Smart Card Embedded Information Systems: a Methodology for Privacy Oriented Architectural Design

C. Bolchini, F. A. Schreiber

Data and Knowledge Engineering
Vol. 41, n. 2-3, pp. 159-182

Smart Card Embedded Information Systems: a Methodology for Privacy Oriented Architectural Design

C. Bolchini, F. A. Schreiber
Dipartimento di Elettronica e Informazione - Politecnico di Milano
P.zza L. da Vinci, 32 – I20133 – Milano – Italy
{bolchini, schreibe}@elet.polimi.it

Abstract – *The design of very small databases for smart cards and for portable embedded systems is deeply constrained by the peculiar features of the physical medium. Privacy concerns are relevant due to the fact that personal information may be stored on the card (e.g. medical records). We propose a joint approach to the logical and physical database design phases supporting the required security levels, on the basis that all information is stored on the Flash EEPROM storage medium, managed as a file system by the Smart Card operating system.*

Keywords: access methods, data structures, design methodology, federated Database, flash memory, Information privacy, security, personal Information System, Smart card.

1 Introduction

The Information Management area is currently seeing the growth, in number as well as in size, of applications that can profit of smart card physical support.

Though smart cards have been recognised among today as most secure portable computing devices [2][8][12][17], not much attention has been devoted to adapting database techniques to this highly powerful tool. In [2], the authors make a very thorough examination of the most relevant issues that arise in this context, by proposing a storage model complete with query cost evaluation, plus transaction techniques for atomicity and durability in this particular environment. To our knowledge, this is the most complete attempt to analyse in-depth the problems of DBMS design w. r. t. micro-devices.

The attention of [2] is mostly devoted to DBMS design techniques: it follows the traditional assumption that data structures and access methods should be designed, once and for all, by the DBMS manufacturer; the database designer should be mainly in charge of the conceptual and logical database design, limiting the Database physical design phase to the mere choice of fields to be indexed.

However, owing to the intrinsic limitations of the smart card medium, which pose unique challenges with respect to traditional DBMS based Information Systems, we believe that it is very important that also the database designer be aware of the physical constraints from the very beginning of the design process. There are two main steps where these constraints are to be enforced:

- partitioning data and procedures among the smart card itself and other fixed sites the smart card might be connected to;
- designing and evaluating on-card data structures and algorithms, and data manipulation policies (e.g. delete operation management) which optimise performance, power consumption and card endurance.

Moreover, some applications can involve sensible personal data that require a variable level of privacy. While privacy by itself is a legal and normative issue, the technical tools to enforce it are the domain of the system designer, who can choose among several security techniques and options, down to a major involvement in the design of the physical data structures.

The purpose of our research is to propose a full-fledged database design methodology. It should guide the database designer from the conceptual design step, carried out in the traditional way by using one of the well-known conceptual design models [9, 14], to the semi-automatic choice of the most appropriate data structures, access methods and memory allocation/management policies, based on the evaluation of the physical storage devices currently offered by smart card technology. To this purpose, we propose a logical-physical data model that should support the design of such novel database applications for on-card data management.

In this paper, after outlining the steps of the methodology, we will concentrate on the logical/physical database design phase with respect to the privacy issues and the management of secure access of different users to the Smart Card database.

The paper is organised as follows: Section 2 outlines the general privacy issues in Information Systems outlining the scenario and the main concerns w. r. t. the database information stored on the card. With the aid of a running example of a personal information system, Section 3 presents the reader with the proposed methodology for smart card Information Systems design and singles out the specific topic addressed by this paper, i.e. the logical and physical support for implementing a profile management system for accessing the Smart Card database. Section 4 introduces the basic aspects related to Smart cards, along with the most recent technological issues affecting the design of the privacy support, which is described in the following sections. Section 5 introduces two possible scenarios for organising owner-guests database accesses, while Section 6 defines the implementation of such organisation for supporting the privacy requirements within the proposed methodology for designing databases for Smart Cards. Future developments and research trends are discussed in Section 7, along with the final conclusions.

2 Privacy in Information Systems

Information privacy is one of the most debated topics on the social implications of informatics in the last years. The availability of every kind of information - even the most personal - in electronic form on some storage medium of some computer (e. g. in banks, supermarkets, medical authorities, lawyers, etc.) arises in citizens and consumers a great concern about the diffusion and the use of private data [25]. Regulating Authorities in different countries (OECD, Australia, Canada, Hong Kong, USA) are studying or issuing several behaviour codes and principles [7][11][13][16][19-21] in order to discipline and control people *dataveillance*, that is all those monitoring activities performed "not through people's actions, but through data trails about them" [7].

Issues about privacy were not born with information technologies; they have been a crucial issue in many social and legal systems as to the amount of privacy an individual can be entitled in the frame of the more general interests of the society (a noticeable example is that of people refusing to vaccinate against infective diseases which could be spread among the whole population).

We can distinguish among several dimensions of privacy:

- privacy of the person;
- privacy of personal behaviour;
- privacy of personal communications;
- privacy of personal data.

But for the first item of the list, all the other aspects are relevant to Information Systems designers, since personal behaviours can be monitored and inferred from data and communications.

One of the most referenced documents about information privacy was issued by OECD in 1980 [11]; it sets some general principles for protecting privacy and, in particular:

- limitations on the *collection* of personal data;
- limitations on the *legal use* of collected data, for which a personal consent is required;
- safeguards against unauthorised access, use and disclosure of personal data.

As to the interaction between users and data, three levels of privacy can be established:

- *identified* transactions, which directly or indirectly link data to a particular person;
- *anonymous* transactions, which preclude any identification of the parties related to transaction data;
- *pseudonymous* transactions, in which no direct links exist between the transaction data and the involved parties, but an association can be established between some of the data and some of the parties, provided that additional data be accessed and particular procedures be followed.

An easy way to achieve pseudonymity is to tag transactions with aliases, which can be resolved by means of cross-index tables, maintained by a security manager and subject to technical and legal protection.

2.1 The smart card issues

The availability of quite a large amount of storage and of processing power on the smart card can be both a plus for privacy protection and a threat for it, depending on the architecture of the systems on which the card is used. In his comprehensive report on the use of smart cards in the retail financial sector [7] Roger Clarke summarises the pros and cons of the smart card medium w. r. t. privacy; we briefly discuss some of them in the following.

Enhancement in information privacy is provided by:

- *storing data only on the card*; this feature puts the card holder in full control of personal data and he/she can control also possible backup copies of data for recovery purposes. However it is very unlikely that the same data will not be stored also in other locations of an Information System, since in this case the potential benefit can, at least in part, vanish;
- *using non-identified cards*; this feature is used in prepaid cards for using particular services as transportation, telephones, etc. instead of using cash transactions. However anonymity can be violated if the card itself (not its holder) has an identification code which can be tracked by external log procedures and successive data analysis and comparisons (e.g. data mining and knowledge discovery procedures). For these reasons, transaction trails are among the most relevant concerns to privacy for the smart card user, independently of the application;
- *segregated zones on the card*; this feature amounts to merging a number of separate cards on a single one by strongly separating, at the storage level, data belonging to different applications. This entails data duplication and the related well-known consistency problems. A Smart Card Information System, on the contrary, should strongly benefit of data integration, and privacy should be obtained through different mechanisms: for instance, views can be used at the conceptual/logical level, bounding address registers at the physical level, cryptography at the storage level;
- *support of Public Key Cryptography*; smart cards have been proposed as a supporting medium for digital signature procedures, biometrics measures, and other identification procedures. As such, they can be used as privacy enforcing procedure support in order to clearly recognise the partners in a transaction.

2.2 Privacy constraints implementation

Implementation of privacy constraints is mainly based on security techniques, in particular authentication and access control [5][10].

Authentication is the mechanism that checks the user's identity by means of:

- something the user is acquainted with (e.g. a password);
- something the user owns (e.g. a magnetic badge);
- physical characteristics of the user (fingerprints, signature, etc.).

Since authentication often uses cryptography and possibly third parties through complex communication protocols, the computing power available on the smart card and the ability to store complex patterns make it an ideal medium for supporting authentication procedures. From the point of view of privacy, authentication allows the user to be assured about the identity of the partner (person or procedure) whom personal data are communicated to.

Once the actual identity of the user has been ascertained, the system must control the user's right to access the single pieces of information. There are lots of security mechanisms for granting access to data, for which we refer the reader to [5][22-24].

In this paper we are mainly interested in mechanisms which work at the conceptual/logical and at the related physical level without entering into coding and encryption problems at the storage level.

3 A Smart Card Database Design Methodology

In order to meet the design requirements mentioned in Section 1, current design methodologies should be adapted to accommodate physical aspects which are relevant also at the logical design level; the aim is to provide the designer with guidelines for identifying the structural aspects of the database elements (index, sorted/unsorted, …) and their temporal duration properties. Moreover, since privacy aspects have a great importance in a uniform storage medium with tightly packed sensitive information, possibly belonging to different "owners", as in multifunctional smart cards, their impact should be considered as early as possible in the design phases.

A noticeable feature of smart card databases is that the owner and all his/her information have a peculiar role in the ER schema; in fact, while the card owner's data constitute the virtual centre of the database, they often amount to a unique entry, i.e. a singleton record. This role could be compared to that of the home page of a Web Site, which is just a singleton entity in the site schema, or to the *fact table* of a relational star schema for a Data Warehouse [1].

In order to discuss the proposed methodology for designing Smart Card Database Systems (SCDBS), a running example will be introduced: a personal information manager concerning an individual within the governmental/medical institutions

3.1 The personal folder database

An application field in which the use of Smart Cards seems promising is that of Personal Portable Folder Information Systems (PPFIS). These systems can store all the information pertaining to an individual such as: personal identification documents, health records, driving/flying/sailing licenses, insurance policies, bank accounts and credit cards. Other similar applications could manage travelling and shopping information such as supermarket fidelity cards, airlines mileage points, skipasses etc.

Although much of this information is already stored in portable magnetic cards or even in smart cards, one has to carry a dozen plastic cards, one for each information/application domain. The PPFIS concept allows all the data which could be of immediate need to an individual, wherever he/she is, to be available on a single smart card using simple terminals which can equip e.g. patrol police cars or ambulances [4].

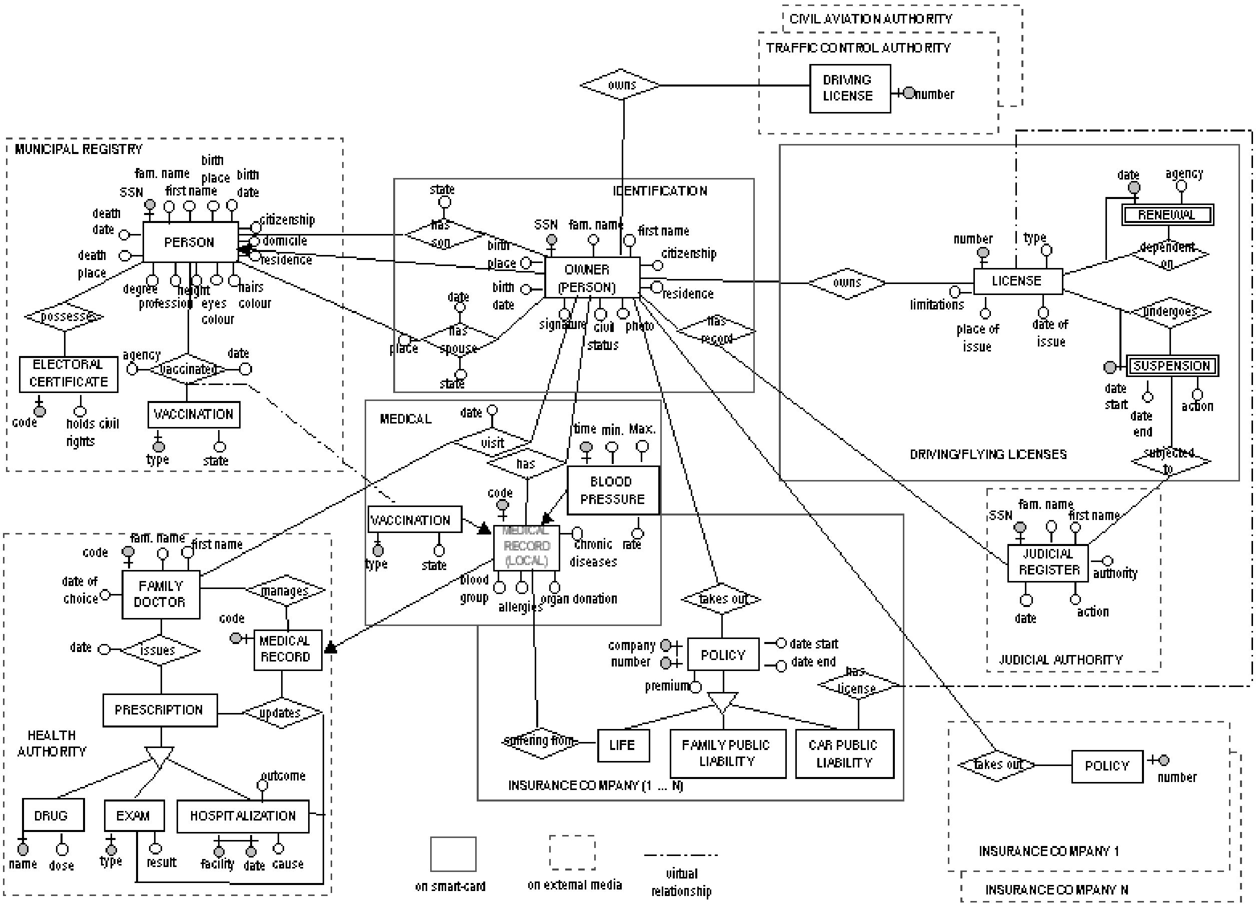


Fig. 1. The Personal Portable Folder Information System.

In Fig. 1 the conceptual schema of an example PPFIS is shown as an Entity-Relationship (E-R) graph. We notice that several "functional areas" can be highlighted whose data are managed by different owners:

- the municipal registry;
- the health care authority;
- several insurance companies;
- several mobility authorities;
- the judicial registry.

Moreover, the data belonging to an individual can be partitioned into those stored on the smart card (continuous boxes) and those stored on other media (dotted boxes). The connection of the smart card to the external data sources can be required to update the "on-card" data or to answer complex queries that cannot be answered on-card.

The result is a "star schema" distributed database with both horizontally and vertically partitioned data items similar to a ROLAP Data Warehouse schema [1]; the peculiarity here is that the "fact table" constituting the star centre is made of a single record.

The connections between the central entity and the other parts of the schema can be made by:

- entity duplication (e. g. DRIVING LICENSE, INSURANCE POLICY);
- entity subsetting by horizontal (e. g. OWNER) or both horizontal and vertical partitioning (e. g. MEDICAL RECORD);
- distributed relationships (e. g. HAS SON, HAS SPOUSE, etc), in which case the table implementing the relationship must be stored on-card.

We must also notice that some relationships between on-card entities and also between on-card and external entities have been added to the schema with dotted lines as "virtual relationships". These virtual relationships violate the pure star schema, by adding some threads, in order to reduce the number of necessary join operations and speed-up queries which could be frequent or which could be made through narrowband terminals. However the same effect could be obtained by choosing a special purpose architecture for the smart card processor which, by itself, can optimise the particular join operations; this solution also enhances security, as discussed in Section 3.3.

3.2 Privacy issues

As we previously mentioned, privacy is an issue in PPFISs. Actually the identification of different functional areas, which are connected only through the star centre, allows a clear separation such as that obtainable by means of the views mechanism. We notice that this example is different from the *segregated zone* case of Section 2.1, because *information is actually shared* among the different applications through the star centre. Therefore the first step toward information isolation is a careful conceptual design of the database, not a physical separation of storage areas.

However there are several different levels of isolation:

- among different on-card functional areas (e. g. driving licenses, medical records and insurance information);

- among different on-card instances of the same functional area (e. g. insurance information from different companies);
- among the smart card and external ISs.

These isolation levels can only be achieved by applying logical and physical security techniques.

3.3 The methodology

Fig. 2 shows the whole methodology we propose for smart card database design. It is composed of a common track and two branches. The common track, which is borrowed from distributed/federated database design methodologies [6][18], takes care of the conceptual and logical aspects; the lower part deals with "on chip" features: the left branch concerns the privacy/security aspects while the right branch concerns the physical memory data structures and algorithms.

While shortly discussing the whole tree, in the following chapters we shall mainly focus on the problems related to the lower left branch, the choice of data structures having been dealt with in [3]. Notice that steps denoted by the people icons are a burden of the designer and an input to the procedure while those denoted by PC icons can be automated.

1. The relevant information is chosen and modelled. Homogeneous information areas are singled out and the corresponding views are defined at a conceptual level, regardless of the target storage media. Referring to the PPFIS, this step corresponds to the choice of the 5 functional areas listed in Section 3.1, and to their separate representation in a suitable formalism (e.g. an Entity-Relationship graph).

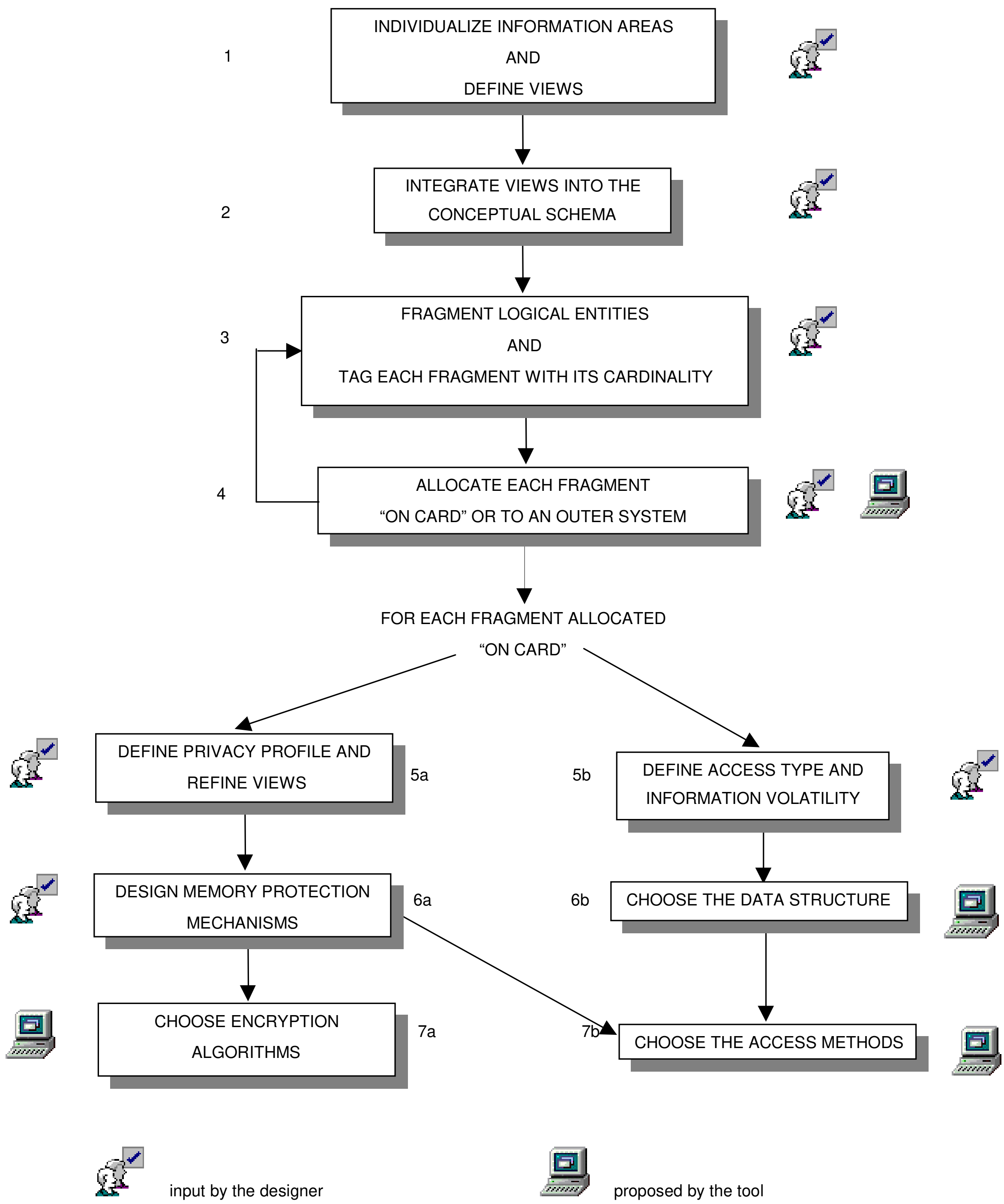


Fig. 2. The entire methodology.

2. views are integrated into a global conceptual schema and possible representation and semantic conflicts are resolved. With this step the complete schema of Fig. 1 is built.

3. logical entities are fragmented, possibly both horizontally and vertically, in order to separate "first need" information for a set of applications managed by very different user classes. A rather close estimate of the fragment cardinalities is to be made at this step. In this step the

continuous and dotted boxes of Fig. 1 are defined, together with some subsetting and duplication of entities (e.g. MEDICAL RECORD, INSURANCE POLICY etc.).

4. "first need" fragments are allocated to the smart card, the other fragments belonging to other sites of the Information System. This step includes the allocation of the tables corresponding to relationships such as HAS SON, HAS SPOUSE etc.

At this point a first comparison is to be made between the fragments and the smart card storage capacity: possibly fragmentation and allocation criteria shall be reconsidered.

5. a) the privacy profile, i. e. the access rights for each fragment and each user class, is defined and the relevant constraints are included in the views definitions: for instance, the first care personnel of an ambulance should know about the patient's blood pressure record, but not about his/her possible insurance policies, while the traffic authorities should only have a read-only access to license relevant information of the judicial registry;

   b) the type of access mode (read only, read/write) and the volatility (for example in terms of update/query ratio) are estimated for each fragment;

6. a) memory protection mechanisms are to be designed in order to prevent unauthorised users to access sensitive information;

   b) the type of storage is chosen with respect to the volatility features of each fragment (EEPROM, flash, …) and the most convenient data structures are chosen;

7. a) possibly an encryption algorithm is chosen for some very sensitive data;

   b) access methods are chosen for the data structures defined at step 6.b) under the constraints coming from step 6.a).

At step 5.b) the information is gathered in order to allocate the same type of information, in terms of volume and volatility, on the same sets of blocks. In particular, if asymmetric Flash-EEPROM is considered (see Section 4.2), where four different block sizes are usually used, very low volume information can be stored in the smallest block while high volume information can be stored in one or more of the biggest blocks. At steps 6.b) and 7.b) the physical data structures and the relevant access algorithms are designed. Steps 5b, 6b and 7b are discussed in [3].

4 Smart Card Technology Issues

Smart cards are devices that allow information storage and processing, but they need to interact with an active device providing the necessary power supply. Different types of smart cards can be identified considering both the technology adopted for the memory device and the possibility to elaborate on card information.

The most common and simple type of smart card is the *magnetic stripe* card that stores a record of information on a magnetic band. More sophisticated smart cards contain a chip. *Memory cards* contain only memory chips that can be preloaded and depleted and the active role is performed by a terminal, e.g. the telephone.

For our purposes, we are interested in smart cards with autonomous processing capability: Microprocessor Multifunction Cards (MMC's) expand the application field of smart cards by embedding a microprocessor and, often, a cryptographically enhanced co-processor within the card. The simplest architecture is the *processor* card that contains a microprocessor, a simple cryptographic co-processor, and blocks of memory including RAM, ROM, and a non-volatile memory (usually EEPROM or Flash-EEPROM). More sophisticated cards, *crypto* cards, are based on the *processor* card architecture where the simple cryptographic co-processor is replaced with an advanced one. The improvement in the *crypto* cards consists in the possibility to enable public key encryption, whereas *processor* cards can only use private key encryption. For the application environment described in Section 3, the target architecture is the Microprocessor Multifunction Card.

4.1 MMC (Microprocessor Multifunction Cards)

The microcontroller used in Smart card applications contains a central processing unit (CPU) and blocks of memory, including RAM, ROM, and re-programmable non-volatile memory (NVM). RAM is used to store executing programs and data temporarily, while ROM is used to store the operating system (Card Operating System - COS), fixed data, standard routines, and lookup tables. The re-programmable non-volatile memory is used to store information that has to be retained when power is removed, but that must also be alterable to accommodate data specific to individual cards or any possible changes over their lifetimes. More specifically, the smart card NVM constitutes the data storage for the database. Based on this consideration, technology issues have a significant impact on the overall system performance.

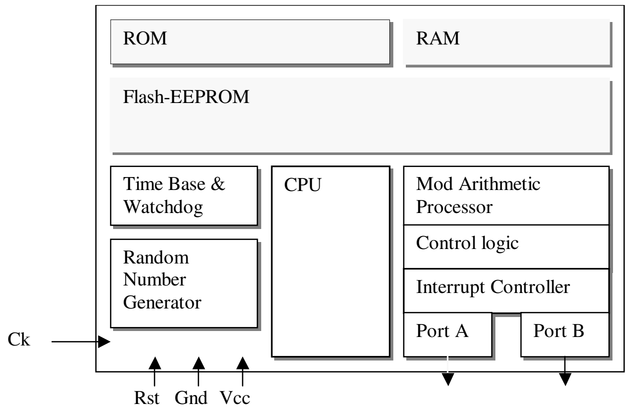


Fig. 3. A schematic representation of a smart card microcontroller.

An example of today's smart card controller includes an 8-bit CPU, 128 to 780 bytes of RAM, 4 to 20 KB of ROM, 1 to 16 KB of EEPROM and, optionally, an on-chip hardware encryption module. We will use such architecture as our reference model (see Fig. 3). However, there are several plans to improve the architectural characteristics of the microcontroller; as an example, there is a plan from Hitachi and Infineon (a semiconductor spin-off of Siemens AG) to produce a new 0.25µm device that combines an increased Flash-EEPROM memory with a 16/32 bit RISC processor.

A preliminary analysis concerns a comparison of the features and performance of the two most common NVM architectures, in order to focus our analysis on a specific memory type. In fact, although classical EEPROM and Flash-EEPROM are functionally equivalent (both are electrically erasable re-programmable non-volatile memory) they differ in terms of performance, integration density, cost, read/program/erase methods. The comparison that follows seems to the advantage of Flash memories as more promising for future MMC.

In classical EEPROMs (from now on called simply *EEPROMs*), the contents may be erased and programmed at byte or word level. The erase operation is contained in the program cycle and transparent to the user; *read time* is approximately 35 nsec, *program time* is 1 msec/byte while *erase time* is not applicable.

In Flash-EEPROM (form now on called simply *Flash*) *write* data operations are *program*ming operations that can only be performed if the target location has either never been written before, or has been previously erased. Further complication and space/performance cost arise from the fact that the *erase* operation works only on blocks of data one at a time. Flash memories are divided into blocks and each block can be erased separately; *read* and *program* operations work independently of the blocks, any byte (word) can be read or programmed separately.

Thus, in Flash the minimum erase element is larger than the minimum read/program element; the erase element is a page, sector or the entire device, while the read/program element is a byte or word. Table 1 compares the main features of EEPROMS and Flash memories.

Table 1. EEPROM and Flash-EEPROM technical characteristics.

	EEPROM	**Flash-EEPROM**
Read (nsec)	20 to 150	20 to 150
Program (μsec/byte)	1000	2 to 7
Erase (s/block)	N.A.	0.7/0.8
Cell Size (μm^2 – 0.6 μm tech)	30	6,4
Cost per bit	Medium	Low
Endurance (program, erase)	100.000	100.000

Summarising, although Flash memories are more interesting than EEPROMs in terms of read/write time and integration density, they present a disadvantage concerning the erase operation that is particularly expensive w.r.t. time and power consumption. However, these drawbacks can be strongly reduced if a correct erase policy is applied. For example, if a block is *erased only when its entire content has to be modified*, an EEPROM memory is approximately 50 times slower than a Flash one in writing 64k bytes in the same single block for *n* times. Although this example represents a special case, an appropriate set of policies can make Flash memories much more interesting than EEPROMs. As a consequence, also considering the technological trend we will focus on an architectural solution based on Flash Memories.

4.2 Flash memories: dimension, power and timing issues

Flash memories can be classified into two categories: *uniform* block type and *asymmetrical* block type (Fig. 4). In the uniform block architecture, all blocks have the same size (16KB, 64KB, 128KB) and the number of blocks ranges from 8 to 64. Asymmetrical block memories usually contain four different sizes of blocks; more precisely, there are typically one boot block, two parameters blocks, one small main block and several normal main blocks; in a 512KB Flash Memory, the block dimension is 16KB, 8KB, 32KB and 64KB (seven blocks), respectively. It is worth noting that, even if this type of architecture is designed to store the boot code of the system (boot block), the system parameters (parameter block) and, finally, code and/or data (main block), this architecture results particularly significant if data properties (volume and volatility) are correlated with block dimension.

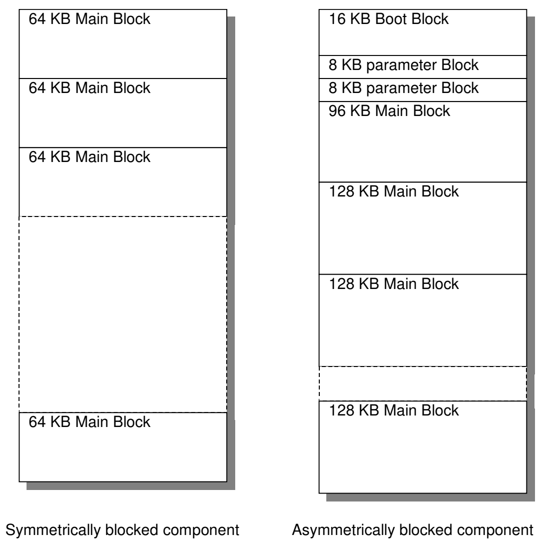


Fig. 4. Flash memory organisations.

Power requirements, in terms of current needs, vary depending on the operation that has to be performed. A *read* operation requires an average of about 10mA whereas *program* and *erase* operations require an average of about 20mA.

As introduced in Section 4.1, access time depends on both the operations and the mode: as far as *read* access is concerned, random *read* requires about 35ns whereas *read page mode* and *read burst mode* (when supported by the memory) requires about 150ns to access the first page or data and 20ns for subsequent reads (in a page mode for access within the page). Concerning programming time, each program operation takes approximately an average of 5µs per byte or word. Erase time depends on block dimension: typically a 64Kbyte block takes about 0.7s. Another consideration concerns the endurance of a Flash: in fact, a Flash memory works for a 100,000 program/erase cycles. These data are all reported in Table 1.

It is worth noting that the time to erase a block is approximately 10 times the time to program a block that, in turn, is 100 times the time to read a block; as a consequence, our effort is thus to reduce the number of erase operations in order to achieve good performance, reduce energy consumption and to extend the life of the device. These aspects have been taken into account when proposing the implementation of the mechanism for supporting privacy management discussed in Section 5.

4.3 The file system

Data management on the Smart Card EEPROM is a task of the file system as part of the operating system; most modern Smart Card operating systems allow files to be created, enlarged, erased and locked, as well as allowing other file management functions, all within limits imposed by specific security conditions []. The file system, which manages data in the EEPROM, is often hierarchically structured in three basic elements (Fig. 5a):

- a *master file* (MF) component
- a *dedicated file* (DF) component and,
- an *elementary file* (EF) component.

The *master file* is the root of the file hierarchy; there is only one *master file* on a smart card; it is implicitly selected when the Smart Card is reset. Equivalently to a DOS/UNIX system, the root contains all other directories and files. It is a special type of *dedicated file* and represents the entire extent of the Smart Card memory that is available for the file region. The *dedicated file* is a container for other *dedicated files* and/or *elementary files*; a *dedicated file* may contain from zero to many *elementary* and/or *dedicated files*. The hierarchical structure of the smart card file system is shown in Fig. 5b.

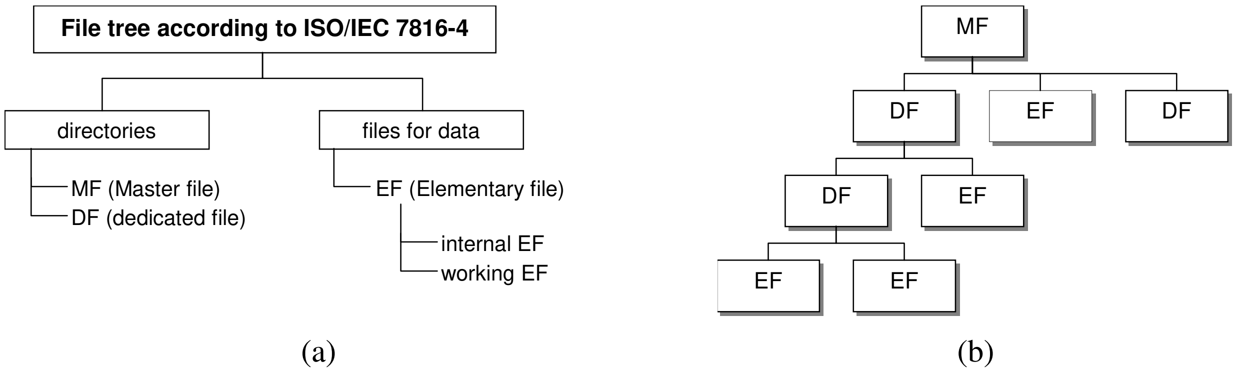


Fig. 5. (a) Classification of Smart Card file structures according to standard ISO/IEC 7816-4 and (b) a generic smart card file system architecture.

An *elementary file* is a file that contains data. There are two variants of *elementary files*: an *internal EF*, which is to be used by applications on the card, such as, for example, the materialisation of a view, and a *working EF*, which is used to store information used by an off-card application.

The standard (specified in the ISO/IEC 7816-4 standard document) defines four types of working elementary files:

- transparent;
- linear, fixed-length record;
- linear, variable-length record;
- cyclic, fixed-length record;

The transparent and linear, fixed or variable, file organisations are similar to those present in the disk based file system. In particular, a *transparent* file is organised as a sequence of bytes while *fixed*- or *variable*-length record files are organised in records. The main difference between fixed and variable length record architecture consists of the overhead of the variable-length record organisation, both as to the read/write access time and to the management of the file system.

A cyclic file, on the contrary, is significantly different from the file we find in the typical disk based file systems. A cyclic file is organised as a *ring of records* where each successive write takes effect on the next free physical record in the ring while any read operation is performed on the last written physical record. This type of management aims at spreading the erase and write operations across a set of EEPROM memory locations positively affecting the endurance of the card's non-volatile memory. Furthermore, this cyclic structure is managed for supporting the circular data logical organisation proposed in [3] where the authors provide a mechanism for maintaining a constant window of valid records, taking into account the technological environment. In fact, when a record needs to be erased, the entire block containing it must be erased and re-written. As a result, if the ring of records does not have any spare record, once the ring is full, each time a new record needs to be stored, the entire structure has to be saved in RAM, erased and copied back to the EEPROM except for the last record, to be substituted by the new one. The data structure proposed in [3] uses dummy records to avoid the costly process (in terms of time, power consumption and device endurance) introducing an area overhead. As a result, the standard physical cyclic structure can be used efficiently to keep track of the single last record added to the file, whereas the data structure model we proposed in [3] (the circular list relation) is aimed at maintaining the last *n* records added to the relation. This latter model acts on the relation and is meant to be independent of the physical file management, even though the two entities may be the same one.

To manipulate the Smart Card file system, the application level protocol defines a collection of functions for selecting, reading, and writing files. Some of these functions are discussed, briefly and qualitatively, in the following.

4.3.1 The Select File Command

The *select file* command is used to activate a logical connection to a specific file in the smart card's file system. Once a file is selected, any subsequent file manipulation commands will operate on the selected file. It is worth noting that any access to the smart card's file system is referred to a single file; however, it is possible to have simultaneous multiple logical connection (*channels*) between the reader-side application and the card and commands to access different files are multiplexed by the reader-side application, allowing different files on the card to be in various states of access by the reader-side application at the same time.

4.3.2 The Read (Write) Binary Command

The *read* (*write*) *binary* command is used by a reader-side application to retrieve (to put) some information from (to) a segment of an *elementary file* on the card. The EF must be a transparent file. If a *read* (*write*) *binary* command is issued on a record-oriented *elementary file*, the command will abort and an error indicator is returned to the reader-side application. As far as the *write command* is concerned, the performed operation can either set or clear a set of consecutive bytes, depending on the attributes passed from the application to the card.

4.3.3 The Erase (Update) Binary Command

The *erase* (*update*) *binary* command is used by an application to erase and write a contiguous sequence of bytes in a segment of a transparent *elementary file* on the card. The command is aborted, returning an error indication, if the command is applied to a record-oriented file.

4.3.4 The Read, Write, Append, Update Record Command

These commands have the same behaviour as the binary commands briefly discussed in Sections 4.3.2 and 4.3.3, and the difference is the fact that they work on a record-oriented elementary file, whereas in the present case the basic element id a record in an *elementary file*. As indicated for binary commands, the application to an incorrect file type will abort the command returning an error indicator.

4.3.5 The Get (Put) Data Command

The get (put) data command is a command sent by an application to read the contents of (to put information into) a data object stored within the file system on the card. This command tends to be very card-specific, i.e. the definition of what constitutes a data object varies widely from card to card.

5 Privacy management: the logical model

The methodology we propose for supporting privacy management for the Smart Card database consists of two aspects: a logical design of the database access profiles and the underlying physical mechanism for enforcing the protection of data from undesired tampering and accesses. The logical support we define consists in the design of an access definition environment, where the database designer will specify read and write permissions for each possible user of the Smart Card database, completing the design with the indication of the logical views for allowing a customised access to the data. With this in mind, we analyse the possible situations following a preliminary successful authentication of the user. It is possible to identify three kinds of roles when accessing the Smart Card database:

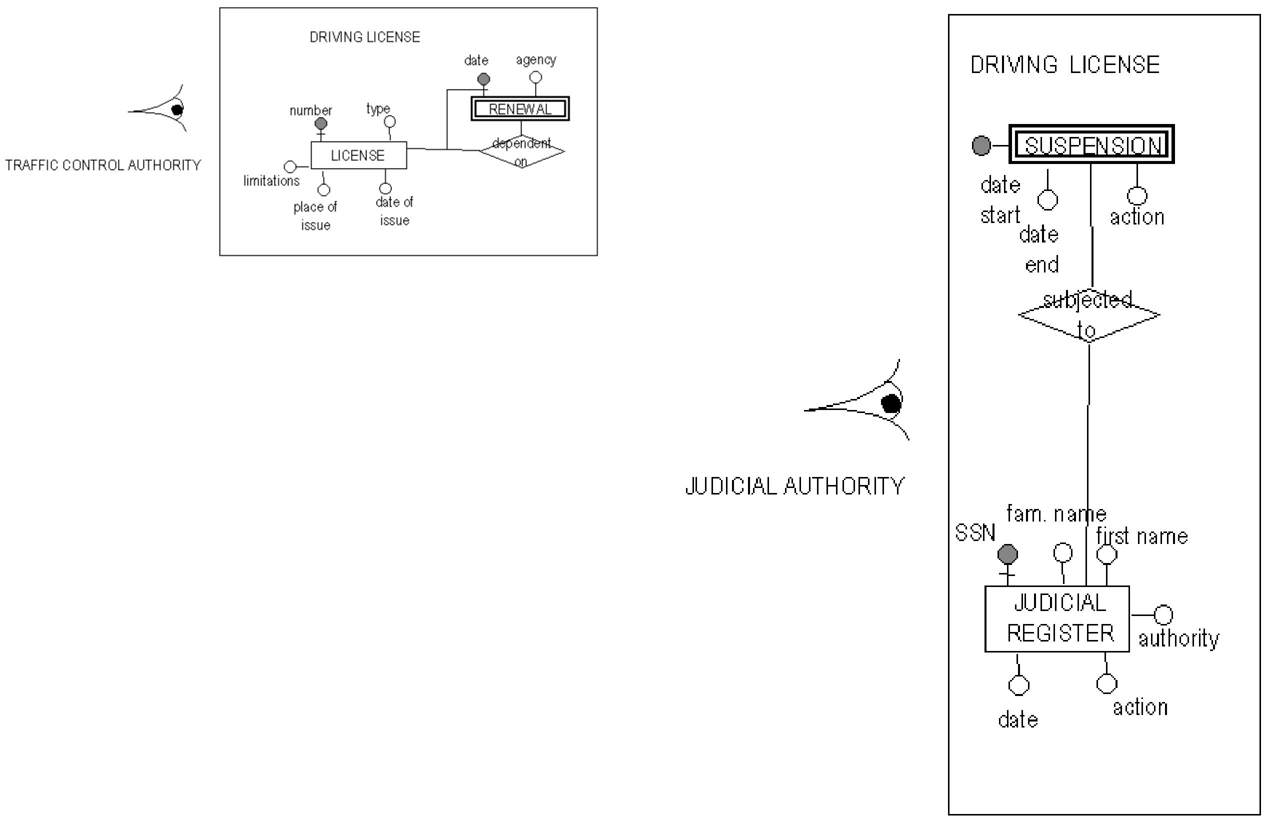
- the *data owner*,
- the *guest*, and
- the *card holder*.

A *data owner*, in general an authority or institution, has full permissions to access its own data, in read and write mode; a *guest* is any other institution or generic user accessing the card data with read only permissions on part or all of the stored data. Altogether, an institution is the owner of its own data and a guest for other institutions' data; there are also institutions accessing the card as guest, not owning any data on the card. The third role is the *card holder*: although carrying the card, he/she is not the owner of the information it bears, but only a guest for all the data issued by public/private institutions. The main characteristic of the holder's *guest* role is that he/she has an unrestricted access to all data, with read only permission, whereas other guests will only be entitled to read specific portions of the data. Furthermore, if the card holds also the holder's personal data that he/she manages, then the holder will also be the owner of part of the data, having a full access to them.

Having identified these three roles, a further characterisation is required for the *guest* definition and identification, with respect to the possible scenarios represented by the database stored on the Smart Card.

Consider the PPFIS example of Section 3.1: each institution provides its own data that is sometimes related to data of other institutions, e.g. traffic control authority and judicial register. Each institution writes, reads and updates its own data on the card and makes all or part of its data available to other users of the card, considering them as guests. More precisely, from the logical point of view, the institution has direct access to its relations and provides views for others to access data.

Therefore, referring to Fig. 1 as an example, we can define three views on the part of the schema related to the driving licence: Fig. 6.a shows the *data owner* view of the Traffic Control Authority; Fig. 6.b shows the *data owner* view of the Judicial Authority; both of them have full write rights on their view. Fig. 6.c shows the *guest* view of the Judicial Authority having been granted only read rights by the data owners.



(a) (b)

Fig 6. (a) Data owner view for the Traffic Control Authority and (b) for the Judicial Authority.

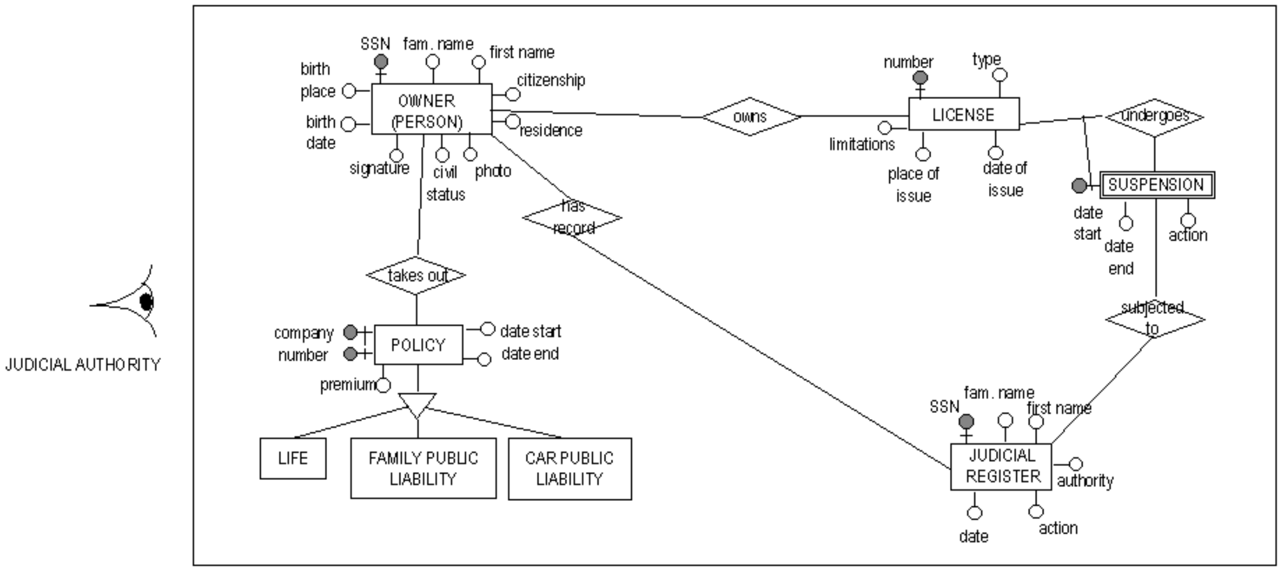


Fig. 6 (c) Guest view for the Judicial Authority

This logical organisation of data is on top of a system of integrated encryption at a lower level, where access is allowed only after a positive authentication. Therefore we will not deal with the security aspects related to data encryption, thoroughly studied in [15]; rather we concentrate on the mechanism for providing different privacy profiles to the users of the Smart Card and its database.

The PPFIS database actually provides a complex example of interacting owners and guests, each of them needing a customised access to data. On the other hand, it is also possible to have a more simplified situation, where each owner has a full access to its data and provides a generalised read-only access to a subset of them, to the other users of the card. In this case, the views will provide all the non-protected data and each user is seen simply as a generic *guest*, who will select the interesting subset of information from the available views.

As a result, two logical schemas have been envisioned for privacy management, based on the owner-guest relationship: a *customised* owner–multiguest schema and a *generalised* owner–guest schema. The former is appropriate when data have relevant sensitivity and each user of the card database needs to be characterised in terms of read accesses to other users' data, as in the PPFIS example. The latter schema is appropriate when the stored data have privacy requirements but either all (or part of) the data can be accessed by a guest or not. An example of a personal database with data with a limited level of privacy, appropriate for a generalised owner-guest schema, is represented by an on/off University Campus student card. The card holds all information concerning the student along with his/her exam records (from the University), information for lodging (from a Housing association), library accesses, reservations, purchases, etc. In this example there are different owners for the data, and each owner will either allow or prohibit access to its data, independently of the guest accessing them, since no distinction among guests applies.

As a first step in our methodology, thus, the database designer will select one of the two schemas, to support the privacy requirements, according to the access profiles specified by the data owners.

Let us consider first the *customised* owner-multiguest schema; each authority or institution may be one or more of the following figures:

- an owner of the stored data, with full access permissions,
- a guest accessing specific data in a read-only fashion.

It is quite burdensome and sometimes impossible to define a priori a set of views to be made available to possibly different users, identifying, for instance, *group*s of users (i.e. medical institutions users) and associating the necessary access permissions. In fact, this solution is characterised by the following costs:

- definition of views,
- identification of groups of users,
- customisation of access permissions for users within groups.

This hierarchical organisation is typical of standard database systems [1], and leads to the definition of access permissions for all the possible users of the card database, also adding an *unlisted guest* with read-only access permission restricted to data with no privacy requirements. This schema is rather complex when considering the users' permissions definition phase, yet it provides the necessary mechanisms to manage highly private data. The design of the group and user access permissions needs to be carried out at database design time; later modification to the database in terms of new institutions sharing data or accessing other's data may imply quite an overhead. In fact, data structures and physical organisation are selected on the characteristics of the database tables and relationships (as provided by the rest of the proposed methodology [3]). The introduction of new tables and views could require the reorganisation of the existing data in order to achieve acceptable performance in accessing data. Given the technological issues related to the Flash EEPROM memories, data reorganisation causes time and performance costs. The *unlisted guest* aims at dealing, at least for a transient period, with the subsequent introductions of new institutions or authorities that should be able to access data on the card; this user will though still be entitled to read data but only with low privacy requirements. When the database structure is significantly out-of-date and too many users need to be managed via the unlisted guest, a database maintenance phase is required, to upgrade the unlisted guests, updating the database and its permissions' system.

In the database design phase each of the data owners will specify:

- its relations (which will be fully accessible, read, write, delete, update permissions),
- a set of views on its relations, and for each view
- a list of guests with read permissions

The list of guests is created on the basis of the other owners of data in the database and with respect to other guests (institutions accessing the data).

The *generalised* owner-guest schema is appropriate when a simpler access control is required, where each user is seen as a uniform, unvarying *guest* to others' data. Each institution thus will provide a set of views to its data, to share all the visible information, independently of who will access it. This approach has a limited cost, only constituted by the definition of the views of the relations, allowing read-only access.

In the generalised schema, during the database design phase each data owner will specify:

- the owned relations (which will be fully accessible, read, write, delete, update permissions);
- a set of views on the owned relations.

The resulting design task is simplified still guaranteeing the desired level of data privacy for the part of information that can be accessed by others. It is worth remembering that underneath this logical masking of data, information is encrypted and before accessing the database a successful authentication of the user is required. The simplified environment of the generalised owner-guest schema only refers to the organisation of the database access permissions; in fact, while user authentication aims at guaranteeing that the user be entitled to access the database, views define which part of the database data can be accessed by the user.

6 Implementation of the privacy support

The definition of user access profiles according to one of the selected schemas as described in the previous section acts at a logical level of the database design to enforce data privacy. Nevertheless, underneath the logical protection, a physical approach is also proposed, to prevent a breach in the database security. This protection is pursued by means of providing a selective view of the file system, according to the user, so that if the logical database protection is bypassed, data can still be kept unavailable.

The standard ISO/IEC 7816-7 also defines a subset of SQL for Smart Cards (Smart Card Query Language – SCQL) acting on a suitable file structure for managing the database information; the layout of this structure is not standardised [15]. Yet, in order to manage databases, a common format system table is maintained by the card and contains information necessary to manage the database structure and access. There are three basic system tables:

- the object description table,
- the user description table,
- the privilege description table.

The object description table contains information about the tables and the views stored in the database. The user description table contains basic information about the user(s), and their access to the database(s). The privilege description table contains information about the privileges to the database tables and views for each user. Privileges describe which tables and views can be accessed by which users and which actions can be accordingly performed.

To access the information contained in the system tables, views on these tables can be created. A view on a system table is called an *SCQL dictionary*. The only action a user can perform on a dictionary is reading.

We propose to provide different *SCQL dictionaries* for the different users accessing the card, making the masking task transparent to the upper logical levels of the application. In order to implement such a view system for a database, we base our approach on the standard smart card file system infrastructure (discussed in Section 4.2), by providing each physical user with a *customised* view of the file system, depending on his/her access permissions.

Let us consider a generic file system constituting the physical layer of the Smart Card database: for each standard file there is an owner having full access permissions; eventually there are also guest permissions for read mode (Fig. 7a). The same file system, viewed from the point of view of a specific institution, consists of a set of files owned by the institution and, possibly, some files with a read only permission (see Fig. 7b).

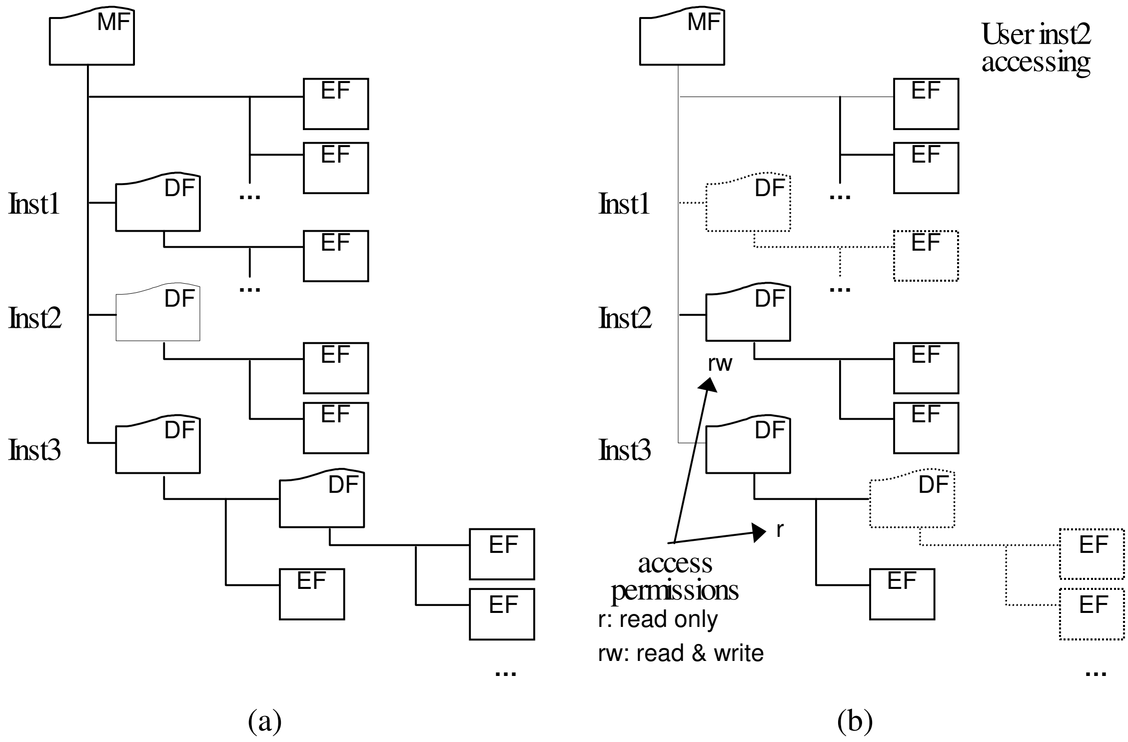


Fig. 7. The smart card file system: (a) its entire content and (b) the partial view offered to one of the institutions issuing the card.

Each owner provides the views for the guests to be accessed by specifying their definition. The view definition is stored as an elementary file in a DF. The view is not materialised due to the limited memory size, also considering that Flash-EEPROMs are characterised by low read access times, and hence the dynamic view materialization can achieve satisfactory performance. As a result, when an authorised guest accesses the view (selects an EF) he/she actually accesses the information on the data matching the stored conditions, which are then dynamically retrieved and sent to the requesting application. Owner data will then be organised at the physical level by creating a DF and one or more elementary files for storing the entire table data; the owner has full access to these data. The views defined for the guests are stored in separate EFs in a different DF, so that the identification of access privileges are managed in terms of directory/file accesses. In the customised owner-multiguest scenario, user groups are used to manage directory accesses leading to a complex directories' tree for specifying the views for different users (Fig. 8a). In the generalised owner-guest scenario, instead, there is only one directory for storing the views since no further distinction is required (Fig. 8b).

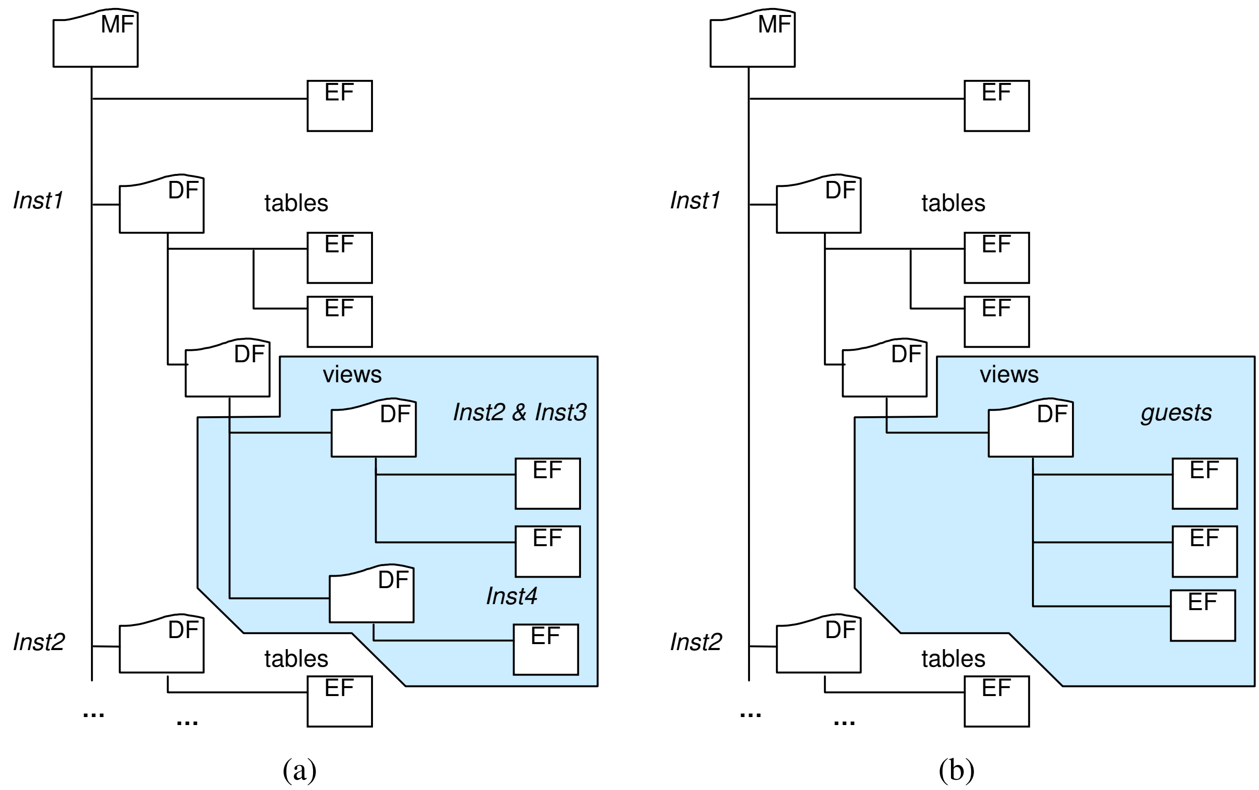


Fig. 8. Implementation of tables and views in the (a) owner-multiguest scenario and in the (b) generalised owner-guest scenario.

We also envisage the adoption of a mechanism for providing different system tables and consequently different underlying file systems, by storing several SCQL dictionaries reflecting the user/privilege schemas designed at a logical level. Following a successful authentication, an SCQL dictionary is selected and loaded for providing the desired scenario according to the present user. To simplify data accessibility and privileges, without incurring in data redundancy on the card (especially considering the limited available memory, besides performance and maintenance issues), the physical organisation of the database data in directories and files reflects the generic guest and multiguest scenarios, so that privileges are directly applied to directories and files rather than to a subset of a table records. This is consistent with the traditional management of Smart Card data; in fact, according to a de facto convention, all files containing useful data for a particular application (the EFs for that application) are always grouped together in a single DF. This produces a clear and easily understood structure, and extensible in that it is easy to enter a new application into a Smart Card by creating the appropriate DF.

The relationship between an owner (and guest) in either one of the two discussed scenarios, and the associated SCQL dictionary, is maintained by means of a table with a sorted organisation given the expected limited number of records in the table and the fact that data will be stored at production time and accessed only in retrieval (as discussed in [3]). This approach has an additional advantage with respect to security: each time a memory access to the application occurs, once the selected SCQL dictionary is loaded, the operating system can check whether it lies outside the boundaries of the memory allocated to the relevant application DFs. If so, it may prohibit the access. This differentiation of system tables to define different file systems on the basis of the active user allows achieving a transparent profile management for the higher-level applications that can then access the portion of the database.

A stable and static profile definition is required to guarantee security and cope with the technological constraints of the limited memory size and its technological characteristics. When new users need to be added or privileges modified, a maintenance session is required, and data can be modified and re-sorted off-line, so that the Flash-EEPROM life endurance degradation is limited.

7 Conclusion and future work

We presented a methodology for Smart Card database design using Flash-EEPROM storage. Unlike traditional database design, this process is technology driven since a very careful design of the logical and physical data structures is needed to meet the technological constraints the smart card architecture introduces, and to provide satisfactory performance.

Our analysis identifies two possible scenarios related to privacy requirements with respect to a multifunction environment on a Smart Card database, both requiring the definition of owner-guest user profiles and access permissions. We examined a logical-physical model for guaranteeing a protected access to the database information which provides a customised view of the Smart Card system tables (and consequently file system) in order to hide the part of data that cannot be accessed by a user.

The provided mechanism is well suited for the Smart Card architecture since it acts at the operating system level, providing a differentiated and personalized view of the physical data to the file system level and to the upper application levels.

Further research will focus on ad hoc architectural enhancements of the smart card with particular attention to the processor, to the possibility of implementing different storage technologies on the same card in order to benefit from their peculiarities, and to the analysis of other policies for data management to improve performance.

Acknowledgements

We thankfully acknowledge the clarifying discussions and the helpful suggestions of our colleagues Letizia Tanca and Fabio Salice

References

[1] P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, *Database systems*, McGraw-Hill, 2000.

[2] C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, "PicoDBMS: Scaling down Database Techniques for Smart card," Proc. 26th Int. Conf. on Very Large Databases (VLDB), 2000, pp.11-20.

[3] C. Bolchini, F. Salice, F. Schreiber, L. Tanca, "Logical and Physical Design issues for Smart Card Databases", Technical Report no. 2001.68, Politecnico di Milano, submitted for publication.

[4] A. T. S. Chan, J. Cao, H. Chan, G. Young, "A Web-enabled Framework for Smart Cards Application in Health Services," Communications of the ACM, vol. 44, no. 9, pp. 77-82, 2001.

[5] S. Castano, M. Fugini, G. Martella, P. Samarati, *Database Security*, Addison-Wesley, 1994.

[6] S. Ceri, G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.

[7] R. Clarke, *Privacy Issues in Smart Card Applications in the Retail Financial Sector*, www.anu.edu.au/people/Roger.Clarke/DV/ACFF.html, 1997.

[8] DataQuest, *Chip Card Market and Technology Charge Ahead*, MSAM-WW-DP-9808, 1998.

[9] R. Elmasri, S. H. Navathe, *Fundamental of Database Systems*, 2nd edition, Benjamin Cummings, Redwood City, 1994.

[10] M. Fugini et al., *Sicurezza dei sistemi informatici*, Apogeo, 2001.

[11] *Guidelines on the Protection of Privacy and Transborder Flows of Personal Data* – www.oecd.org/dsti/sti/it/secur/prod/PRIV-EN.htm, 1980.

[12] Microsoft Corporation, Windows for Smart cards Toolkit for Visual Basic 6.0, http://www.microsoft.com/windowsce/smart card/, 2000.

[13] Privacy Protection on Global Networks – www.oecd.org/dsti/sti/it/secur/act/privnote.htm, 1999.

[14] R. Ramakrishnan, J. Gehrke, *Database management systems 2nd ed.*, McGraw-Hill, 2000.

[15] W. Rankl, W. Ewffing, *Smart Card Handbook – Second Ed.*, Wiley, 1999.

[16] Smart card adoption for ID application in the Italian Government http://www.palazzochigi.it/fsi/ita/eEurope.htm#securizzato

[17] Sun Microsystems, *JavaCard 2.1 Application Programming Interface Specification, JavaSoft Documentation*, 1999.

[18] Ö. M. Tamer. P Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, 1991.

[19] Consumer Privacy and Smart Cards – A Challenge and an Opportunity, www.ftc.gov/reports/privacy3/comments/001-scf.htm, 1997.

[20] Smart ID card: Data Privacy Protection Measures, www.info.gov.hk/gia/general/200010/24/1024195.htm, 2000.

[21] Privacy Law to Pave Way for "Smart Card", http://insight.mcmaster.ca/org/efc/pages/media/spectator.21.feb98.html, 1998.

[22] W. Stallings, *Network Security Essentials – Applications and Standards*, Prentice-Hall, 2000.

[21] R. van de Riet et al., "A comparison of two architectures for implementing security and privacy in cyberspace," Security and Privacy in Cyberspace, Vrije Universiteit Amsterdam, Rapport nr. IR-471, no.6, March 2000.

[24] R. van de Riet, A. Junk, E. Gudes, "Security in Cyberspace: a Knowledge-base Approach," *Data and Knowledge Engineering*, Vol. 24, no. 1, 1997, pp. 69-98.

[25] R. van de Riet, J.F.M Burg, "Modeling Alter Egos in Cyberspace: Who is responsible?", Proc. WebNet96, San Francisco, AACE, USA, 1996, pp. 462-467.