CHAPTER 13: SECURITY

Delta-4 being open and compatible with current operating systems and applications, the basic Delta-4 security is provided by the local operating systems such as Unix. But, in some distributed contexts, this basic security is not sufficient, e.g. because some individual sites or operators cannot be trusted, and/or because the information processed by the distributed system is very sensitive. This chapter presents some solutions to provide secure distributed application services, such as a file archiving service, and to manage the security of a Delta-4 system in such a way that an intrusion into a part of the distributed system will not endanger its security, i.e. in an intrusion-tolerant way.

13.1 Introduction

Computing system dependability has been classically considered as being the ability of a computing system to deal with faults that are, implicitly, seen as being accidental. Security, i.e. the ability of a computing system to deal with intentional attacks, was historically a somewhat later worry than "classical" dependability. Such intentional attacks are becoming more and more numerous and subsequent losses are ever-increasing, causing more and more trouble in computing centres, banking systems, etc. This is particularly true in large network environments. The numerous cases of computer fraud that are related in the newspapers underline the fact that although quite a lot of techniques for achieving security have been developed, the problems of computing system security have not really been satisfactorily solved.

Until now, security and dependability have had their own meetings, terminologies, standardisation committees... and very few research groups work on both aspects. The quite evident relationship that exists between security and dependability has not been clearly formalised. Nevertheless, recent attempts at relating these two domains tend to consider dependability as a general concept that embraces security, reliability and safety as different attributes. This viewpoint is not only conceptually interesting but should lead to the application to intentional faults of concepts originally intended for accidental faults, and vice versa. On the other hand, designing systems that are both reliable and secure, using homogeneous technical solutions is an exciting challenge while these two requirements are commonly considered as being antagonistic.

Dependability can be more rigourously defined as that property of a computing system which allows reliance to be justifiably placed on the service it delivers (cf. chapter 4). This general but precise definition does not need any specification of the impairments or faults that may lead to a system failure. These faults may be either accidental ("classical" dependability) or intentional (security). This is why we believe that one has to consider dependability as a general concept, involving all kinds of faults. Reliability and safety may also require security in the sense that an intentional action may alter the mechanisms intended, for example, to ensure system reliability. It is also true that security requires reliability because an accidental fault in a security mechanism may allow intrusions that are usually not possible. As we can see, it is not possible to integrate one aspect into the other as is sometimes attempted, but we have to consider these concepts at the same level with tight relations linking one with each other. Figure 13.1 shows a conceptual organisation of dependability as a generic concept (cf. chapter 4).

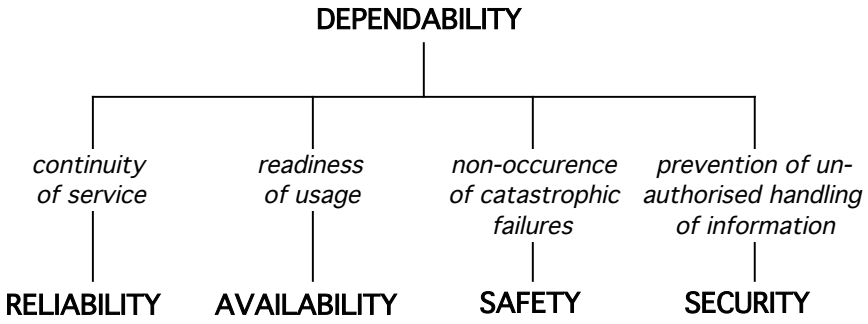

Figure 13.1: Perceptive attributes of dependability

Impairments to security come primarily from intentional faults, which may be either design faults or interaction faults. Intentional design faults (usually called *malicious logic*) are introduced into the system so that it delivers a service that is not as described in the specifications in order to mislead the user about the operations that he performs with the system. These illicit functions are often carried out without being detected by the user who may continue to work with a bad system for a long time before discovering the mischief. Interaction faults are performed while the system is already in use, taking advantage of the weaknesses in the security mechanisms or policy. They are usually called intrusions. Both classes of intentional faults may cause a failure of the system. Their effects depend on the efficiency of the mechanisms designed to ensure security.

13.2 Principles of intrusion-tolerance

13.2.1 General concept

An *intrusion* can be defined as an *intentional operational external fault* (cf. chapter 4). Different types of intrusions can be classified according to who makes the intrusion:

- It can be somebody outside the system who tries to access it. This is the most well known kind of intruder, but not the most important. In this case the intruder has to *by-pass* physical, procedural and logical protections.
- The second kind of intruder is a user of the system who tries to access information or services for which he has no access right. The intruder tries to *extend his privileges*. This is the most common intrusion. The intruder has "only" to by-pass the logical protection.
- The third - and the most dangerous - type of intruder is a security administrator who *abuses his privilege* in order to perform illegitimate actions. In this latter case, the administrator has enough access-rights to do these actions but, according to the security policy, is not supposed to do them.

Intrusions can be treated with the same means as for other faults, i.e. by means of fault-avoidance and fault-tolerance. Intrusion avoidance techniques are the most common in secure systems and in particular are the basis of the notion of *trust*. For instance, when the "*Orange Book*" [DoD85] states that a reference monitor must be *tamperproof*, it means that you must prevent intrusions into it. In such systems, the protection mechanisms must prevent the unauthorised actions. If an intruder succeeds in by-passing these protections, the security of the system is no longer ensured. If a security administrator decides to carry out illegal actions, there is no logical protection to prevent him from doing so. He could only be detected by using an intrusion-detection model [Denning86] which is able to detect intrusions by monitoring audit records for abnormal patterns of system usage.

The principles of the intrusion tolerance are different [FDP86],[Fraga85]. The system tolerates a bounded number of misuses. If one or more intruders by-passes the protection mechanisms and if the number of misuses they do is less than a given threshold, the security properties of the system (confidentiality, integrity and availability) are still ensured.

Three types of intrusion tolerance can be formulated:

- for *confidentiality*: read access to a subset of confidential data gives no information about the data,
- for *integrity*: the change of a subset of data does not change the data perceived by legitimate users,
- for *availability*: the change or deletion of a subset of data or of a server does not produce a denial of service to legitimate users.

For each property, a tolerance threshold is defined. If the reading, modification or destruction is done on a part D' of data/server D such that |D'| (size of D') is less than the threshold, the properties are always verified (figure 13.2).

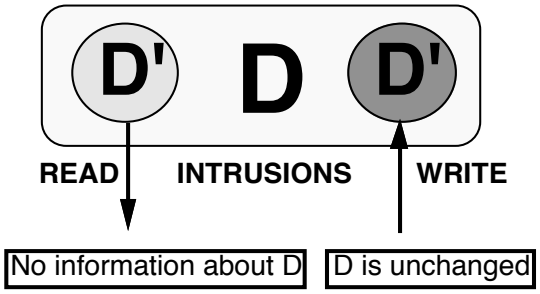

Figure 13.2: Intrusion tolerance

13.2.2 Fragmentation-and-scattering

Fragmentation-and-scattering is an intrusion-tolerance method which can be compared with redundancy in a classical fault-tolerance context; redundancy is used to ensure that the occurrence of a fault in one copy will be of no consequence because this fault will not appear in the other copies. Fragmentation consists in defining different fragments of the data so as to ensure that once isolated, every fragment is of no interest due to the lack of sufficient information. Scattering refers to the way by which each fragment is isolated from the others. Once this operation is carried out, an intrusion into a part of the system has no consequence due to the lack of significant information involved in the fragments to which the intruder has access. The number of intrusions that can be tolerated without delivery of significant information is very dependent on the way the operations are defined. This number is a parameter which can be chosen in regard of the trade-off between security and performance.

Fragmentation may be carried out in many ways, with or without a secret parameter (fragmentation key). It can be performed with help of other techniques such as cryptography to increase the level of security. It may also be performed with various granularities of the data such as bit level or byte level. The granularity of fragmentation directly influences the security and the performance of the method.

Scattering may be carried out in different ways, depending on which intrusions one wants to tolerate. *Geographical scattering* may be used for both communications and data storage. One example of using different communication channels for sending the different fragments was discussed in the previous section. As another example, Rutledge discusses the use of different telephone lines to transmit the different fragments to the receiver [Rutledge87]. Geographical scattering for data storage may take place in a network environment where several archive sites are available. This solution will be detailed later.

Another form of scattering is *temporal scattering*: fragments are sent over a communication channel at different and unpredictable moments. In this case, fragments coming from different sources must have the same length in order to hide their origin.

Frequential or spread-spectrum scattering is also interesting: it consists in the use of different frequencies to transmit the fragments in radio-communication applications or in broad-band local area networks.

Yet another scattering technique is *access right scattering*, which is a way to implement the separation of duties proposed by the Clark-Wilson policy [CW87]. With this technique, certain sensitive operations need the cooperation of several users to be performed.

The main idea that governs all forms of scattering is the difficulty that an intruder would have for retrieving all the information, because he would have to realise several intrusions in different places or at different times or frequencies in a consistent manner.

Data cannot be protected by fragmentation-and-scattering all of the time since it must exist as a whole when it has to be processed. It is possible to apply intrusion-tolerance for communication security or for data storage security but at the time of processing, data must be reassembled. Once data is reassembled, other forms of protection are needed. There is a similar problem with classical error-masking in which the final vote procedures must be trusted. Although fragmentation-scattering must

be used in conjunction with other security means, it greatly restrains the locations where data is not fragmented, thus making protection easier.

13.3 Fragmentation-scattering applied to file archiving

13.3.1 Overall framework

Our proposition in Delta-4 is to implement a *secure data archiving system* for different types of intrusions that could occur either in the storage devices (archive sites) or in the communication channel. Data security is provided by intrusion-tolerance and more precisely by geographical fragmentation-scattering (see figure 13.3).

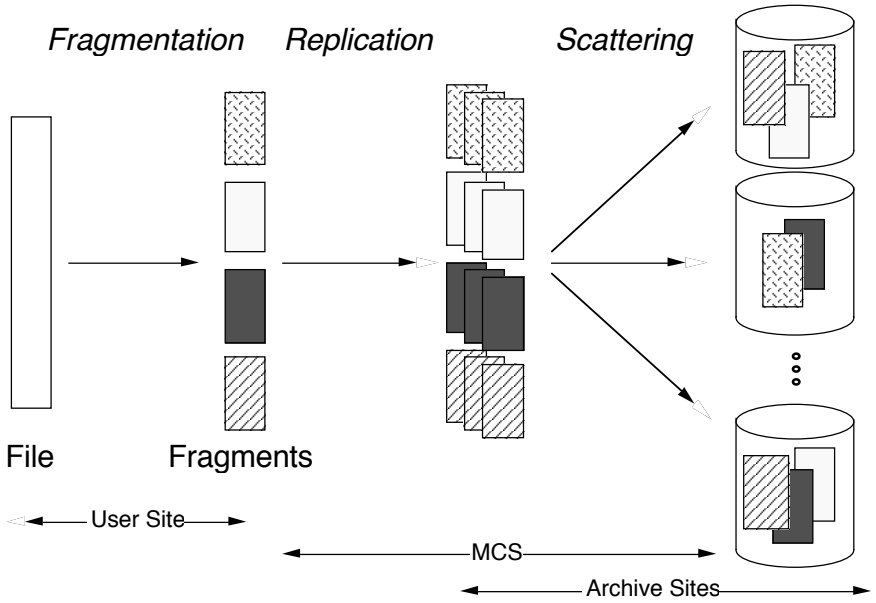

Figure 13.3: General principle of fragmentation-scattering applied to file archiving

The basis of the fragmentation and scattering technique is to cut every sensitive file into several fragments in such a way that one or more fragments (but not all) are not sufficient to reconstitute the file. These fragments are then stored in geographically distributed archive sites. An intruder accessing some sites cannot obtain all the

fragments of a given file unless he has almost overall control of the complete distributed system. On the other hand, in order to obtain availability, several copies of each fragment are stored on different archive sites. This service thus answers all three security requirements: confidentiality, integrity and availability. Confidentiality and integrity are directly provided by fragmentation-scattering while availability is obtained by replication of fragmented data. In this section, after a presentation of the environment needed for implementing fragmentation-scattering, we describe the basis of the different operations that will take place in order to secure sensitive data.

For the purpose of describing the file archiving service, we consider only two kinds of sites, interconnected by the multipoint communication system (figure 13.4):

- *user sites*, which are personal workstations that constitute the physical domains of their users,
- *archive sites*, designed for long-duration data storage, while the user of the data is not logged-in,

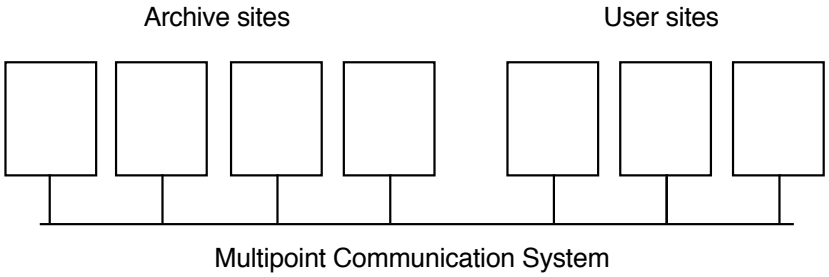

Figure 13.4: System environment

A user has full authority over the objects stored in the machine he/she is working on. During the user session (log-in time), nobody else can work on the same machine. Key management and access right management will be addressed in the next section (see paragraph 13.4: intrusion tolerant security server).

13.3.2 Archive service

An archive is processed under a set of operations as described below. Some of the archive management operations are carried out in the user site, others are

remotely executed by the archive sites. To ensure a high level of security, whole files are never available, except in the connected-user site. Thus, data sent by the user is always in a fragmented form with no possibility of recognising the different fragments derived from a given file (otherwise, eavesdropping of the communication channel could annihilate the added advantages of the technique). Consequently, fragmentation and naming of fragments is executed in the user site.

13.3.2.1 User site operations

Every user site has a well-defined archive-management service. This service is concerned with storage and retrieval of the files within the distributed file archive system. The archive management-service uses the fragment services of the different archive sites and the directory services of the security server (see paragraph 13.4) in order to build the file structure. It provides for the archives all the usual file operations such as creation, deletion, opening, closing, reading, writing...

As stated earlier, two basic operations related to file security are provided in the user site: fragmentation and naming of the fragments. The fragmentation operation uses a fragmentation key that is distributed by the security server. This operation is based on fast and simple algorithms that give flexible access to any file whilst ensuring a high level of security due to the scattering of information. The names given to the different fragments are generated by cryptographic methods using the fragmentation key, such that no information can be derived from these names. The naming is carried out in such a way that fragments have a unique identifier, derived (through a non-inversible transformation) from the fragmentation key, the name of the file and some other parameters. During the read operation, the original file is reconstituted with the same key that was used for fragmentation.

13.3.2.2 Archive site operations

A fragment service is provided by the archive sites. The operations of these sites correspond to simple space allocations on the physical storage devices and data transfers between the storage device and the network. These fragment operations are only available to the file-management service embedded in the user sites. The archive sites carry out an access control to the stored fragments. The presentation of the fragment names and of a ticket generated by the security server (see 13.4) constitutes a capability-type access control mechanism. At the end of a user session, file archiving is executed by sending the fragments over the communication channel, from the user sites to the archive sites. Every archive site decides whether or not it should be saved locally, depending on a distributed algorithm ensuring

security and availability, based on principles discussed in section 13.3.5. For the read operation, the user site broadcasts the names of the fragments, and for each fragment, every archive site which had stored a fragment copy sends it to the user site.

Each archive site acts as a file server which would only store fixed length files, with a "flat directory" structure. The operations managed by these archive sites are fragment read, fragment writing, and fragment deletion. Only fragment names are visible for any archive site; thus, it cannot determine where a fragment comes from or to which file a fragment belongs.

13.3.3 File access session

The aim of this paragraph is to clarify the different actions needed to access a file either for writing or reading from a user site, where the user is already logged-in, and thus recognised as authorised.

13.3.3.1 Write operation

The write operation is achieved with the following steps:

W1: The user request for file opening with write-access is transmitted to the security server, which checks the access right according to the archive descriptor.

W2: If this user has write-access rights on this file, the fragmentation key is transmitted by the security server to the user site.

W3: The file is fragmented and the fragments are named, using the fragmentation key in the user site.

W4: The fragments are broadcasted from the user site to the archive classes in a random order (temporal scattering). An intruder who is eavesdropping the communication channel or who controls an archive site does not know in which order the fragments have been sent, and then has to run a large number of trials to reconstitute a file page (see 13.3.4.2).

W5: Each archive site which receives a fragment decides whether or not it should be stored locally depending on a specific, pseudo-random, distributed algorithm that takes into account the relative available storage space on each archive site (the need to take into account the available space at each site is necessary to maintain a good balance among the different archive sites).

W6: The user-site copy of the file must be deleted.

13.3.3.2 Read operation

The read operation entails a similar sequence:

R1: The user request for file opening with read-access is transmitted to the security server, which checks the access right according to the archive descriptor.

R2: If this user has read-access rights on this file, the fragmentation key is transmitted by the security server to the user site thus allowing re-computation of the names of the fragments.

R3: Fragment requests (transmitted in a random order) are broadcasted to the archive sites that send the fragments back to the user site.

R4: Once all the fragments have been received, the file is reassembled in the user site.

We propose now to discuss more accurately the way in which both fragmentation and scattering have been implemented.

13.3.4 Fragmentation principles

A general approach is proposed for the fragmentation operation. This operation consists in defining all the fragments of a file. The file may be of any length and of any type. The fragmentation operation must ensure that, once the fragments are isolated, no information can be obtained from them. Moreover, it must be impossible to guess their origin, this requirement implying that all the fragments from a given file must be of fixed length and that their names do not allow any information to be deduced. Finally, an important requirement concerns data integrity: modification of a fragment must be easily detected at the time of reuse.

13.3.4.1 Partitioning

A method has to be defined that is suitable for producing identical-length fragments from files with very different lengths. The solution proposed is to first cut every file into pages of fixed size (partitioning). Every page may then be fragmented into an identical number of fragments. The files are padded out to reach a size equal to a multiple of a page size (figure 13.5). All the fragments so obtained will have the same length, which may be chosen equal to that of a packet sent on the

communication channel, or a quantum in the mass storage, for example. These choices may also improve the speed of access to information. Another advantage is that one does not need to get the whole file: pages can be retrieved independently. So, a user does not need to reassemble a whole file if he only needs a single page. A cryptographic checksum is added to each page; this checksum is checked by the read operation to verify the integrity of the page.

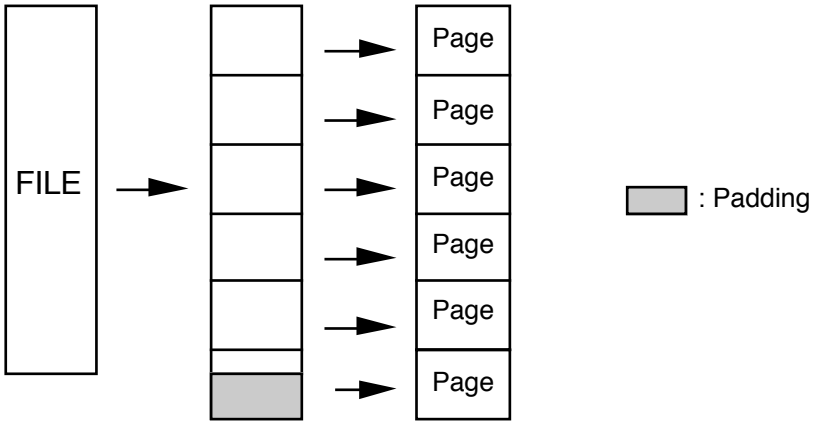

Figure 13.5: A file partitioned into pages

The mean overhead due to padding information is half a page, and is of course a large overhead for small files. The shorter a page is, the smaller the overhead is, but the longer the management time, mainly due to fragment storage time is.

13.3.4.2 Fragmentation

A very simple fragmentation scheme is to just scatter the data symbols among the fragments, without any real ciphering; however in this case, an intruder having several fragments could possibly derive the whole page by guessing the missing data. At the very least, he could obtain a good idea of the symbol vocabulary used in the unfragmented file. Cryptographic methods must thus be used in conjunction with the fragmentation. The question that one may then ask is: why use fragmentation since cryptographic techniques must still be employed? There are two good reasons:

- first, the geographical scattering of fragments makes theft of individual storage media of no avail to the intruder - even if he possesses the cypher key,
- secondly, the added security of scattering means that the ciphers employed can be much simpler and thus faster than conventional ones.

One may imagine different schemes to realise ciphering and fragmentation together. Our choice consists in the fragmentation of ciphered pages (figure 13.6). Each page is first ciphered and the fragments are obtained from this ciphered page. The fragmentation itself uses a fixed scheme wherein each successive quantum of data is put into one of the fragments according to a fixed distribution that does not depend on the key. This operation leads to a fine-grain scattering of the data among all the fragments.

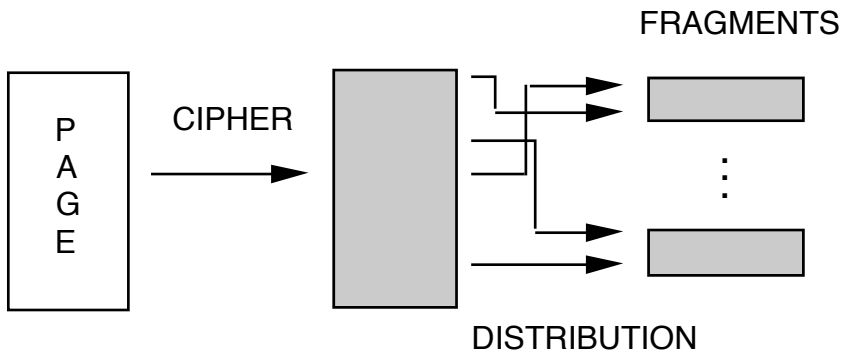

Figure 13.6: Ciphering/fragmentation

13.3.4.3 Choice of an appropriate cipher

In order to make it as difficult as possible for an intruder to decipher an individual fragment or sub-set of fragments, it is preferable to choose a cipher scheme that makes the ciphertext of each data-quantum, and thus each fragment, dependent on the others. This may be realised by using a stream cipher. With such a cipher, the key used to cipher a quantum of plaintext changes for each quantum. If used in a *cipher feedback mode* the preceding ciphered text is necessary to determine the following plaintext (figure 13.7a). This introduces secrecy if any quantum of text is missing. Another method, not very different, is *cipher block chaining* (CBC), in which the plaintext is chained with a part of the preceding cipher text (figure 13.7b).

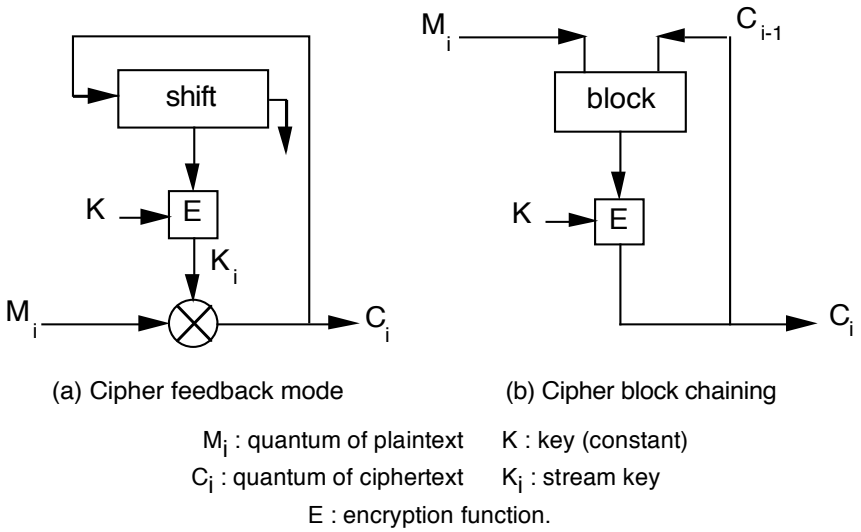

Figure 13.7: Ciphering techniques

Both methods enable to check the integrity of the fragments, because if one quantum is modified, all the following deciphered text is changed.

Once the ciphered block is obtained, fragmentation is carried out: fragment number *j* contains the fixed quanta (bits, bytes, words...) number *i* such that j=i (mod N) , *N* being the number of fragments.

13.3.4.4 Fragment naming

As stated earlier, the operation of naming the fragments is carried out in the user site. Naming consists in assigning a unique identifier to every fragment; this unique identifier is derived from the fragmentation key, the name of the file, the index of the page and the index of the fragment. The naming algorithm is based on one-way cryptographic functions such that no information concerning one fragment can be derived from its name.

13.3.5 Scattering principles

The fragment writing requests, fragment read requests or fragment deletion requests which are transmitted by the user site to the fragment server sites, are sent in a random order for each page. That means that if an intruder is eavesdropping the network or controls a fragment server site, he can receive all the fragments of a given page, but he is not able to know in which order he has to put the fragments to attempt a cryptanalysis of the page. For instance, if a page is cut in 16 fragments, an intruder should attempt $16!/2 \approx 10^{13}$ trials to find the correct arrangement. Thus, the confidentiality depends more on this random order than on the efficiency of the cipher.

Once the file is fragmented in the user site, the fragments are broadcasted to the archive sites in order to be stored, each one in a fixed number of sites to ensure availability via redundancy. Using a broadcast communication channel, each fragment is sent only once and is received by all the archive sites. In this paragraph, we discuss the way the fragments are stored.

Once a fragment has been received by all the sites, a decision has to be taken by these sites to ensure that R copies (exactly) will be stored. A distributed pseudo-random algorithm is then required that takes into account the relative available space at each site in order to decide the final localisations of the fragments.

The need to take into account the available space at each site is necessary in order to maintain a good balance among the different archive sites. On the other hand, (pseudo-)random behaviour is interesting to prevent an intruder from knowing the actual localisations of the replicates.

13.3.6 Security of the archive service

Scattering increases confidentiality because a number of concerted intrusions is necessary in order to get all the fragments coming from a single page. It also increases data availability, due to the replication of fragments: an intruder would have to destroy as many sites as the number of replicates to make a fragment unavailable. Integrity properties are provided because an intruder would have to modify all the replicates and so must realize several intrusions. Moreover, it is very unlikely that an intruder could modify even one byte of a fragment without modifying the cryptographic checksum of the page.

Other techniques could be used to implement a secure file archiving server. The first one consists in ciphering the file on the user site, and storing the ciphered file in several copies on different archive servers. In that case, confidentiality relies on the efficiency of the cipher algorithm: an intruder who is eavesdropping the network or who gets a copy of the ciphered file (e.g. a file server back-up tape) can take all the time and all the computer power he wants to cryptanalyse the file; with one intrusion, he gets all the information he needs. With our technique, the intruder who gets all the fragments of a page, would have to try 10^{13} arrangements before cryptanalysing the ciphered page. That means that our cipher can be $\approx 10^{13}$ times less strong, and can be much faster. For integrity, the two techniques are comparable. For availability, the two techniques are equivalent for accidental server failures, but the fragmentation-replication-scattering technique is less robust against simultaneous destruction of storage servers: if an intruder is able to destroy R (out of N) fragment server sites at the same time, he will make much more files unavailable than if he destroys R ciphered file servers. The overhead of the two techniques are equivalent for the communications and the storage space, but fragmentation-scattering can be made much less CPU-consuming than the ciphered file approach.

Another technique has been proposed by Rabin for fault-tolerant file servers: the "Information Dispersal" approach [Rabin89]. This technique consists in coding the file with a special error-correcting code and in splitting the coded file in n pieces, such that m out of n of these pieces are sufficient to rebuild the complete file. The n file pieces are stored by different storage servers. The coded file is longer than the original file, but the redundancy is much smaller than replication in $(n - m + 1)$ copies, for nearly the same availability and integrity. But this code is much more CPU consuming than the fragmentation and it does not ensure file confidentiality: to prevent information disclosure to eavesdroppers and storage server intruders, the file has to be ciphered before coding. To summarize, the storage and communication overhead is much less important in the Rabin's technique, but it consumes much more CPU time.

13.4 Intrusion-tolerant security server

This section describes the Delta-4 approach to the distributed system security management, by means of a distributed, intrusion-tolerant security server.

13.4.1 Basic principles

13.4.1.1 Centralised versus distributed security management

Most of the currently developed secure systems are based on paradigms such as the *access control matrix*, *reference monitor*, *security kernel* or *trusted computing base concepts*. These concepts are essentially centralised, in order to keep their implementation simple and verifiable. Such a centralised approach is inconsistent with the distribution, local autonomy and concurrency that distributed systems are supposed to provide.

Nevertheless, this approach has been maintained for the design of some distributed systems in which all the security relevant functions have been centralised in a specific site which is a mandatory mediator within any interaction. For instance, in the Secure File System proposed for the Newcastle Connection [RR83], a specialized, trusted site called "secure file manager", implements a "file access reference monitor" which is responsible for the enforcement of the mandatory access control policy. The throughput of such an approach is very low because all communications between a subject and an object must be relayed by the reference monitor site. This site is a "hard core" for both security and availability of the whole distributed system.

Another approach is proposed by the "Red Book" (Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria [NCSC87a]), in which a Network Trusted Computing Base is composed of a set of cooperating Trusted Computing Bases. Within each site of the distributed system, a local TCB is responsible of the authentication of local users, and of the access control to local objects. For the accesses from local subjects to remote objects, the local TCB must cooperate with the remote TCBs responsible for the objects. The enforcement of the authorisation policy is based on the cooperation between the TCBs, which must trust each other. This approach is unsuitable for heterogeneous, open distributed systems. Moreover, a successful intrusion of a local TCB can endanger the security of the whole distributed system. Such a case has to be considered seriously since, with current workstations, it is easy for a local user to obtain complete local control (e.g. as superuser).

For open distributed systems, yet another approach is to give the responsibility of user authentication to a trusted site, the "*authentication server*", and to give the responsibility of authorisation to the servers responsible for object management. An example of this approach is the Kerberos system [SNS88]. In such systems, security

depends on the trusted servers and on the trustworthiness of the people in charge of these servers. For instance, in the Kerberos authentication servers, all the user passwords are recorded in a plain text file, which makes this file and this site good targets for intrusions and the administrators of this site good targets for bribery.

The approach described here consists in giving the responsibility of nearly all the security functions to a set of specialized sites so as to realize a *distributed security server*. While this global distributed security server can be *trusted*, it is not necessary to trust any of its components or any individual site administrator: such a security server is *intrusion tolerant*. Since no individual site has to be trusted, such an approach is compatible with the heterogeneity of open distributed systems, such as Delta-4.

13.4.1.2 Security server requirements

The objectives of the intrusion tolerant approach are:

- openness, no specific hardware.
- reduction of TCB (by intrusion tolerance).
- modular security requirements.

In Delta-4, the security view of the network defines three kinds of sites (figure 13.8):

- *User sites* are untrusted computers where users can log in. The local security is ensured by users.
- *Security sites* are computers providing security services: registration, identification-authentication, authorisation, sensitive information management, audit and recovery service.
- *Particular servers* whose access needs to be secured. In the current system, this is the case of the archive service located on archive sites.

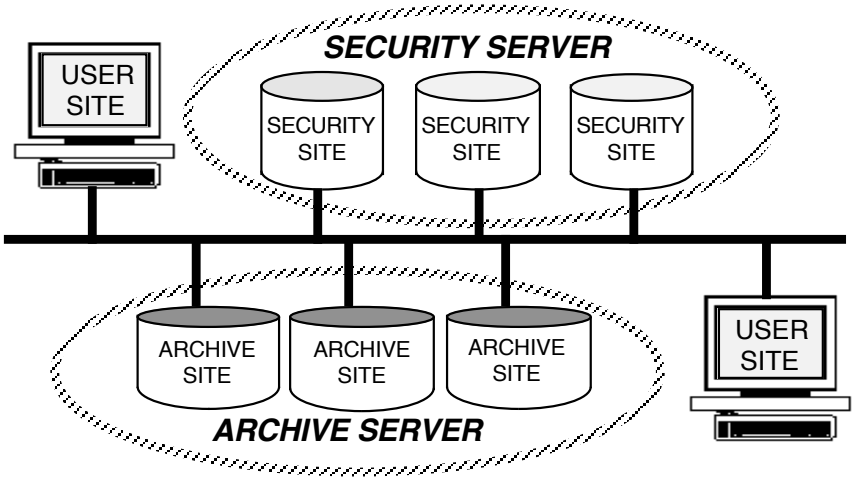

Figure 13.8: The different types of sites of the network

The fault assumptions for security sites are:

- The probability of more than one intrusion before detection and recovery is small.
- The probability of an intrusion into a site is independent of the previous intrusion(s) in other site(s).

The characteristics of the intrusion-tolerant security approach are:

- An intrusion or a misuse on one security site is immediately masked and has no consequence on the service and on its properties.
- If errors occur on some security sites before recovery, the number which will be masked depends on the services and their properties (confidentiality, integrity, availability). Service performance can be degraded.

13.4.1.3 Intrusion tolerance by distribution

The fragmentation-and-scattering technique has shown that distribution can be exploited for intrusion-tolerance. Other intrusion-tolerance mechanisms have been proposed such as some cryptographic tools like the Shamir's threshold scheme [Shamir79]. The data is split in "shadows", each shadow being stored on one security site. To build the data you only need a sufficient number of shadows called

the threshold. If you do not have enough shadows, you cannot build the initial data. The same scheme can ensure availability and integrity.

Among all the information managed by the security server, some data is not confidential; so it is possible to replicate such data on each security site. Data is *shared* or *replicated* according to its confidentiality.

Another important point is the prevention of denial of service. In this case, it is not just data but a service that has to be protected. The server thus has to be replicated on each security site in order to prevent denial of service in case of a security site unavailability. However the different sites cannot take certain decisions independently. They must agree by communicating data and local decisions. To ensure the last property, the servers must obey to two implementation principles: *replication* and *agreement* (figure 13.9).

The distribution of the security service permits a geographic distribution of the security sites. This makes the intruder's task more difficult since even if he succeeds in accessing one site, it will be more difficult for him to access other sites if they are not in the same place. It would indeed be a pity to distribute programs and data to perform logical intrusion tolerance and to not provide geographic distribution to assist physical intrusion tolerance.

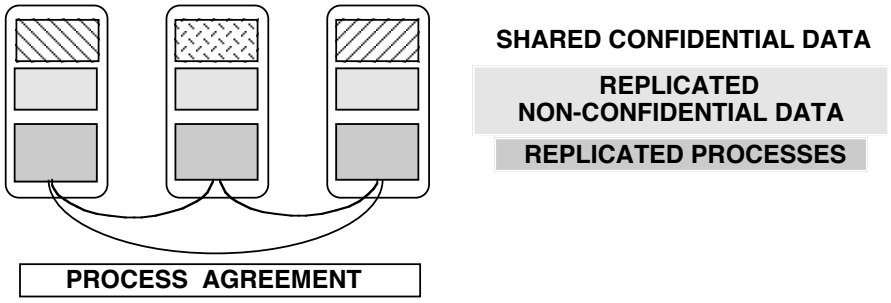

Figure 13.9: Distributed Security Services on three security sites

13.4.1.4 Distributed TCB

In the Red Book architecture, there exists on each computer a part called the Trusted Computing Base (TCB) including the Trusted Interface Unit (TIU) and the Network Trusted Computing Base (NTCB) partition. The TIU assumes only com-

munications security whereas the NTCB partition implements the local part of the network authorisation policy. This NTCB partition must have a very high physical and logical protection. Moreover all sites have to trust one another. On each computer, accesses are mediated by the local TCB which is firstly the implementation of the *local* authorisation policy. The *global* authorisation policy is implemented in the set of NTCB partitions. When all computers are connected by a network and a subject wants to access a remote object, the NTCB partitions communicate with each other. In this case, a subject on a given site must trust all sites he wants to access. If one site has been penetrated by an intruder, the security of the network cannot be ensured.

In the Kerberos architecture, the most important part of the TCB is within the security server. There also exists on each server an important TCB which has to carry out authorisation operations. However if an intruder succeeds in penetrating a server, he cannot access the other ones. The consequences of an intrusion are then limited. The only site you really have to trust to ensure global security is the security server site. However, in this case there exists a single point of failure.

In the intrusion tolerant approach, there is no local "Trusted" Computing Base on the security sites. Only the set of security sites is globally trusted (figure 13.10). There is also a small local TCB on the user sites and the secured servers. The servers themselves, for instance the Archive Service, can have a distributed TCB. In this case, the only single point of failure is on the user site. This can be minimized if the user site is considered as a one-user computer when a user accesses a secured service. If this were not so, there would always exist trapdoors on the local protections between users.

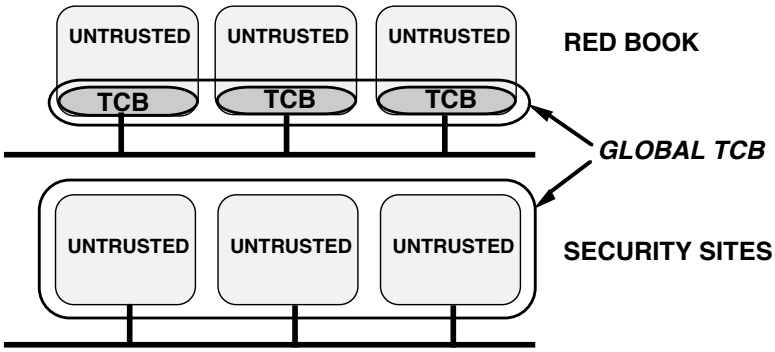

Figure 13.10: Red Book versus Security Sites TCB

The security server is trusted, but not the different computers. The security service is considered as one global server. If an intrusion in one computer is successful in a classical architecture with a local TCB, the security of the full system is no longer verified. On the contrary, if protections of one security site in the intrusion tolerant approach are by-passed, the security of the global system is maintained. These differences come from three different points of view about security and networks:

- In the Red Book, a network is seen as only a communication channel (low layers) between centralized systems. A subject, or an object, is located on one site and cannot be shared between several sites. In our architecture, the network is a support for distributed applications. A subject or an object can be shared on several sites.
- In the Red Book, sites are time-sharing systems with several users working on the same host. These computers are controlled by a system administrator. In Kerberos, sites are workstations under the control of one local user. In our system, whenever possible, the use of one host by several users during access to a secured service will be prohibited.
- The Red Book architecture is a support for DoD policy, a multilevel mandatory policy where confidentiality is the most important property to be ensured. All sites thus need to have an important trusted part. It is not possible to implement this policy in our system (see paragraph 13.5). Trusted paths between all user sites would be needed.

13.4.2 The security services

The different services which must be provided by the security sites are registration, authentication, authorisation, sensitive data management, audit and recovery service.

13.4.2.1 User registration service

Registration of a new user in the system must be done under the control of the security administrator on each authentication site. The user is registered on the first site by the first security administrator, on the second site by the second administrator and so on. On each site, the identity of the user is stored with some authenticator which will be used by the authentication process to verify the claimed identity. The authenticator can be a password, a secret permanent key or a public key. Except if public keys are used, different authenticators will be stored on the different authentication sites for the same user. This can be considered as an implementation

of some *separation of duty*: one or a minority of security administrators cannot register an illegitimate user and cannot prevent the registration of a legitimate user on a majority of authentication sites. Moreover, if such malicious administrators try to use local information stored on their sites in order to impersonate a registered user, they will fail because they cannot be authenticated by the other sites.

13.4.2.2 User authentication service

When a registered user wants to access remote servers from a workstation, he has to be authenticated by the authentication server: an authentication protocol has to be run between the user site and the security sites. This protocol is composed of three phases (see figure 13.11). In the first phase, the user site attempts an independent authentication with each of the security sites. This authentication can be based on classical authentication algorithms [NS78, Lamport81, ...], but with different authenticators (or different challenges with public key systems) for the different security sites. In this first phase, each security site independently decides if, for itself, the authentication attempt succeeds or not. During the second phase, each security site broadcasts its own decision to all the other security sites, and receives the decisions of these sites. In the third phase, according to the majority of all the received decisions and its own decision, the security site authorises (or not) the session for the user, and confirms this session authorisation by sending a session key to the user site (or a ticket containing the session key, depending on the authorisation algorithm which has been selected). A different session key is randomly generated by each security site. This session key or ticket will be used by the user site to authenticate its requests in the authorisation process (see section 13.4.2.3).

In this protocol, the majority voting on the different authentication decisions enables tolerance of accidental faults affecting the registration data stored on the different security sites or communication faults during the protocol>. It also enables tolerance of intrusions into a minority of security sites which could lead to false local decisions. Moreover, different session keys are generated independently by each security site, and sent only to the user site; thus, no intruder, even with the complicity of an administrator, can impersonate the user site (except if he cracks the authentication algorithms).

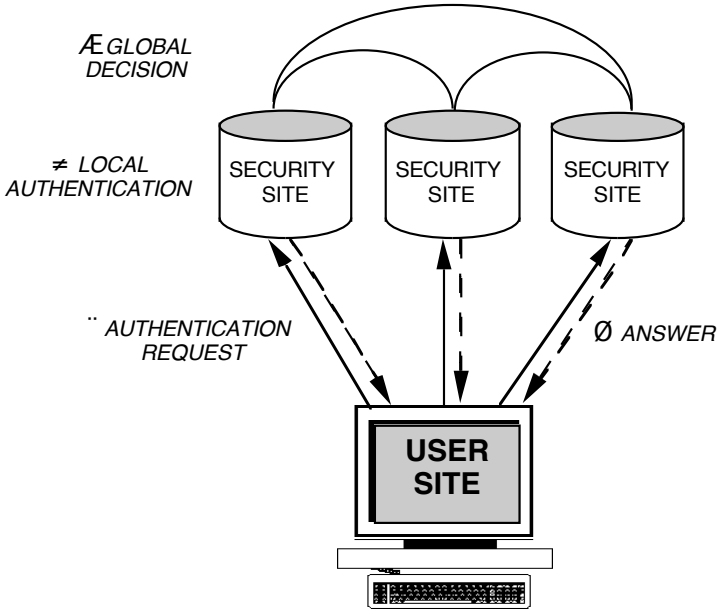

Figure 13.11: The Authentication protocol

It would be difficult for a user to memorize several strong independent passwords. A better solution is to store secret keys on a personal smartcard, the owner of which has only to memorize his PIN. The current Delta-4 implementation uses Bull CP8 smartcards with shared keys, one smartcard for each user and one for each security site administrator; all the administrators of a given security site have identical master smartcards, except for the identification and the PIN of the administrator. On the user smartcard, there is a set of areas, one area for each security site, i.e. for each master smartcard. That means that when a user is registered by a security site, the local master smartcard generates a secret key that it writes within its own reserved area on the user smartcard. When the user has been registered by the N sites, his smartcard possesses N secret keys within N areas. In the authentication phase, each security site sends a challenge to the user. The user smartcard applies a one-way function to this challenge and the shared secret key and it sends the result to the security site. The master smartcard makes the same operations to the same data and it compares the results. This protocol is performed by every security site.

13.4.2.3 Authorisation service

The role of this service is to check that the access to a secured service by a subject is authorised according to its access-rights. This service is made intrusion-tolerant by its implementation on the security sites, and by means of an authorisation protocol between the user site and the security sites.

As a matter of fact, the authorisation protocol is quite similar to the authentication protocol: whatever the user request, a local decision is first taken by each security site according to the user access rights which are locally stored; this local decision is then broadcast to the other security sites; the authorisation decisions received from the other sites are voted locally (with the local decision), and, according to the vote result, the user request is locally executed or not. This majority voting technique ensures that a legitimate request cannot be denied or an illegitimate request cannot be granted, except if a majority of authorisation data copies have been destroyed or altered.

The different phases of authorisation are (figure 13.12):

- The subject asks the security servers for permission to access a secured service by sending its identifier (received in the authentication phase) (1).
- The access-rights stored on the security sites enable the latter to verify that the subject is authorised to access the requested service.
- The security sites vote to decide if the access is authorised using the same protocol as that defined for authentication (2).
- If the sites agree to permit access, they send a ticket to the subject and another ticket to the secured server (3).
- With the ticket, the subject can open a session with the server.

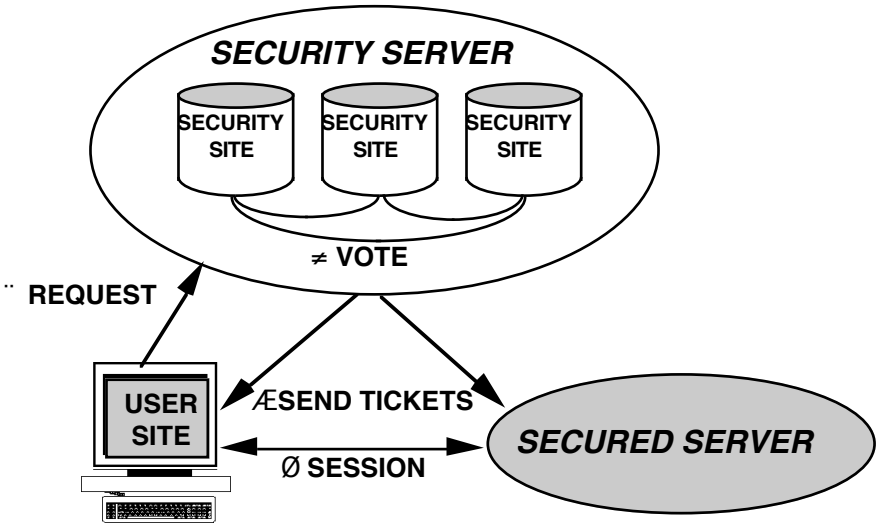

Figure 13.12: The Authorisation protocol

A subject may also want to access secured data stored on security sites such as access-rights or sensitive object descriptors. In this case, the access control protocol is the same as the one described above for the phases 1 and 2 and then, if and, when the security sites accept the access, they perform it on each site and send the information or the affirmative response to the user site.

13.4.2.4 Sensitive data management service

The role of this service is to store, manage and retrieve the sensitive information on the security servers so that their protection verifies the hypothesis made in paragraph 13.4.1.2. This information consists of short data items needed to achieve security services.

The data management service must enforce the three main security properties (confidentiality, integrity and availability). The integrity property is provided by modification detection mechanism such as cryptographic signatures. According to the sensitivity of the security data, it can be important to preserve both the confidentiality and availability of this data, or only the availability. For this, two storage techniques can be applied: replication (for availability) or threshold schemes (for

confidentiality and availability). In function of this, the security administrators (data for a service access) or a subject (data for authentication) will store data with one of the two algorithms (figure 13.13).

If a data item is replicated on N security sites, it is assumed:

- with respect to availability, that N-1 replicates can be lost (modified or destroyed),
- with respect to confidentiality, that one replicate is sufficient to observe data.

If one data is shared on N security sites (in this case we speak of shadows) using a threshold T, it is assumed:

- with respect to availability, that N-T shadows can be lost,
- with respect to confidentiality, that T shadows are necessary and sufficient to observe data (less than T shadows gives *no information* about the data).

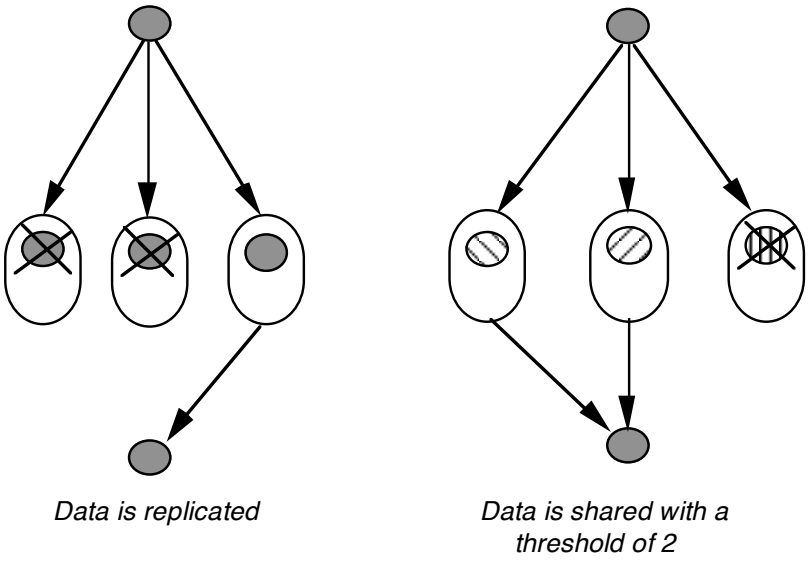

Figure 13.13: Replication and Threshold Scheme

13.4.2.5 Audit service

The role of this service is to record all information related to security. Such information is sent by the services defined above. There exists two kinds of information, authorised operations performed by authorised users (registration, access, rights change, ...) and attempted or successful intrusions or misuses. It is not the role of the services to determine what is an intrusion or misuse by an authorised user. This is the function of an audit trail analysis.

The audit information is sent not only by security sites but also by secured servers and user sites. For the former, it will be access-requests, and for the latter it will be, for instance, information about correct or incorrect shared data sent by security sites (bad shadows received from certain security sites).

The audit trails are stored on each security site. The information received on one site is not sufficient to compromise the security of the system.

The analysis of this audit information will be done off-line by security administrators. As one intrusion is masked, it is not necessary to detect intrusion on-line.

13.4.2.6 Recovery service

It acts as an error recovery mechanism to correct certain modified data (e.g. shadows of the threshold scheme). Other recovery functions can be performed manually by security administrators using audit trails.

13.5 Selection and implementation of an authorisation policy

The security policy of an enterprise strongly influences the design of the system intended to support applications for this enterprise. In our case, the architecture was defined without a particular security policy in mind. All the policies are therefore not necessarily compatible with the security architecture and the hypotheses. In our case, mandatory policies cannot be implemented for the same reasons as Kerberos: security is not ensured within a user site and between user sites. We have thus chosen a discretionary authorisation policy.

13.5.1 Implementation of discretionary policy

Discretionary policy is an authorisation policy where some authorised subjects have right-modify right on an object. *Right-modify right* on an object is the right to modify the access-rights on the object. Discretionary policy means that subjects which have right-modify rights must evaluate by themselves the trustworthiness of other subjects. Discretionary policy is very close to our distributed security ideas. In our system, if a subject can read information he can disclose this information to whoever he wants. Only some paths are trusted. Subjects can be identified as all processes acting on user sites. If this site can reassemble an archive, it can forward it to other user sites without any control. The unauthorised disclosure of information is only limited by the trust a subject puts in another subject. If integrity is ensured by security sites, confidentiality is in part under the responsibility of users.

The propagation of other rights (execute, write, ...) can be limited by using hierarchical authorisation (right-modify right is given to the subjects at the top of hierarchy) or centralized authorisation (right-modify right is given to one security administrator) [NCSC87b]. The centralized scheme can only be used in a rigid system where most subjects or objects are created during an initialisation phase. Changes have to be infrequent. It will certainly not be the case in DELTA-4 applications. The hierarchical scheme is more flexible and is simple to implement.

13.5.1.1 Access lists and access tokens

The implementation of access-rights can be made in two ways, either using *access-control lists* or *capabilities* [Fabry74].

In our system, access control is done by security sites and then by archive sites. Security sites must authenticate subjects and authorise them to access archives. When they grant access, two solutions are possible. Firstly, security sites could be the mandatory gateway between a user site and the archive sites during a session (figure 13.14). In this case performance and security decrease since:

- security sites become a fragment-transfer bottle-neck,
- security sites know information that they should not have to know (fragment, fragment name).

The second solution is that security sites supply user sites and archive sites with elements that enable the creation of a direct session between them (figure 13.15). These elements form a ticket with cryptographic keys and access tokens (enciphered

fragment name and access-right). The access tokens will be used as capabilities and will be sent to archive sites so as to authorise users to access fragments.

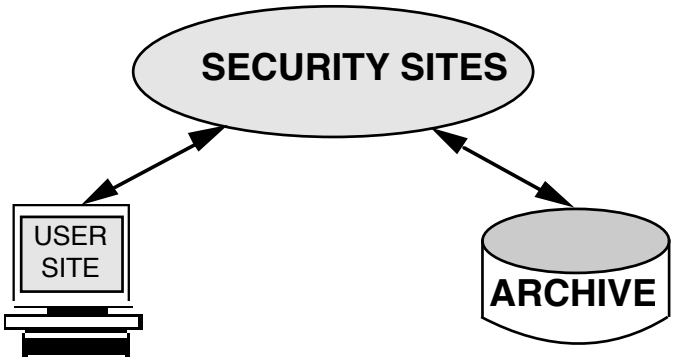

Figure 13.14: Security sites are gateway

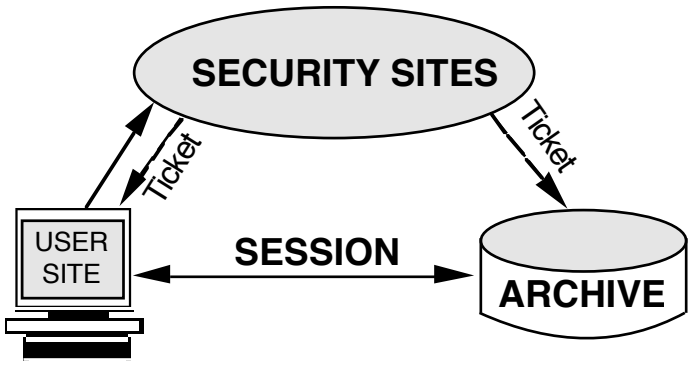

Figure 13.15: Opening session with tickets

The security sites have to verify if a subject is authorised to access an archive. The fastest way to check authorisation is the use of an access-control list which can be replicated on all sites.

13.5.1.2 Access rights

Since access-control lists are to be used, it appears interesting to use the existing UNIX file protection scheme. Under this scheme, the file authorisation list is made up as follows:

- Access-right types are read, write and execute.
- The rights are defined for the owner, the members of the file group and the other users.

A fourth field, called dynamic access-control-list (ACL), is added. This access-control-list contains a set of groups or users with added or restricted rights (figure 13.16). These groups or users have more or less rights than ones defined by precedent fields. For instance, if the object group right is execute only; in order to give write right to a particular member of the group, an entry for this subject in the access-control-list with added write rights is created. This is the same thing for groups. Rights for some subjects can also be reduced. The advantages of this dynamic implementation is to be close to complete access-control lists granularity without excessive data overhead.

Fixed List	
Owner	rights
Group	rights
Others	rights
Dynamic ACL	
User or Group	Extended Rights
User or Group	Restricted Rights

Figure 13.16: Archive access-control-list

13.5.2 Discretionary multi-categories policy

Groups can be defined in a such a way that security categories can be built. Only the security administrators will be authorised to create and modify groups.

We have seen that it is impossible to implement a mandatory policy, but it is possible to make categories such as Bell-LaPadula categories [BL74]. These categories represent interest classes. In military or commercial domains, it is easy to partition objects into categories. A subject can access to more than one category whereas an object can only belong to one category (this is different from the Bell-LaPadula model). In our system, categories are built with groups. These groups should be defined by security administrators when they implement procedural part of the security policy. A group can also be added or subtracted when the system runs.

To implement the categories, UNIX group concepts are used with some differences. Firstly, a subject can belong to several groups. This is already implemented in UNIX System V, however its use is not really easy. The file access-control does not use the groups the user belongs to, it uses the "current shell group" only. It is first defined in a login configuration (passwords file) and when a user wants to have other group permissions, he has to change his "current shell group" by a command (*newgrp*).

Our idea is to verify group members during each access try. If a user belongs to several groups, he must have access-rights of these groups all the time. These rights have to be defined. It is the object and subject creation rules which do this.

13.5.2.1 Groups

Groups are subject sets and each group specifies one category. To partition categories, one category cannot include an object which belongs to another group, but a subject can belong to several groups (figure 13.17). A subject has all the rights of all the groups to which it belongs.

When the groups are specified by security administrators, and subjects are distributed over them, it is necessary to write the object creation rules which give rights to subjects.

The rules are:

- when a new object is created, the object owner is the object creator and has U-rights, the object group is the group of the owner and group members have G-rights, the other subjects have O-rights (often no rights).
- if a subject belongs to several groups, it may decide to which group the object is to belong.

These rules are modular enough to build different policies with different constraints. Security administrators have to choose the various U, G and O rights.

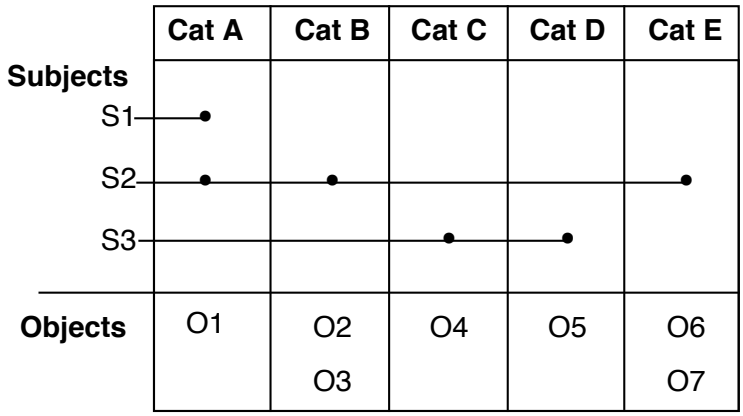

Figure 13.17: Example of groups/categories

In this example:

- S1 belongs to the category A and has G(A) rights on A-objects and O-rights on B, C, D and E-objects.
- S2 belongs to the categories A, B and E, and has G(A), G(B) and G(E) rights on the A, B and E-objects and O-rights on C and D-objects.
- S3 belongs to the categories C and D and has G(C) and G(D) rights on the C and D-objects and O-rights on the A,B and E-objects.

13.5.2.2 No discretionary control by owner

The discretionary control on the objects is not carried out by the owner. The owner has particular rights but does not have the right-modify right on the objects. A user must not be able to give rights on an object in a given category to users in other categories.

The owner of the object will be the creator of the object, but the name of the owner cannot be changed by any user.

13.5.2.3 Category managers

There exists a super user, called category manager, for each category. He is a member of the corresponding group. This user has right-modify right on all objects which belong to the category. He can change access-rights of all these objects. He does not have right-modify right on objects of other categories. A category manager cannot change the owner or the group of an object. This can only be done by security administrators. This allows separation of duties. Users, category managers and security administrators all have different roles.

13.6 Future extensions

In a large distributed system, a distributed security server would be responsible of the security management of only a subset of the whole system. That means that several security servers would have to cooperate for the global management of the distributed system. Our intrusion tolerance approach is compatible with this requirement: for instance, a user can be authenticated by a nearby distributed security server, and this user may ask for an access to an object managed by another security server; in such a case, the "nearby" security server can relay the request to the "more-remote" security server in the same way as in the "Red book" approach.

13.7 References

[BL74] D.E. Bell and L.J. LaPadula, *Secure computer systems: unified exposition and Multics interpretation*, Tech. Rept. MTR-2997 (AD/A-020-445), The MITRE Corporation, April 1974.

[CW87] D.D. Clark and D.R. Wilson, "A comparison of commercial and military computer security policies", *Proceedings of the 1987 IEEE Computer Society Symposium on Security and Privacy,* IEEE, Oakland (Ca.), USA, May 1987, pp. 184-194.

[Denning86] D.E. Denning, "An intrusion-detection model", *Proceedings of the 1986 IEEE Computer Society Symposium on Security and Privacy,* IEEE, Oakland (Ca.), USA, May 1986, pp. 118-132.

[DoD85] DoD TCSEC, *Trusted Computer System Evaluation Criteria*, Tech. Rept. DoD 5200.28-STD, Department of Defense, USA, December 1985.

[Fabry74] R.S. Fabry, "Capability-based addressing", *Communications of the ACM*, vol.17, n°7 (July 1974), pp. 40-412.

[FDP86] J.M. Fray, Y. Deswarte and D. Powell, "Intrusion-Tolerance using Fine-Grain Fragmentation-Scattering", *Proceedings of the 1986 IEEE Symposium on Security and Privacy,* IEEE, Oakland (Ca.), USA, April 1986, pp. 194-201.

[Fraga85] J. Fraga, *La sécurité des données par la tolérance aux intrusions*, Ph.D. dissertation, Institut National Polytechnique de Toulouse, Rapport LAAS-CNRS n°85.133, 157 pages, juillet 1985.

[Lamport81] L. Lamport, "Password authentication with insecure communications", *Communications of the ACM*, Vol.24, n°11 (November 1981), pp. 770-772.

[NCSC87a] NCSC TNI, *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, Tech. Rept. NCSC-TG-005, National Computer Security Center, 31 July 1987.

[NCSC87b] NCSC Guide, *A Guide to Understanding Discretionary Access Control in Trusted Systems*, Tech. Rept. National Computer Security Center, September 1987.

[NS78] N.S. Needham and M.D. Schroeder, "Using encryption for authentication in large networks of computers", *Communications of the ACM*, Vol.21, n°12 (December 1978), pp. 993-999.

[Rabin89] M.O. Rabin, "Efficient dispersion of information for security, load balancing and fault-tolerance", *Journal of the ACM*, Vol.36, n°2 (April 1989), pp. 335-348.

[RR83] J.M. Rushby and B. Randell, "A distributed secure system", *IEEE Computer*, vol.16, n°7 (July 1983), pp. 55-67.

[Rutledge87] L.S. Rutledge, *A spatial encoding mechanism for network security*, Ph.D. dissertation, Institute for Information Science and Technology, Washington, March 1987.

[Shamir79] A. Shamir, "How to Share a Secret", *Communications of the ACM*, vol.22, n°11 (November 1979), pp. 612-613.

[SNS88] J.G. Steiner, C. Neumann and J.I. Schiller, "Kerberos: an authentication service for open network systems", *Proceedings of the USENIX Winter Conference,* Dallas (Tx.), USA, February 9-12 1988, pp. 191-202.