SECURITY MODULE COMPRISING MEANS GENERATING LINKS BETWEEN MAIN FILES AND AUXILIARY FILES

The invention relates to a security module for cooperating with a data processing device and comprising data processing means and data storage means, which storage means store several files.

The term "security module" should be taken either in its conventional sense, in which it designates a device whose purpose in a communication or information network is to be held by an authority supervising the network and to store, in a protected fashion, secret fundamental parameters of the network such as cryptographic keys, or more simply, in the sense in which it designates a device allocated to various users of the network, enabling each of them to have access to the latter, which device is also capable of holding secret parameters. The security module could take the form of a portable object of the chip card type.

The present invention relates particularly to microcircuit cards, and more generally to portable objects equipped with integrated circuits comprising at least one microprocessor, a read-only memory (ROM) containing an operating system for the card, and one or more nonvolatile memories that are programmable by the microprocessor. These nonvolatile memories make it possible to store data and code. The microprocessor controls the transfer of information and, if necessary, stores the data received from the outside or reads it in order to transmit it to the outside. These objects have one or more means of communication. The memories can use EPROM, EEPROM FeRAM, SRAM or FLASH technology.

Due to the evolution of the technology, the size of the ROM memory containing the program is getting larger and larger; thus, the designers of this program can add more and more functions. Some of these functions relate directly to the organization of the programmable memory. This memory, which has also increased in size, is organized hierarchically into independent files; this organization is specifically described in ISO standard 7816-4.

At present, the cards can be used for multiple applications and for this reason, they often have two or three hierarchical levels called, for example: CARD, APPLICATION and SERVICE. At each level, the information related to the same allocation are grouped into files, these files comprising two levels of storage, namely "DIRECTORIES" and in each directory, information of the same nature stored in "DATA FILES."

This architecture defined in several levels is generally developed during the customization of the card, i.e., prior to its utilization. It is possible, however, to add other lists or other data files during

utilization, but that depends on the remaining space available in the programmable nonvolatile memory. This memory being limited in size, it is important not to waste space and to define during customization only the necessary space required for the proper functioning of the directories and the data files.

One excellent way not to lose space is not to duplicate the information. Thus, it is necessary to avoid having the same information usable by several directories be written in identical fashion in several places in the memory. Unfortunately, the hierarchical architecture of the directories prevents the sharing of the same data file among several directories. If two directories must have the same information, there is currently no solution for creating two data files comprising this same information inside. The present invention solves this problem by avoiding the duplication of common information, while retaining the hierarchical links between data files and directories.

Another problem is that of updating files present in several places in the memory; in effect, the necessary modifications must be made in each place. In addition to the fact that this is a long operation, it can be disturbed by an accidental interruption in the operation of the card, with the probable consequence of an incomplete update of this set of files, i.e., an update of some files and not others. Consistency among all of these files could no longer be ensured.

Moreover, the multilevel structure can be detrimental to the access times to lower-level data files or directories. In fact, in order to retrieve data from a lower-level directory, in many cases it is necessary to select all of the main directories on higher levels. For example, in order to go from one directory to another on the same level, it is necessary to ascend a tree structure to a first common directory, then re-descend by selecting intermediate directories. This successive selection mechanism is cumbersome and time-consuming.

The object of the present invention is to solve these various problems; it provides a means for avoiding the duplication of identical data in memory; it ensures the consistency of information shared among several files; lastly, its optimizes the search for information in remote directories, in the tree structure of the files in the memory.

To this end, it relates to a security module of the type mentioned at the beginning of the disclosure, which comprises:

- link generating means for generating a link between at least one main file and one auxiliary file, the main file having a given content and being made accessible to the processing means in the storage

means by means of location data, which link generating means associate said location data with the auxiliary file;

- branching means for making available to the processing means, when the latter execute an access request for accessing the auxiliary file, said content of the main file, using said location data.

Other details and advantages of the present invention will emerge from the following description of a preferred but non-limiting embodiment, in reference to the appended drawings, in which:

Fig. 1 shows a tree structure with several hierarchical levels in a card;

Fig. 2 shows a typical organization of directories and data files in a card;

Fig. 3 shows the detailed structure of two fundamental categories of files used in the invention;

Fig. 4 is a flow chart detailing the steps of a file creating procedure according to the invention; and

Fig. 5 is a diagram of a security module for which the invention is intended, cooperating with a data processing device.

The data processing device 51 represented in Fig. 5 comprises, in a way that is essentially known, a microprocessor 52 to which are connected a ROM 53, a RAM 54, a means 55 for cooperating, with or without physical contact, with a security module 58, and a transmission interface 57 enabling the data processing device to communicate with another similar device, either directly or through a communication network.

The device 51 can also be equipped with storage means such as diskettes or disks, which may or may not be removable, entry means (such as a keyboard and/or a pointing device of the mouse type) and display means, which various means are not represented in Fig. 5.

The data processing device can be constituted by any computing equipment installed in a private or public site, capable of providing means for managing information or providing various goods and services, which device may be permanently installed or portable. Among other things, it could be a telecommunications device.

Furthermore, the security module 58 includes data processing means 59, a nonvolatile memory 60, a volatile working memory RAM 64, and means 63 for cooperating with the data processing device. This module is defines, in the memory 60 a secret area 61 in which information, once recorded, is inaccessible from outside the module and accessible only to the processing means 59, and an open area

62 that is accessible from outside the module for reading and/or writing data. This area of the nonvolatile memory 60 can comprise a non-modifiable ROM part and a modifiable EPROM or EEPROM part, or can be constituted by a RAM of the "flash" type, i.e., having the characteristics of an EEPROM with access times identical to that of a conventional RAM.

As the security module 58, it is possible to use a microprocessor with a self-programmable nonvolatile memory, as described in US patent No. 4,382,279 in the name of the Applicant. As indicated in column 1, lines 13-25 of this patent, the self-programmable feature of the memory corresponds to the capability for a program fi located in this memory to change another program fj, also located in this memory, into a program gj. Although the means used to carry out this self-programming can vary depending on the technology used to design the data processing means 59, it is noted that in the case where these processing means are constituted by a microprocessor associated with a nonvolatile memory according to the above-mentioned patent, these means can include:

- data and address buffers, associated with the memory;
- a program for writing into the memory, loaded into the latter and specifically containing the instructions that make it possible to maintain the programming voltage of the memory as well as the data to be written and their addresses, for a sufficient length of time, which write program could, however, be replaced by a logic circuit write controller.

In a variant, the microprocessor of the security module 58 is replaced, or at least supplemented, by logic circuits implanted on a semiconductor chip. Essentially, circuits of this type are capable of performing calculations, particularly authentication and signature calculations, using hardwired, rather than microprogrammed, electronics. In particular, they can be circuits of the ASIC (Application Specific Integrated Circuit) type. The SIEMENS component marketed under the reference number SLE 4436 and the SGS-THOMSON component marketed under the reference number ST 1335 may be cited as examples.

Advantageously, the security module 58 is designed in monolithic form on a single ship.

In a variant of the microprocessor with a self-programmable nonvolatile memory described above, the protected nature of the security module could result from its being located in an inviolable enclosure.

The nonvolatile memory of the cards is organized into files which, as indicated above, can be of two types: directories or elementary data files. Each elementary file comprises a header and a body

containing information. The hierarchical level is indicated in the header, which also contains the references of the file, the status or life expectancy of the card, the access conditions and the size. As a general rule, the header contains all of the information that makes it possible to manage the information stored in the body. Two or three levels are normally used. In reference to Fig. 1, and in general, the upper level is called "CARD," and the lower levels are called "APPLICATION" or "SERVICE." It is entirely possible to design cards with more than three levels; in the example given, three levels are described.

By way of example, the same card can be used for various applications such as banking, municipal applications, medical files, or cellular telephones, which are represented by APPLICATION-level directories. In the municipal application, there are parts such as public transportation, pool and library access, and parking payments, which are represented by SERVICE-level directories.

Fig. 2 illustrates an example of the hierarchical links between files in the programmable memory of a card. The CARD directory contains two APPLICATION directories 1 and 2 and the elementary file C1. The APPLICATION directory 1 contains two SERVICE directories A1-S1 and A1-S2, and the elementary file A1-1. The SERVICE directory A1-S2 has only one elementary data file: A1S1-1. The APPLICATION directory 2 has two SERVICE directories A2-S1 and A2-S2. The SERVICE directory A2-S1 has two elementary data files: A2S1-1 and A2S1-2. The SERVICE directory A2-S2 has one elementary data file: A2S2-1. All these files take up a non-negligible amount or space in the limited memory of the card, and it is important to optimize the use of memory space and to avoid duplicating the same information in several different places.

In some cases, the same information is used by two different directories. For example, an individual card holder's banking information: the holder's name and address, the bank name and location, account number, credit information, etc., can be stored in an elementary file included in the directory corresponding to the banking application, for example: the elementary file A1-1 in the APPLICATION directory 1, described in Fig. 2.

The card can also serve as a "city card"; this application is managed by the APPLICATION directory 2. Among other things, it makes it possible to pay for public transportation, to gain access to the municipal library and certain admission-charging cultural activities (theater, films, etc.). These services are managed by the two SERVICE directories A2-S1 and A2-S2, hierarchically dependent on the APPLICATION directory 2. When the card is used as a means of payment, for example to pay for trips

made via public transportation, the money is debited directly from the bank account whose the information is indicated in the elementary file A1-1. The information in the elementary file A1-1 of the APPLICATION directory 1 must therefore be made accessible from the SERVICE directory A2-S1. This access is symbolized by the arrow in Fig. 2. The solution comprised of reproducing the data is unsatisfactory.

Another example relates to secret keys and confidential codes: their values can be identical when accessing various directories that are not hierarchically dependent. This is a substantial problem when using keys of the RSA type (from the inventors Rivest, Shamir, and Adleman) stored in more than 1024 bits. A final example relates to the elementary ratification files: these elementary files are used to store correct or incorrect presentations of keys or codes. The grouping of several elementary ratification files corresponding to different keys makes it possible to gain space and increase security.

One way to implement the invention is comprised of creating and managing so-called "Link" files wherein the body is identical to that of other files. The invention is comprised of being able to share the same file body among several files. This can be done by indicating, either in the header of the file or in its body, the address at which the data is actually located.

Fig. 3 represents two files, i.e., a target file 30 and a link file 31. The following description relates both to the case where these files are data files and to the case where they represent directories. These directories contain either a tree of sub-directories that provide access to data files, or data files that are attached to them directly, or both. The term "data" includes both non-executable data and executable data, or programs.

The target file 30 in this example is organized into two parts comprising a header 32 and a body 33. The header 32 includes a first group of parameters that are themselves known, i.e.:

- a type, which indicates whether the file is a directory or a data file;
- an identifier, which designates the file within a directory that contains it; this could be, for example, a name or a number; and
- access conditions which provide a list of access rights to a given file, for any user; they indicate, for example, whether or not the file is read- or write-accessible; in a known way, the granting of these rights can be subject to the presentation of keys or passwords.

The header 32 includes a second group of parameters that are specific to the invention, i.e.:

- a parameter <Link>, which can have two values: either the value 1, which indicates that this file is a link file, or the value 0, which indicates that it is not a link file; in this case, this parameter has the value 0;
- a parameter >Link<, which can have two values: either the value 1, which indicates that this file is a target file, or the value 0, which indicates that it is not a target file; in this case, this parameter has the value 1;
- a parameter A-Link which can have two values: either the value 1, which indicates that this file can be linked to a link file, or the value 0, which precludes this; and
- a parameter CA-Link which defines the creation conditions that the user must fulfill when intending to create a link between this file and a link file: these could, for example, define keys or passwords to be presented by the user.

Lastly, the header 32 comprises a reference RC indicating to the microprocessor of the card a binary value of a storage address RC at which the above-mentioned body 33 is stored.

In a variant, the body 33 is stored in memory immediately after the header 32, so that the mention of the reference RC is not necessary.

Moreover, if the target file 30 is of the "directory" type, the body 33 contains either a tree of sub-directories that provide access to data files, or data files that are attached to it directly, or both.

If, on the other hand, the target file 30 is of the "data file" type, the body 33 contains a set of data that is directly read-accessible, modifiable, or executable by the microprocessor of the card.

In a variant, the organization of the target file 30 could be different from the two-part (header and body) organization shown in Fig. 3. Thus, for example, the parameters of the header 32 could be distributed to specific places in the body.

The link file 31 comprises only one part, i.e., a header having the same structure as the header 32 of the target file 30, but with a content that differs in the following way:

- since it is a link file, and not a target file, the parameters A-Link and CA-Link are not generally used, except in the particular case described below;
- moreover, the "reference" is not one that relates to a potential body attached to the link file, but a reference RFC indicating the location in memory of a target file thus linked to this link file. In this example, it is the target file 30. The reference RFC is either, preferably, "physical" and constituted by a

binary value of a storage address at which the above-mentioned target file 32 is stored, or in a variant, "logical" and constituted by an access path indicating the identifiers of one or more directories from which the target file 32 is accessible.

In the very particular case of a dynamic management of the memory of the card, the latter can be reorganized by the microprocessor in order to optimize the use of memory space. The location of the files can therefore fluctuate; thus, so can their storage addresses. In this case, only the logical reference of the target file can be used easily, since the physical addresses constantly run the risk of changing. Taking the case described above as an example, the logical reference is: [CARD → APPLICATION 1 → Data file A1-1].

Therefore, it is clear that a link file is devoid of a body, but is linked to a given target file whose the body is thus made available to the link file.

It will be noted that in one particular case, a second link file, different from the link file 31, could be linked, not to a target file directly, but for example to the link file 31. The situation would then be as follows:

- the reference contained in the second link file would be that of the link file 31;
- the parameter CA-Link would advantageously be used in the header of the first link file to verify the conditions for creating the second link file.

In operation, when a file is selected by the user, a program of the microprocessor reads its header and tests its parameter <Link>. If it is equal to 0, the operation takes place according to the prior art: the body is directly attached to this header.

If <Link> is equal to 1, the file is a link file. Consequently, the program reads the reference RFC of this link file indicating the address or the access path of a target file containing a body indirectly attached to the header of the link file. Before making the content of the target file available to the user or to the microprocessor, the program performs the following verifications, consulting the respective headers of the link file and the target file:

- during the creation of the link file, it verifies that the type of the target file identified is identical to that of the link file; if it is not, the procedure for accessing the link file is interrupted;
- during each access to the target file, it verifies the fulfillment of the access conditions using a procedure that will be defined below.

Once these verifications have been performed, the program continues its procedure for accessing the content of the link file. If the size of the body of the target file is "zero bytes," i.e., if it does not contain anything, the program is interrupted and the card returns an error message. If not, the program searches for the information contained in this body, beginning at the address RC. Returning to the example of Fig. 2, it is assumed that the file A1-1 of the application directory 1 was created in the form of a target file, and that the files A2S1-1, A2S1-2, A2S2-1 of the service directories A2-S1 and A2-S2 were created in the form of link files. Consequently, the selection of a link file such as A2S1-1 provides access to the content of the target file A1-1.

The conditions for accessing the body of the target file, defined in the header of the latter, must always be fulfilled during the execution of a link between a link file and a target file. There are several possible strategies. The simplest of these is comprised of obeying the access conditions defined in the header of the target file; thus, access to the information in the target file via the link file is only accorded if the access conditions of the target file have been fulfilled.

Another strategy is comprised of taking the access conditions of the target file into account during the creation of the link file. In this case, it is necessary to verify that the access conditions written into the header of the link file include all of the access conditions of the target file to which it will be linked.

A third strategy can be applied when the access conditions are expressed in the form of a binary value; it is comprised of combining the two access conditions. Concretely, this operation can be carried out by executing an AND operation between the two values. Access to the information in the target file via the link file is only accorded if the access conditions of both the target and link files are fulfilled.

It is understood that the objective of these various strategies at the access control level is to prevent any possibility of circumventing the access conditions of a file by linking it to a file that has more favorable access conditions. Other strategies known to one skilled in the art could be used to ensure security of access.

An important refinement relative to security is comprised of using the parameter A-Link of a target file to prevent it from being linked to another file. If the value of this parameter is "1" during the creation of a link file attached to this target file, the operation is completed successfully and the content of the body of the target file is accessible by the link file. If, on the other hand, the value of this field is

"0," this target file cannot be linked to any other file. When an attempt is made to create a link file designating a file in which the field A-Link is equal to "0," the operation is denied and the card returns an error message.

A problem arises during the erasure, i.e., the deletion of link or target files. If a target file is erased, access via link files to the information it contained is no longer possible. The consequences of such an erasure are therefore not limited to just the directory that contained this file, but can have repercussions in other directories. The solution is comprised either of prohibiting the erasure of the target file to which one or more link files are attached, or of warning the user of the card that certain files will no longer be operational after this erasure. A first method is comprised of testing the value >Link<. If this value is 1, the file being erased is a target file. The operation is therefore either prohibited, or successfully completed, but with a warning; in the latter case, the control terminal of the card must erase all the link files linked to the target file erased.

In order to erase a target file without disturbing the rest of the memory, it is first necessary to erase all the link files that are attached to it. When the last link file has been erased, the indicator >Link< will have the value "0," thus indicating that there is no longer any target file. A simple binary indicator is not enough to count the number of link files without having to scan the entire nonvolatile memory of the card each time. One solution is comprised of providing a counter Cp-Link in place of the parameter >Link<. Advantageously, this is an 8-bit counter, which allows the existence of a maximum of 255 link files: this number seems to be quite sufficient for current applications. This counter is incorporated in the header of the target file.

During the creation of the target file, the counter Cp-Link is set to "00." With each new creation of a link file that is attached to the target file, the counter is incremented. With each erasure of a link file that is attached to it, the counter is decremented. Advantageously, during the selection of the target file, the value of the counter can be sent with the other header information; thus, the user can know the number of link files attached to the target file selected.

To solve the problem of the erasure of a target file more effectively, the program of the card is equipped with a command that can be activated from outside the card, which makes it possible to exchange the respective "link" and "target" statuses of the two files. Thus, a target file that has become a link file can be erased without consequences for the other link files. The content of the new target file

is then constituted by that of the old target file. This operation is particularly easy when the file bodies are physically separated into headers. For security reasons, the execution of this command is subject to the verification of the access conditions defined in the header of the old target-file, and possibly to the verification of the conditions for creating links between files, defined by the parameter CA-Link in the directory of the new target file. During the execution of this exchange command, the value of the counter CP-Link of the old target file is stored in the counter CP-Link of the new target file.

Fig. 4 illustrates a process for creating a file, whether or not it is a link file. In addition to the steps specific to the invention that relate to the link file, it includes certain steps that are themselves known and that relate to the creation of any file, no matter what its nature. In step 1, a file creation order is received by the card, accompanied by creation data: this data specifically defines the type and the identifier of the file to be created and, if it is a link file, the reference RFC (Fig. 3) of a target file to which it must be linked.

Next, the operating system of the card verifies that the creation of a new file is possible in the current directory (step 2). In effect, the creation of a new file may be subject to the prior correct presentation of keys defined by the access conditions of the header of the current directory. Then the operating system verifies whether their is still enough memory in the current directory to contain the new file (step 3). If one of these tests is negative, the creation order is interrupted (step 13), and the card then sends a message corresponding to the source of the stop.

The operating system then tests to see if the file created is a normal file or a link file (step 4). One important difference between a normal file and a link file, on which the test can be based, resides in the creation data, and primarily in the precise indication of the location of the target file (physical or logical address). If it is not a link file, the program jumps directly to step 12, described below. Otherwise, the information corresponding to the designated target file is searched for and analyzed in step 5; then the operating system performs a certain number of tests to ensure that the link between the designated target file and the link file to be created is possible.

First of all, it uses the creation data to verify that the target file actually exists (step 6). If on the other hand this data does not correspond to any file, the creation operation is interrupted and the card sends an error message (step 13). In step 7, the parameter A-Link of the target file located is tested. If its value is "1," the operation can be completed successfully. If not, the target file cannot be linked to any

other file. The creation operation is then interrupted and the card sends an error message. In step 8, the operating system tests to see whether the potential keys defined in the creation conditions of the link file, i.e. those defined by the parameter CA-Link of the target file, have first been presented. If this is not the case, the creation operation is interrupted.

In step 9, the operating system of the card verifies that the file types and the conditions for access to the information are compatible. To do this, the parameter TYPE of the target file is compared to that transmitted in the creation data. If the values are different, or at least incompatible, as for example in the case of a creation order attempting to create a link between a data file and a directory, or a link between a data file of the "public" type and a data file of the "secret" type, then the creation operation is interrupted and the card sends an error message. This test is optional, since another solution is comprised of setting the data received for the link file to be created, to the same value as that of the designated target file; in this case, compatibility is certain.

Lastly, a final test performed is based on the conditions for access to the information contained in the target file (step 10). It has to do with preventing the circumvention of the access conditions of the target file via a link file that has more favorable access conditions. One of the strategies described above is comprised of prohibiting the creation of a link file having less restrictive access conditions than those of the target file; the creation operation is then interrupted and the card sends an error message (step 13). Another strategy is comprised of adapting, and hence modifying, overly favorable access conditions of the link file, so as to make them at least as restrictive as those of the target file. In this case, the test in step 10 becomes a calculation operation which, if necessary, includes a modification of the access conditions transmitted in the command.

Once the test steps are finished, the creation of the link file can occur. In step 11, the header of the target file is updated. This involves mainly the parameter >Link< or Cp-Link. If it involves the parameter >Link<, the program verifies that it has the value "1," or if not, sets it to this value. If, on the other hand, it involves the counter Cp-Link, the latter is incremented by one unit.

Finally, in step 12, a new file is actually created, and the values of the header parameters of this file are determined in working memory from the creation data. These values are written into programmable nonvolatile memory. If it is a link file that has been created, a reference linked to the location of the target file (a physical or logical address) is written. Once all of the steps are finished, the

card returns a correct status message, and the new file created is operational.

The case of the creation of a target file will not be described in detail since, in its creation, this file is similar to a conventional file. It is only at the moment it is linked to a link file that it becomes an actual target file.

One particularly advantageous application of the invention relative to link directories is that in which an electronic purse directory is used by the card to allow payments. This directory contains elementary files containing keys, debit-credit areas, password validation areas, etc. A directory of this type can be used in various applications (transportation, restaurant, purchasing departments); therefore, each of them can contain a link directory linked to the electronic purse directory, which then becomes a target directory.

CLAIMS

1. A security module intended to cooperate with a data processing device and comprising data processing means and data storage means, which storage means store several files, characterized in that it comprises:

- link generating means for generating a link between at least one main file (30) and one auxiliary file (31), the main file having a given content (33) and being made accessible to the processing means in the storage means by means of location data (RFC), which link generating means associate said location data with the auxiliary file (31);

- branching means for making available to the processing means, when the latter execute an access request for accessing the auxiliary file (31), said content (33) of the main file (30) using said location data (RFC).

2. The security module according to claim 1, in which the auxiliary file (31) contains a parameter (<link>) indicating that it contains location data for a main file (30).

3. The security module according to claim 1, in which the main file (30) contains a parameter (>link<) indicating that its location data (RFC) is associated with at least one auxiliary file (31).

4. The security module according to claim 3, in which said parameter (>link<) is a value of a counter (Cp-Link) for counting a number of auxiliary files (31) that are linked to this main file (30).

5. The security module according to claim 1, in which the main file (30) contains a parameter (A-Link) indicating whether or not the making of its content available, when there is a request to access the auxiliary file (31), is authorized.

6. The security module according to claim 1, in which the main file (30) contains a parameter (CA-Link) defining creation conditions that the processing means must fulfill during the

* C L M S *

creation of a link with a given auxiliary file (31).