adequate by everybody and for all kinds of use. In 1975 Diffie and Hellman proposed a scheme for breaking DES by using massive parallelism and exhaustive search of keys. This would make breaking DES quite costly but possible for organizations such as intelligence services. This scheme is presented in [DH77].

Secondly, the design criteria of the eight substitution tables (called "S-boxes") used in DES were never made public. This led many to believe that there was an intentional trapdoor in DES making it very easy break for NBS and NSA, and very hard for everybody else.

Before accepting the standard, NBS organized two workshops in order to study the security of the proposed algorithm. One was on mathematics and the other on the economic trade-offs of increasing the key-length. Based on the results of these workshops, NBS decided to pass the standard without modifications but it was to be reviewed every five years. The standard was last reviewed in 1988 and passed for another five years. The next revision process will start in early 1992.

Despite extensive cryptanalysis during the past fifteen years and the proposed attacks nobody has to date been able to show a flaw in the design of DES. All of the serious attacks are based on exhaustive search. DES is therefore still considered a reasonably safe algorithm for commercial (but not for military) use. Some of the most recent cryptanalytic studies (see e.g. [BS90]) seem to indicate that the standard 16-round DES still is reasonably safe – in fact much safer than many of its more recent rivals, such as the Japanese FEAL algorithm [SM87].

DES is a symmetric block cipher with a block length of 64 bits and effective key length of 56 bits. The DES standards [ANS81, ANS83] define the basic 64-bit block cipher and four modes of operation for it. These modes are called: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB). Of these modes, the first one is the basic block cipher and the latter three turn DES into a chained "stream cipher". CBC and OFB are well suited for calculation of cryptosums (e.g. integrity check values) and CFB is good for encrypting individual characters. These modes of operation can also be used with any other block cipher, such as FEAL. Similar definitions now also exist as international standards, [IS8372] defines the modes of operation for 64-bit and [IS10116] for N-bit block ciphers.

As a summary, it can be stated that among the good properties of DES are the following: it is a commonly accepted standard encryption scheme with lots of efficient hardware implementations available, it is computationally reasonably light, it has several standardized modes of operation suited for various purposes, and its safety is still believed to be adequate for most purposes.

 © Arto Karila, 1991

Among the shortcomings of DES are the following: the key length is insufficient by today's standards, it is still rumored to contain trapdoors, even though the most recent studies [BS90] don't seem to support this belief, and the initial and final permutations of DES decrease the efficiency of its software implementations.

It is expected that DES will survive yet for a long time in commercial applications, even though in has already been replaced by later, more secure cryptosystems, such as CCEP, in the most critical (e.g. defense-related) applications.

A2.2 RSA

The Rivest-Shamir-Adleman (RSA) algorithm first presented in [RSA78] is the most commonly used and probably most usable Public Key Cryptosystem today. RSA is based on the exponentiation of primes (modulo n) and breaking it is as hard as finding the logarithm (modulo n) of an integer, which is believed to be as hard as factoring a large integer. With key lengths of at least 500 bits RSA is still believed to be reasonably and with about 1000 bits quite safe.

RSA is fairly simple, safe, and adaptive to various kinds of use. Among its shortcomings are its large (and variable) block length and heaviness (RSA is about three orders of magnitude heavier to compute than DES).

As we can see, RSA can easily be used for generating digital signatures. However, the group of possible future algorithms meeting one of the two requirements stated above is presumably much larger than just RSA and we can feel confident that generating digital signatures using a PKC as described here does not limit us into using RSA.

Because breaking RSA is at most as hard as factoring a large integer, some recent advances in the factoring of large integers have made the use of RSA less secure and increased the needed key length. Considering the possibility of a final break-through in factoring, we cannot base everything (authentication, non-repudiation, key distribution etc.) on RSA but have to be ready to quickly adopt new PKCs and authentication schemes if necessary.

 © Arto Karila, 1991

A2.3 The Diffie-Hellman Algorithm

The Diffie-Hellman scheme, first proposed in [DH76] as the first published "public-key" algorithm, is still considered one of the best methods for secretly sharing pairwise symmetric keys. The algorithm is based on public "half-keys" and secret values associated with them. From their public half-keys K_A and K_B the communicating parties A and B can determine a pairwise session key K_{AB}, which remains secret from other parties. This key can then be used for mutual authentication and or exchanging secret information (such as a fresh session key).

In fact this algorithm is not a data encryption scheme but a key management scheme where n parties can secretly agree on $n \cdot (n-1)/2$ pairwise encryption keys using only n public "half-keys". Normally $n \cdot (n-1)/2$ pairwise keys would have be created and n-1 keys securely delivered and stored at each site in order to allow for secure communications between any two sites. When using the Diffie-Hellman algorithm only authenticated and integral (not confidential) copies of n-1 public "half-keys" have to be delivered to and a site-specific (locally created) secret number securely stored at each site.

One of the problems associated with the Diffie-Hellman scheme is, that all the parties wishing to communicate with one-another have to use the same parameters values (namely the generator and modulus) of the cryptosystem. Obviously this is not possible on a global scale. Another problem with the Diffie-Hellman algorithm is that, since the master key K_{AB} between A and B is shared by both parties, non-repudiation is not possible without employing a trusted third-party.

Furthermore, with this algorithm each user in the group needs to have an integral copy of every other user's public authentication information in order to establish secure communication with him. This public authentication information either has to be distributed through an authenticated and integral channel or certified by a CA (which requires the use of a public key signature scheme).

These restrictions together make the Diffie-Hellman algorithm only suitable for secure communications in a closed, reasonably sized, fairly stable user group where it is possible to agree on the common parameters and for all users to keep the integral copies of the public authentication information of all the users. Despite its shortcomings, the Diffie-Hellman algorithm is the basis of public key cryptography and even some recent publications, such as [OT89], do propose authentication and key distribution systems based on it.

 © Arto Karila, 1991

A3 The Kerberos Authentication Server

Authentication protocols have been widely studied for a long time and it is not our purpose to present a tutorial here. The interested reader is referred to the classical papers [NS78, DS81, NS87, OR87] or any good tutorial on cryptology.

Perhaps the most prominent strong authentication service in wide use today is the Kerberos Authentication Server created in the Athena project at MIT [SNC88]. Kerberos is in everyday use in several major universities (including MIT and Carnegie-Mellon University) and obviously has solved a number of security problems in them. It has also recently been adopted as a standard part of distributions of the DEC Ultrix operating system. Therefore, its significance should not be underestimated. However, it deserves to be pointed out that the approach taken in this work is fundamentally different from that of Kerberos in several respects and is assumed to supersede the currently existing authentication schemes.

In Kerberos authentication is based on symmetric encryption which precludes the stronger service of non-repudiation and leads to the problems of key management described in chapter 4.1.3. Obviously, in an academic environment, repudiation is not considered a serious threat. The way in which key management has been solved in Kerberos is interesting and deserves a brief summary.

To avoid the need for $O(n^2)$ pairwise master keys, each communicating party shares one master key with Kerberos. Kerberos, therefore, is in possession of all the master keys. When using Kerberos, the authentication process proceeds as follows:

1) Client C first asks for a certificate (called "ticket") from Kerberos by sending it the message: "C, TGS"

2) Kerberos authenticates the user by his master key and prevents replay by using time-stamps. Kerberos now passes the ticket for a "Ticket Granting Server" (TGS) to C. This ticket includes a pairwise fresh session key to be used between the C and TGS:
$T_{C,TGS}$: TGS, C, Addr, TimeStamp, LifeTime, $K_{C,TGS}$
"$eK_C(K_{C,TGS}, eK_{TGS}(T_{C,TGS}))$"

3) User C now identifies himself to the TGS by presenting his ticket together with an "authenticator" A_C and asks for another ticket to the actual service S he wants to access.
A_C: C, Addr, TimeStamp
"S, $eK_{TGS}(T_{C,TGS})$, $eK_{C,TGS}(A_C)$"

 © Arto Karila, 1991

4) TGS checks the validity of the ticket and uses the timestamp of the authenticator to detect attempts of replay. TGS now gives C a ticket to S. This ticket includes a pairwise session key to be used between the C and S.
$T_{C,S}$: "$eK_{C,TGS}(eK_S(T_{C,S}), K_{C,S})$"

5) The user can now access the service by presenting his ticket together with an authenticator.
A_C:"$eK_S(T_{C,S})$, $eK_{C,S}(A_C)$"

While this approach works well in limited environments, it still has a number of shortcomings:

- Kerberos is in possession of all the master keys and can therefore impersonate as any of the other parties. Placing this much trust in any one party causes considerable security threats.

- As the number of users and services grows, it will become unfeasible to manage the master keys.

In our approach we only need to trust our own CA and the CAs certified by it. In particular, none of the on-line servers needs to be trusted. Certificates forming an unbroken chain of trust can be received from any untrusted source, such as the Directory. As soon as we know the public keys of our peer, authentication can be done on a bilateral basis between the two communicating end-systems without involving anybody else in the process.

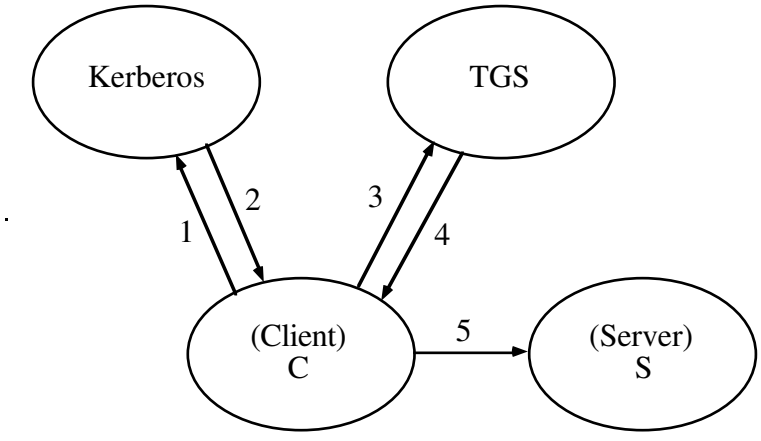

Figure A3-1: The Kerberos Authentication Server.

 © Arto Karila, 1991

A4 Security Policy

Security policy is a statement, made with the knowledge and assistance of top management, that says, in very general terms, what is and what is not to be permitted in the area of security during the general operation of the system being secured. That is, a security policy tells "what" rather than "how" is to be done [IS7498-2].

Security policies are conventionally divided into *Identity-Based* and *Rule-Based* policies:

Identity-based security policies are based on privileges or capabilities given to users and/or Access Control Lists associated with data items and other resources. In a rule-based security policy, Security Classes are normally used for determining what is authorized behavior. In identity-based systems, the user traditionally identifies himself by presenting to the system something he knows (e.g. a password). This is often called "need to know" policy.

In rule-based security policies, information and other resources are conventionally divided into *security classes* and the flow of information between (and within) various classes is controlled by a set of rules. There are also rules regarding the security relations between various resources. A classical rule-based security model is the Bell-LaPadula model, based on finite state machines and security classes. The model was first presented in 1973 and a refined version can be found in [BP74].

It should be noted, that besides security classes there is the orthogonal dimension of integrity by which resources can be divided into *integrity classes* first introduced by Biba in [Bib77]. Combining the ideas of Bell, LaPadula and Biba, we can state the following general rules for information flow in a rule-based secure system:

- A subject can only read an object, whose security level is the same or less than that of the subject.

- A subject can only write to an object, whose security level is the same or greater than that of the subject.

- A subject can only read an object, whose integrity level is the same or greater than that of the subject.

- A subject can only write to an object, whose integrity level is the same or less than that of the subject.

 © Arto Karila, 1991

A recent paper on rule-based security policies in a distributed environment is [LS90]. Some classical papers on security policies are [Den76] and [Lan81]. An overview of the topic can be found e.g. in [IS7498-2].

In a system everything, including processes, data items and channels, are resources and can be labeled with attributes. Security policies indicate how these attributes are to be used to provide required level of security. The rules are stored in a Security Management Information Base (SMIB).

It is only after an explicit security policy has been stated that security becomes an engineering problem and every organization seriously interested in security should one. The enforcement of this security policy and monitoring of security related events lies in the domain of engineering and means for doing this are discussed in this study.

Security finally boils down into an economic issue. Security is on its optimal level when the sum of total losses caused by security defects and the cost of providing security is at its minimum. The losses caused by security violations are usually very difficult to quantify. However, the optimal security level always lies somewhere in between the two extremes. With no security measures losses can easily grow large. If flawless security is pursued the cost of providing security grows out of all reasonable bounds. The situation is depicted in figure A4-1.

While exactly locating the optimum is impossible, careful risk assessment should be used to approximately locate it. In drawing up a security policy and performing risk assessment also non-technical security measures should be considered. Without appropriate measures e.g. in the areas of Personnel Policy and Physical Security all technical efforts to ensure security are in vain, the security chain always being as strong as its weakest link.

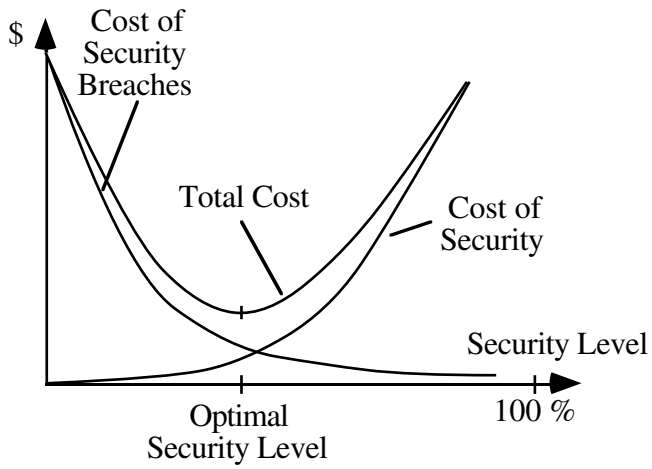

Figure A4-1: Optimal security level.

 © Arto Karila, 1991