CHAPTER

3

Chip Migration

Contents

Payment cards for debit and credit have proven to be a huge business success for the retail financial industry. The magnetic stripe technology is cheap enough to make the cost of cards small. Moreover, the payment network and the terminals at the point of service have been in place for years now. Therefore, there is no further need to invest in infrastructure. Meanwhile, the operation of debit and credit cards increases year after year, both in the number of issued cards and in the geographical coverage. Consequently, profit has increased, so there is no apparent cause for concern.

The first section of this chapter lists several causes of concern that should encourage payment system operators, issuers, and acquirers to consider the migration from magnetic stripe to chip. We believe that this motivation could help a chip solution vendor make his business case when talking to skeptics about switching from the magnetic stripe technology to chip.

The second section reminds the reader of the essentials of chip card technology. In this book the terms ICC and chip card are used interchangeably to refer to one and the same device, which not only stores data in its permanent memory but is also able to process data. Therefore, it would be more accurate to refer to these cards as microprocessor chip cards, to clearly distinguish them from the memory chip cards, which can store data but cannot process it. For the fluency of presentation, however, we will refer to the microprocessor chip cards as simply chip cards or ICCs. In Section 3.2.1 we give an overview of the hardware and software structure of a chip card, as well as the life cycle of the chip card. We then make a diagonal

presentation of the ISO/IEC 7816 standard, which dominates the world of ICC with contacts. The emphasis is only on few topics from Part 4 of the ISO/IEC 7816 standard. We briefly review the basics about a card file system and the methods of referencing files (Section 3.2.2). Then, the formats of the commands/responses sent to and returned from the card, as well as the most common commands, are briefly presented (Section 3.2.3). This is the minimum amount of knowledge someone would need to be able to understand the rest of this book. For an extensive introduction to chip card technology, the reader should refer to [1]. At the end of the section, we present the concepts of terminal application and card application, and their interactions in a client server model when performing a transaction (Section 3.2.4).

After these foundations of the ICC technology are revisited, two possible chip migration paths are outlined: closed proprietary payment applications and open interoperable payment applications. We analyze some of the features of a payment application that allows interoperability (Section 3.4). These features contrast with the homologue features of a proprietary application (identified in Section 3.3), and thus emphasize the price one pays for open design and interoperability.

3.1 A business case for chip migration

A cause of concern against keeping in place the magnetic stripe technology is the increase of abuses in magnetic stripe payment cards reported worldwide. Attackers have great insight about the design details of these cards, which helps them to identify security weaknesses that could lead to fraud.

In face-to-face payment transactions, counterfeiting the magnetic stripe has become a dangerous threat [2, 3]. This threat combined with sophisticated methods of monitoring the cardholder's PIN cause significant damages to financial institutions issuing such card products (see Section 2.6).

Card associations and payment system operators are concerned with decreasing the amount of fraud. In this context, the migration of actual payment card products from implementations using the magnetic stripe as a storage medium to a chip is seen as a necessary security improvement. The term "chip" designates the integrated circuit embedded in the plastic card. For the purpose of this book, we consider only those chips that offer protection against probing their resources. A chip providing this feature is referred to as tamper-resistant. The reduction of fraud becomes possible because of several factors:

- It is very hard to clone chip cards, particularly the secret cryptographic parameters they contain, unless the tamper resistance of the chip is overtaken. Even though more and more papers report methods of subverting the tamper resistance of chip cards, the attacks are far too complicated for common attackers to mount.
- Through its processing power, the chip card is actively involved in the risk management at the point of service. The chip card becomes a remote agent of the issuer that is able to correctly intervene in a local authorization process performed at a terminal that is not connected on-line to the payment network. The chip can enforce the proper policies for an optimal trade-off between the availability of the retail financial service provided to the cardholder and the security of the issuer against fraudulent transactions.
- The chip improves the process of determining counterfeit cards, through implementing the card authentication method with dynamic authentication mechanisms. It also provides greater protection of the cardholder against fraudulent transactions through the off-line verification of the PIN in the card, for transactions authorized off-line.

The cost of the chip migration is impressive. Integrated circuit cards are much more expensive to issue than magnetic stripe cards. This entails significant costs for the issuers. New terminals are needed at the point of service, which are equipped with integrated circuit card readers. This entails high costs for the acquirer. The host computers of issuers and acquirers as well as the payment network must be adapted for chip migration.

These economic factors have caused many financial institutions to question whether it is cheaper to continue to support the loss due to fraud or to change the whole infrastructure. This is mainly the case for financial institutions located in developed countries, where the existing payment infrastructure is huge. Moreover, their losses are kept reasonably low, considering that the majority of the transactions, if not all, are authorized on-line, which decreases the risk of fraudulent transactions. In developing countries with large territories, however, where the payment infrastructure is poor, the payment transaction is assessed off-line in the majority of situations. In these cases it makes sense to invest in a chip solution from the beginning, since the security protection is clearly better.

Card associations and/or payment system operators have adopted new operating rules for their chip card products, which has motivated issuers and acquirers to perform the chip migration. Thus, the policy of decreasing the interchange fees for acquirers that do not adapt their terminals to accept

chip cards can be a good reason for acquirers to implement the chip technology. At the same time, both issuers and acquirers could be encouraged to adopt the chip, through a right liability policy. This policy could stipulate that issuers and acquirers that have not accomplished the chip migration assume the entire risk in case of fraud when making a transaction with an acquirer/issuer that has performed the chip migration.

There is still another strong reason for chip migration. Instead of thinking in terms of reducing fraud, maybe it is better to think in terms of increasing revenue streams as a consequence of chip migration:

1. Because of better decision-making by the chip at the point of service, it is possible to improve authorization controls at a lower cost. This means that communication costs related to the on-line authorization of a transaction can be reduced in situations where the card risk management together with the terminal risk management decides that authorization can be granted locally. This improves the efficiency of debit/credit cards in a segment of payments, which were previously judged too small.

2. Since the chip has computation power, the payment card becomes "smart." Card applications can provide far more flexible financial services and better answer the rapid changes in the retail financial market. The same chip can accommodate several card applications, which provides the multiapplication dimension of the chip cards. This allows issuers to reduce the investment cost per card application and better combine several payment instruments that satisfy different payment behaviors. For example, the same chip card can accommodate a national debit scheme used for domestic payments, an international credit scheme suitable for relatively important payments made while travelling abroad, and a cross-border electronic purse for paying per byte for information on demand bought from Internet providers. Thus, the flexibility of customizing the financial service provided to each cardholder on an individual basis further strengthens the relationship between the cardholder and his or her bank.

3.2 An overview of the chip card technology

This section presents a quick overview of the chip card technology. The functionality of an ICC is based on the standard ISO/IEC 7816, "Identification

cards—Integrated circuit(s) card with contacts" [4–7]. First, we look at the hardware and software structure of a chip card. Second, we review the basics about a card file system and the methods of referencing files. Third, the format of the command/response pairs sent to and returned from the card as well as the most common commands are briefly presented. Finally, the concepts of terminal application, card application, and their interaction in a client server model are presented.

3.2.1 Hardware and software structure of chip cards

The chip card is a plastic card that incorporates an integrated circuit, which is a single-chip computer. This computer contains a microprocessor that can access read-only memory (ROM), electrically erasable programmable memory (EEPROM), and RAM. The memory management unit (MMU) controls the access to these memories. The hardware structure of the single-chip computer is shown in Figure 3.1.

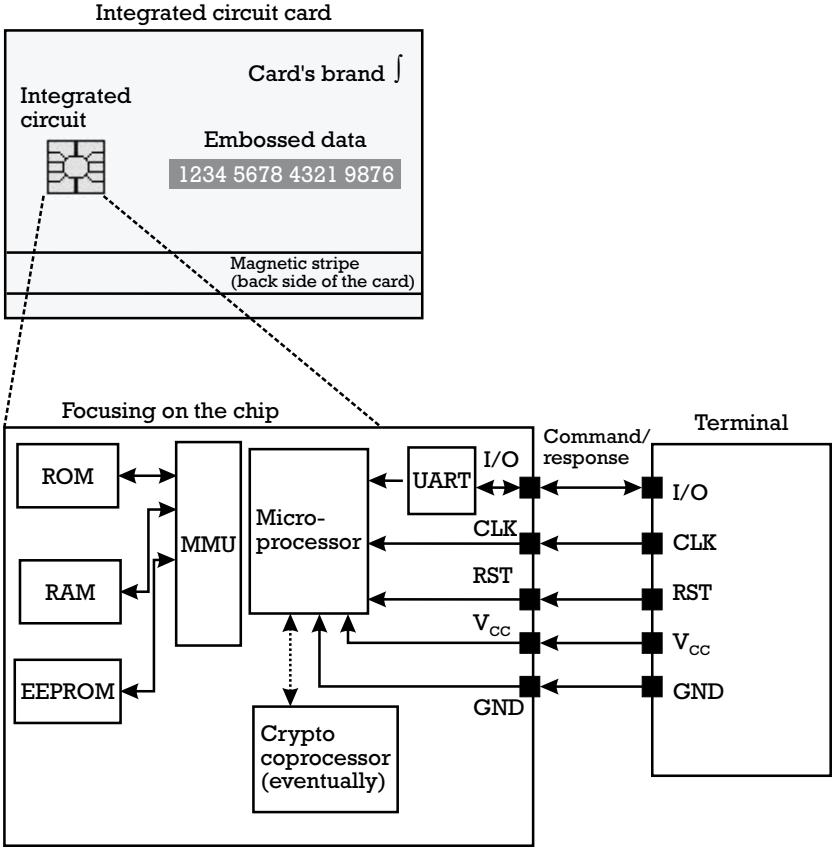

Figure 3.1 Hardware structure of the single-chip computer in the card.

The ROM is masked in the chip and cannot be changed during the whole lifetime of the chip card. The EEPROM permanently stores data that can be read but also modified during the lifetime of the card. The RAM is volatile memory that keeps data needed for the processing performed by the chip's microprocessor during one card session.

The chip is connected to the outside world through five contacts, which are assigned as follows:

- *I/O:* The chip has only one input/output serial line for communicating with the outside world. The universal asynchronous receiver transmitter (UART) serializes both commands coming from and responses going to the terminal. Among the protocols that can be implemented at the transmission level between the chip card and the terminal, we mention only two protocols known as T = 0 and T = 1 [4], since they are the only transmission protocols accepted by the EMV™ cards.
- V_{CC} and *GND:* The electrical power for the chip is provided by the terminal on these two contacts.
- *CLK:* The execution of all the processing in the chip is synchronized with a clock that is received from the terminal.
- *RST:* This contact receives the electrical reset signal from the terminal, which brings the chip to an initial status (see Section 3.2.4).

The single-chip computer is a slave depending on the terminal, which can be regarded as a master. The chip does not take initiative, but is simply driven by the terminal. Figure 3.2 shows two possible software architectures of a chip card, which can be regarded as a pile of software packages.

The left side of Figure 3.2 shows a proprietary software organization that does not allow for the portability of card applications. It can be seen that the card application makes direct calls to either a proprietary application programming interface (API) or to the card's operating system. Since each ICC producer has its own operating system and its own proprietary API, the card application is not portable from one chip card to another. Every time an issuer changes the chip card producer, the card application has to be rewritten. Regarding the mapping of hardware resources to the software architecture, it must be noted that a large diversity of possibilities exists, depending on the memory capacity allocated for ROM and EEPROM. The operating system and proprietary API are masked in the ROM, whereas the card

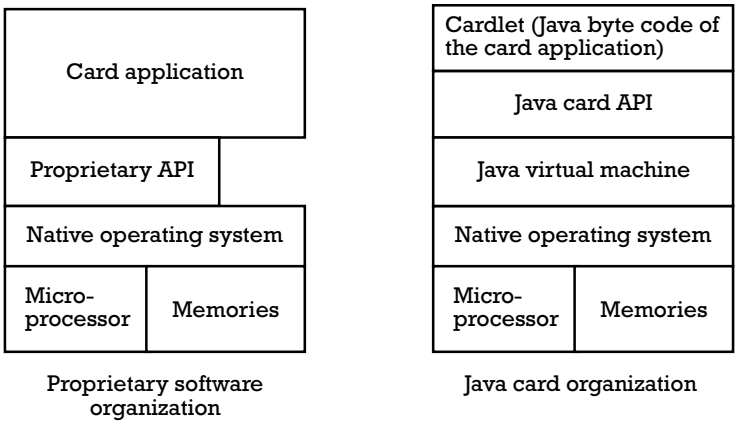

Figure 3.2 Two software architectures for chip cards.

application can reside either in the ROM or in the EEPROM. In the majority of the proprietary card implementations, however, the card application physically resides in the ROM and is logically integrated in the operating system instead of being on top of the operating system. The card file system that contains the data structures needed during the processing performed by the card application is always kept in the EEPROM, since both read and write operations must be available on the permanently stored data.

In the right side of Figure 3.2 the software organization of a Java card [8] is presented. In this software architecture the code of the card application is isolated from specific hardware and operating system libraries through the Java virtual machine (JVM). The JVM interprets the byte code corresponding to the Java source of the card application and translates it into instructions that are executable by the hardware and native operating system. Each chip hardware platform has its own JVM, which allows the card application to be independent of the hardware and the native operating system of the card. One of the big benefits of this platform, which can justify its higher price, is the reduced time to get new applications to the market. They also support the downloading of "cardlets," which is the term sometimes used for the applets downloaded to a chip card, even when the card is already in its utilization stage. Last but not least, the applications written for one chip card can be ported to other chip cards, provided they have the same Java card API, which is actually standardized as Java Card 2.1.1 [9]. Thus, this software organization guarantees the interoperability of card applications written for different chip card platforms.

The actual competitor of the Java card is the MULTOS operating system for chip cards, whose specifications are created by the MAOSCO consortium [10]. Card applications are coded in the MULTOS executable language (MEL), which is an interpreted language that is hardware-independent.

Therefore, similar to the Java card, the MULTOS architecture bases its functionality on a MEL interpreter, which can be regarded as a virtual machine, and an application loader. Generally there is a distinction between off-card and on-card virtual machines. In contrast to the Java card, the MULTOS virtual machine is completely realized on-card. This allows implementing firewalls between the applications, which provides a suitable security level for multiapplication environments. The application loader ensures the possibility of secure loading and deletion of card applications to and from the EEPROM, even during the utilization life stage of the card.

It is important to note that the software configuration and the file system loaded in the card are dependent on the life stage of the chip card. Table 3.1 presents each life stage of the card, along with the most important operations performed by a certain role in that stage.

3.2.2 Card file system and file referencing

The operating system of the chip card manages a file system that stores the data needed by each card application. ISO/IEC 7816-4 [5] supports two categories of files: dedicated files (DFs) and elementary files (EFs). They are organized in a hierarchical tree, with DF as branches and EF as leaves. A typical organization of the card's file system is schematized in Figure 3.3.

3.2.2.1 Master file and dedicated files

The highest DF in the hierarchy, which is the root of the tree, is also called the master file (MF), which is the only mandatory DF in the file organization. In the example presented in Figure 3.3, the MF contains one leaf, the elementary file EF1, and two branches, the dedicated files DF1 and DF2. Data that is used for all the applications in the card (e.g., administrative and general security information such as the ICC serial number, access control keys, card's general PIN, as well as data concerning the management of the card's life cycle) are stored in elementary files at the MF level. This information can be used by the operating system for creating another DF at the MF level.

The dedicated file DF1 contains four leaves. The first three of them (EF11, EF12, and EF13) are working EFs, while EF14 is an internal EF. We will later see the difference between working and internal elementary files. The semantic of the information in DF1 and its elementary files will be explained in Section 3.3. The dedicated file DF2 contains only two leaves, which are the working elementary files EF21 and EF22. Each dedicated file can further contain other hierarchically inferior dedicated files. In Figure 3.3, DF2 contains one subdedicated file DF21.

Table 3.1
The Life Cycle of a Chip Card

Life Stage	Operation	Role Involved
IC fabrication	The integrated chip (IC) is produced, with the operating system in the ROM mask. For a proprietary card, this mask can contain the card application. For a Java card, the ROM contains the Java virtual machine. A unique ICC serial number is assigned to each chip.	IC manufacturer
ICC fabrication	The integrated chip is embedded in the plastic card.	Card manufacturer
Prepersonalization	The file system of the card is created. The data that is specific to the payment system and is common to all chip cards participating in the same scheme is also written during this stage.	Card manufacturer
	For a proprietary card, if the card application is resident in the EEPROM, the application software is loaded. For the Java card all the card applications that are foreseen in the standard configuration of the card are loaded.	
Personalization	The data specific to each cardholder is filled in the appropriate files of the card.	Card issuer
Utilization	The card is operated according to the business goals of each application.	Cardholder
	For Java cards, card applications can be dynamically added in the EEPROM during the utilization stage. The only restriction is that the corresponding byte code originates from an application provider agreed upon by the card issuer and there is enough EEPROM space. Card applications can be also dynamically deleted from the EEPROM, according to the preferences of the cardholder.	
End life	When the validity of the card expires, the card is disaffected by the card issuer, which can for example block the entire card.	Card issuer

A dedicated file can be seen as a container of data belonging to one card application. Several data elements of the card application that are semantically related are stored in the same elementary file. Application control information and cardholder's financial data are stored in the elementary

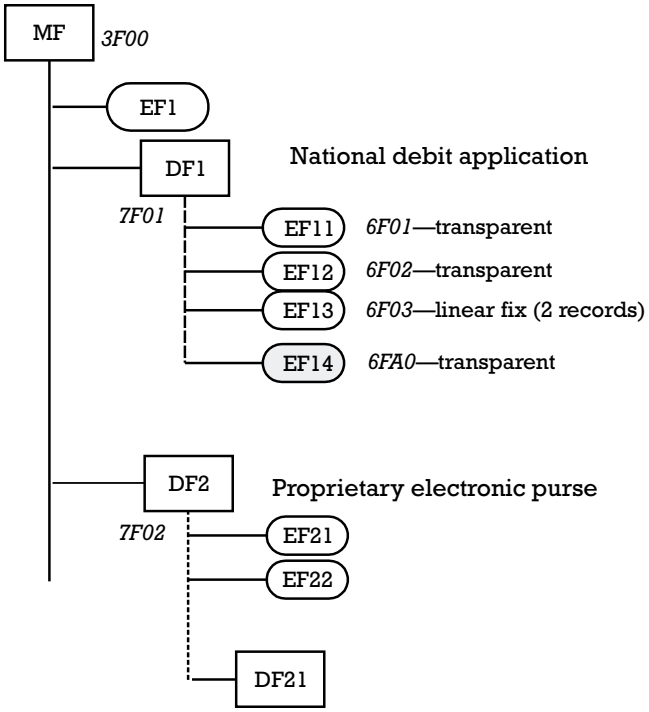

Figure 3.3
The organization of the card's file system.

files encompassed in the same DF. Each DF may contain cryptographic keys for implementing various security services, and each may have its own application PIN, which can be used to refine the access control mechanism of a multiapplication card.

The referencing of a DF in the card's file system, which corresponds to the possibility of selecting a card application from the terminal's side, can be performed in two distinct ways:

1. *Referencing with a fixed file identifier (FID), which consists of 2 bytes (4 hexadecimal digits).* For example, the MF always has the FID equal to 3F00, while DF1 has the FID equal to 7F01 and DF2 has the FID equal to 7F02, and so on. In order to be able to select a card application with its FID, the terminal application must know beforehand the file organization in that card. For example, in order to select DF21 starting from the MF level as the current directory, the terminal must first select DF2 with its FID, and only after this selection is successful can it select DF21 with its corresponding FID.

2. *Referencing with an application identifier (AID), which consists of up to 16 bytes.* The encoding of the AID is detailed in the ISO/IEC 7816-5 [6].

The AID comprises either the registered application provider identifier (RID), which optionally is concatenated with the proprietary application identifier extension (PIX), or the proprietary application identifier. The referencing of card applications with registered application provider identifiers has the advantage that the terminal does not have to know in advance the FID of the DF that stores the application or its position in the card's file system. Moreover, since the RID is unique worldwide, several applications can be stored in the card with no danger of referencing conflicts.

It will become obvious in Section 3.3 that referencing a DF with its FID is suitable for closed design proprietary card applications. In Section 3.4 we show that the open design interoperable card application uses DF referencing through the AID.

3.2.2.2 EFs

The data elements of a card application are encoded in elementary files. The elementary files of a card application can be further subdivided into working EF and internal EF:

- A working EF stores data that is not interpreted by the card application, but rather used by the terminal application exclusively during the execution of a protocol with the card.
- An internal EF stores data managed only by the card application for management and control purposes. Cryptographic parameters used for security services provided by the card as well as the cardholder's witness PIN or other cardholder verification codes (CHVs) are stored in internal EFs.

Two referencing methods for elementary files are used:

1. *Referencing with an FID, which consists of 2 bytes (4 hexadecimal digits).* The same FID referencing mechanism as that described for the DF can be also used for the EF. The disadvantage of this referencing mode is that before a file management command can be applied on an EF, the terminal must explicitly select this EF inside the DF corresponding to the card application. Examples of file management operations are the reading of some bytes from a transparent EF or the writing of a record in a linear fixed EF. Another disadvantage is that the terminal must know beforehand the FID of all the elementary

files inside the card application's DF. The advantage of this referencing mode, however, is that the selection of either a DF or EF in the card is done uniformly, which simplifies the card's implementation. This referencing mode of EF is suitable for closed design proprietary card applications (see Section 3.3).

2. *Referencing with a Short File Identifier (SFI), which consists of a number between 1 and 30 that can be encoded on 5 bits.* This referencing method has the advantage that the SFI can be used as a file handler, which can be given as an input parameter to a file management command. This means that there is no need of executing an explicit selection of an EF inside the DF before calling a read/write command from/to an EF. Moreover, the SFI of all the working EFs in the card's application DF can be easily listed in a kind of DF table of contents. This helps the terminal learn by itself the publicly available working EF(s) existing in a DF. Therefore, this referencing method of EFs is preferred in open design interoperable card applications (see Section 3.4).

The structure of an EF depends on its intended use. As explained in ISO/IEC 7816-4 [5] one can distinguish among four basic types of EF structures. Transparent files consist of a sequence of bytes. A linear fixed file consists of a number of records, all having the same length. A linear variable file consists of a number of records, each with a variable length. The cyclic files contain records of fixed length organized in a ring structure. After all the records are written, the oldest entry in the file will be overwritten by the current entry to be stored. Figure 3.4 schematizes the four types of file structures.

The file header of each EF stores information about the type of EF file structure and the size of the file. It also stores the possible actions to be performed on the file (read, write, invalidate, rehabilitate, increase) as well as the access conditions under which a terminal application can perform that action (card's general PIN or application PIN, authentication with a symmetric key, access always permitted or access never permitted).

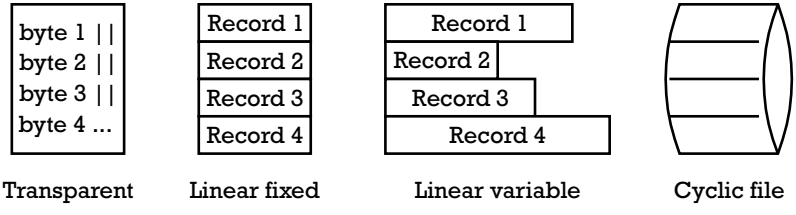

Figure 3.4 Four types of elementary file data structures.

3.2.3 Command and response format

In accordance with the OSI 7-layer model, the information transaction exchanged between the card and the terminal can be divided into three protocol sections:

1. The physical layer protocol (layer 1) corresponds to the electrical signals on the I/O contact of the card.

2. Data transmission protocols (layer 2) correspond to T = 0 and T = 1 protocols [4]. They are both asynchronous, half-duplex protocols. T = 0 is a byte-oriented transmission protocol of the first-generation chip cards when the computing power and the RAM on the chip was limited. It does not allow the transmission of data both in the command and in the response. T = 1 is a block-oriented protocol, which better respects the OSI reference model and allows transmission of data both in the command and in the response. The data is handled in blocks and the error checking is carried out on an entire block of data rather than on 1 byte.

3. Application protocols for command and response data (layer 7). A step in an application protocol consists of sending a command application protocol data unit (C-APDU) from the terminal application to the card application. The latter processes it and sends back the response application protocol data unit (R-APDU) to the terminal application. A schematized picture of a C-APDU/R-APDU pair is given in Figure 3.5.

The C-APDU consists of a mandatory header of 4 bytes and an optional body of a variable length. The header includes the class of instructions to which the command belongs (CLA), the instruction code (INS) determining the command inside of a class, and the parameters of the instruction

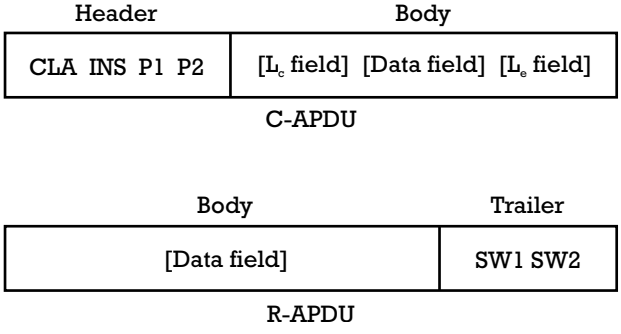

Figure 3.5 Command/response pair (C-APDU/R-APDU).

[parameter 1 (P1) and parameter 2 (P2)]. The meaning of these parameters is dependent on the instruction code. The body of the command is optional and may contain the following fields:

- L_c: This field (of 1 or 3 bytes) can contain the number of bytes present in the data field of the command.
- *Data field:* This field (of L_c bytes) contains the string to be sent as input data to the card application.
- L_e: This field (of a variable length up to 3 bytes) can contain the maximum number of bytes expected in the data field of the response returned by the card.

The R-APDU contains a conditional body of variable length L_r that can be less than or equal to L_e. The R-APDU includes the trailer, which is a mandatory field of 2 bytes containing the status words (SW1, SW2). The status words inform the terminal application about the result of executing the command in the card application.

The ISO/IEC 7816-4 [5] standard defines only the basic commands. They can be grouped in file selection, read data, modify/delete data, generate data, compare data, and authenticate through cryptographic functions. In addition to these standardized commands, each card operating system or each card application defines private use commands. For example, the EMV™ debit/credit card application defines its own commands beside those in ISO/7816-4 (e.g., the `GET PROCESSING OPTIONS`, and `GENERATE AC`). The commands for the creation and personalization of files in the card's file system and the commands for blocking either an application or the entire card are further examples of private use commands. One can understand why even if chip card operating systems have been implemented according to the ISO/IEC 7816 standard, it does not necessarily mean that they are compatible with each other [11].

3.2.4 Card application and terminal application

The terminal at the point of service is a card acceptor device (CAD) equipped with a chip card reader, which is often referred to as the Interface Device (IFD). The terminal interacts with an ICC according to the client server model.

- A client application runs in the terminal. This client application is referred to as the terminal application.

- A server application runs in the ICC. This server application is referred to as the card application.

The terminal application sends commands as a client to the card application, which responds as a server.

To easily explain the processing performed by both the terminal application and the card application, as well as their interaction in the client server model, it is convenient to represent them as algorithmic state machines (ASM). For an ASM the next state depends on the actual state and the event that triggers the transition from one state to another. The state of the ICC consists of a set of data elements and cryptographic parameters. The cryptographic parameters are organized in the card's file system. From this point of view the ICC can be seen as a permanent storage medium. Compared to a magnetic stripe card, which was a passive storage medium, the ICC has computational power provided by its own microprocessor. The event that triggers the transition of the ICC from one state to another is a command with parameters received from the terminal. As a consequence of this transition the ICC performs an action. First, the action consists of updating the value of the data elements and cryptographic parameters stored in the ICC (i.e., the state of the ICC), according to the requirements of the command and the accompanying parameters. Second, the action computes a response that is returned to the terminal. The response can contain the value of one or more data elements stored or computed in the card and a status word, which describes whether or not the command was successfully completed in the card. In case of failure, the status word indicates the source of error. The response received from the card represents the event that triggers the transition of the terminal from one state to another.

The terminal brings the ICC from an initial state S_0 to an operational state S_1 through an electrical reset. The action performed by the card following the reset is to prepare and send back to the terminal an answer to reset (ATR) response, which contains enough information to allow the communication subsystem of the terminal to synchronize with the communication subsystem of the card. Once this initial handshake is performed, the terminal can send C-APDU to and receive R-APDU from the card. The client-server relationship established between the terminal and the card is presented in Figure 3.6.

The set of commands and responses exchanged between the terminal and the ICC in the framework of a transaction is called a transaction profile.

A card application in the ICC contains a set of data elements that can be accessed by the terminal after a successful selection of the application. A data element is the smallest information unit that can be identified by a

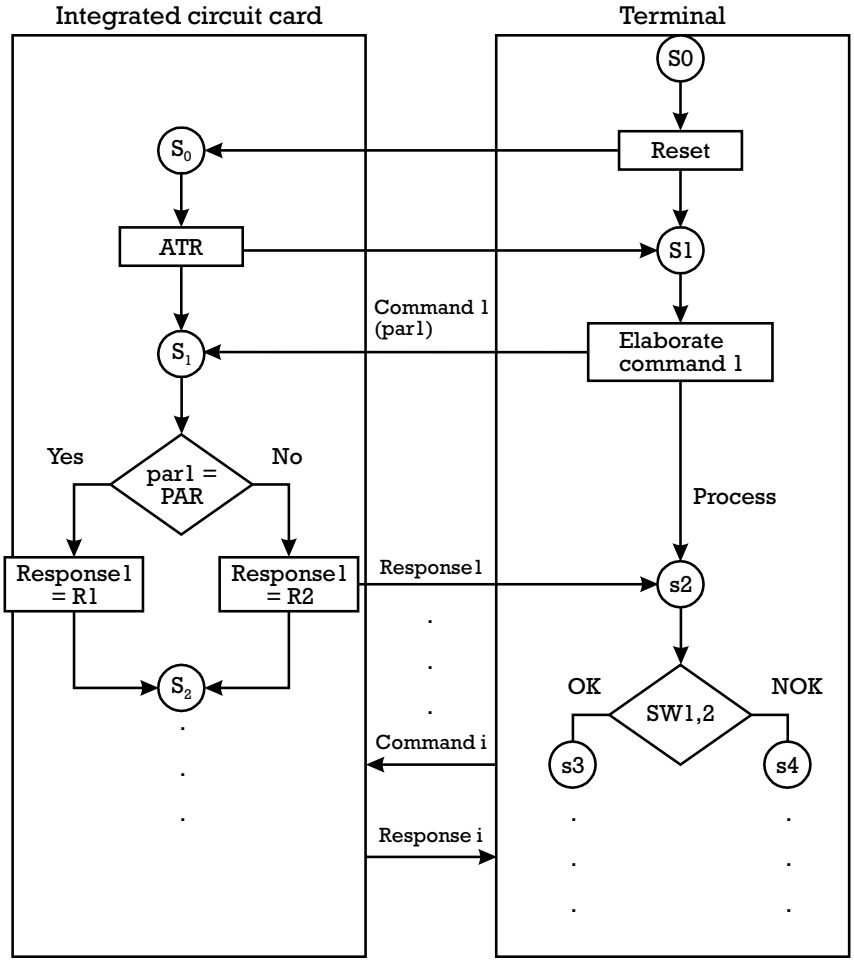

Figure 3.6 Terminal application and card application in a client server configuration.

name, a description of its logical content, and a format. Data elements are mapped onto data objects, which are encoded according to a certain format (e.g., fixed length format, BER-TLV format, and others).

The terminal application consists of the sequence of commands, which are launched by the terminal to trigger the transition of the card application from one state to another. This determines the processing in the card according to the functionality of the card application. The terminal application also processes the data objects received in the responses from the card and the status words reported at the end of each command.

Several issues are identified in relation to the design of a card and terminal application, indifferent to whether it is a proprietary or interoperable solution:

1. One has to define the encoding of data elements into data objects in the card and the terminal application.
2. The organization of these data objects in a file system stored in the card and the referencing of files in this system must be defined. If several card applications reside in the same card, it is necessary to specify the separation of files corresponding to each application.
3. The set of commands supported by the card and the possible responses elaborated towards the terminal must be also defined.
4. The underlying cryptographic technology used for implementing the necessary security protections in both the card and the terminal must be chosen.

The possible solutions to these issues are restricted to the framework provided by the ISO/IEC 7816. The next two sections show how the aforementioned issues are solved in the case of a proprietary and closed payment application and in the case of an interoperable and open payment application, respectively.

3.3 Proprietary payment application

The approach described in this section outlines an oversimplified proprietary design solution, which can be adopted by payment system operators migrating from magnetic stripe cards to integrated circuit cards. The purpose is to show the shortcoming of this approach, in case open design and interoperability are business requirements.

Let us assume that a payment system operator provides a proprietary payment application, which consists of both a card and a terminal application. It is intended for the purpose of a dedicated business goal—for example, a national debit scheme for POS payments. The proprietary application is completely controlled by the payment system operator, who has designed and specified it according to its business requirements.

The card application is instantiated in chip cards of cardholders who are clients of an issuer. The issuer has established a business agreement for implementing the card application, which is provided by the payment system operator. The issuer has no freedom to customize the card application to its specific business needs. An ICC carrying the proprietary card application is accepted with terminals managed by an acquirer that has also established a business relationship with the payment system operator. The

acquirer agrees to implement the terminal application, which is provided by the payment system operator. The acquirer has no freedom in customizing the terminal application specified by the payment system operator.

Let us assume that the card application stores the financial information characterizing the cardholder. This information consists of the following data elements:

- A*pplication Preferred Name:* This is the name associated with the application running in the card. This name is printed on the display of the POS terminal for informing the cardholder about the application that is currently selected in the card.
- *Application Version Number:* This is the version number of the software implementation of the card application.
- Application Expiration Date: This data element represents the date after which the card application expires.
- A*pplication PAN:* This is the information that uniquely identifies the account of the cardholder and the issuer that keeps this account.
- *Cardholder Name:* This represents the name of the cardholder to be printed on the sale slip produced at the point of service.
- *Issuer's operator, first number:* This is a telephone number displayed on the man-machine interface of the shopkeeper if the processing at the point of sale performed by the terminal determines that a voice referral is necessary.
- *Issuer's operator, second number:* This is a second telephone number the shopkeeper can call for the voice referral in case the first number is congested.

When the terminal sends an `INTERNAL AUTHENTICATE` C-APDU, with a body containing a random number and data elements characterizing the business environment at the point of service (amount, terminal ID, date, and time), the card computes a dynamic authenticator on this data. This authenticator is computed with a MAC-based dynamic data authentication (DDA) mechanism, like that presented in Appendix D, Section D.7.1. The card sends an R-APDU, which contains the dynamic authenticator in its body, back to the terminal. More details about the computation by the card of the dynamic authenticator and its verification by the terminal are provided in Section 3.3.4.

3.3.1 Encoding data elements with a fixed format

A convenient and simple method of encoding the data elements can be obtained with a predefined fixed format, where each data element is mapped into a data object consisting of a fixed number of bytes. This number represents the maximal length of the object. If the representation of data is smaller than the maximal length, then data is justified right or left in that field and the remainder of the field is padded accordingly. For the data elements listed above, an example of their encoding is given in Table 3.2.

The definition of data elements can be proprietary to the payment system operator, but it can also be a subset of the interindustry data elements defined in ISO/IEC 7816-6 [7].

The data objects are not explicitly identified in the card application, but they are identified implicitly. This implicit identification is obtained through their location in one elementary file or another of the card's file system, and through their position in that file. This mapping of data objects into the card's file system is totally at the discretion of the payment system operator that decides which data object goes to which elementary file, and in which relative position of that file. A possible mapping is shown in Figure 3.7.

Table 3.2
Encoding with Fixed Format of Data Elements

Name	Format	Length
Application Preferred Name	an 16 (alphanumeric on maximum 16 characters)	16 bytes—maximum. Right justified, left padded with blanks
Application Version Number	b (binary)	2 bytes
Application Expiration Date	n6 (YYMMDD) (numeric on 6 digits, in the order: year, month, day)	3 bytes
Application PAN	cn 19 (numeric on 19 digits)	10 bytes—maximum Right justified, left padded with zeros
Cardholder Name	ans 26	26 bytes—maximum Right justified, left padded with blanks
Issuer's operator, phone number	n16 (numeric on 16 digits)	8 bytes
MAC-based dynamic authenticator	b	8 bytes
Application Transaction Counter	b	2 bytes

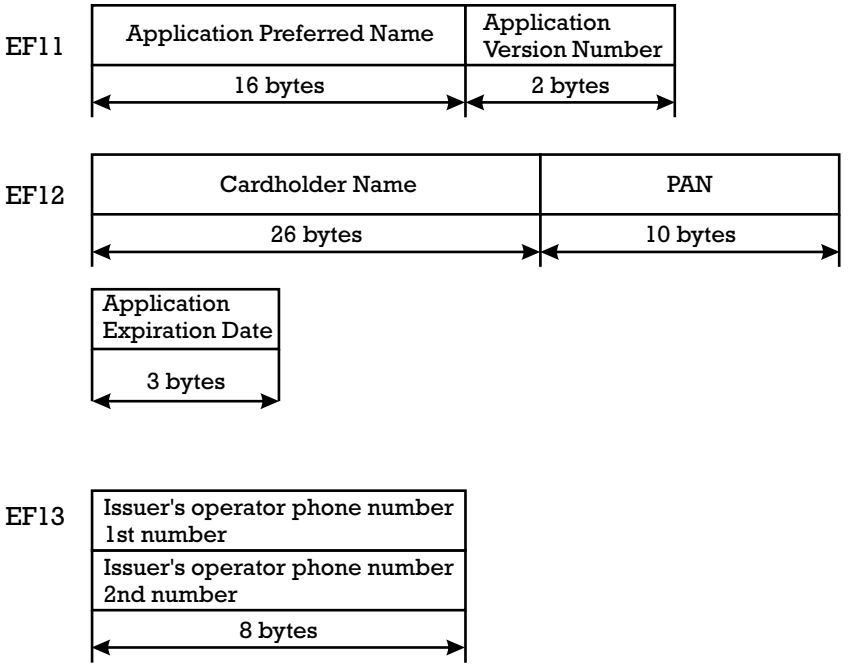

Figure 3.7 Mapping of data objects into the card's file system.

In this example, all the data objects present in the card application are mapped in three elementary files as follows:

1. EF11, which is a transparent file, stores the Application Preferred Name and the Application Version Number in this order. It has a total of 18 bytes, of which the first 16 bytes store the Application Preferred Name and the last 2 bytes store the Application Version Number.

2. EF12, which is also a transparent file, stores the following data objects: the Cardholder Name in the first 26 bytes, the application PAN in the next 10 bytes, and the Application Expiration Date in the last 3 bytes.

3. EF13, which is a linear fixed file, contains two records of the same length. They store the first phone number and the second phone number of the issuer's operator. These are phone numbers where the POS operator can call the issuer if any suspicions appear about the current transaction or cardholder.

3.3.2 Fixed file system organization

The file system of the card hosting the proprietary debit application is outlined in Figure 3.3.

After resetting the card, the current referenced DF is the MF, which represents the default entry in the card file system. The MF in this example has one single EF as a leaf (EF1). This elementary file keeps the ICC serial number, which is a data element that is uniquely assigned by each card manufacturer. There are two DFs that are branches of the MF. Each DF is an entry point to another card application. For example, DF1 is the entry point for the national debit card application, while DF2 is the entry point of a dedicated electronic purse scheme. Note that the payment system operator providing the first application is not necessarily the same as the payment system operator providing the second application.

The DF1 contains four leaves. Three of them are the working elementary files EF11, EF12, and EF13 presented above. The fourth leaf is an internal elementary file EF14, which contains a symmetric key for computing the dynamic authenticator. This key, which is denoted K_d and is unique for each card, is derived from the issuer master key (IMK). The IMK is managed by the issuer for the computation of dynamic authenticators. The key K_d is obtained with the formula $K_d = F_1(IMK)[PAN]$, according to the principles explained in Appendix E, Section E.5. The diversification information *Diversification_Info* consists of the PAN assigned to the cardholder. F_1 is a one-way function, like a MAC based on a 64-bit length block cipher (see Appendix E, Section E.4). The issuer computes the key K_d and writes its value in the EF14 of the DF1 during its personalization stage. The terminal application in the POS, as well as any other agent except the card application itself, have no access to the content of EF14, which should remain secret during the whole lifetime of the card.

The terminal application uniformly references the DF1 and the elementary files EF11, EF12, and EF13, using their FID on 2 bytes. In Figure 3.3 the file identifiers are listed next to each file in the system.

3.3.3 Preestablished command and response formats

In a proprietary payment scheme, the terminal application is aware of the encoding of data elements into data objects, the mapping of data objects into elementary files, and the organization of the dedicated/elementary files in the card. All these design details are fixed beforehand by the payment system operator and are implemented in the same form by all the participants in the system.

Therefore, the format of the commands and responses is fixed. The set of data objects that is transmitted within the body of each C-APDU is always the same. The set of data objects that is returned in the body of each R-APDU is also preestablished.

Moreover, the transaction profile is fixed, since the sequence of commands in the terminal application is predetermined and is not negotiable between the card and terminal. The steps below describe this transaction profile:

- Step 1: The terminal application selects DF1, which contains the debit application.
- Step 2: The terminal selects EF11 and reads its binary content.
- Step 3: It repeats the same sequence of commands for EF12.
- Step 4: It selects the linear fixed file EF13 and reads its two records.
- Step 5: The terminal prepares a message *M1* containing a random number *R*, and some data about the business environment. This business environment data includes the amount of the transaction (which is typed in the terminal's keypad by the POS operator), the identifier of the terminal *TerminalID*, and the time/date when the transaction took place *TimeDate*. The message *M1* is the body of the `INTERNAL AUTHENTICATE` C-APDU, which is sent to the card. This C-APDU triggers the computation of the MAC-based dynamic authenticator in the card, the value of which is denoted *mac_card*. A more detailed look at the computation performed by the card is postponed to Section 3.3.4. The value *mac_card* together with the Application Transaction Counter (ATC) is returned in the body of the R-APDU. The R-APDU body is always 10 bytes, where the first 8 bytes contain the value of *mac_card* and the last 2 bytes contain the ATC.

In each step of the transaction profile described above, the terminal sends a set of commands and processes the received responses. The processing performed by the terminal on these responses can be described as follows:

- After reading the content of the elementary files EF11, EF12, and EF13, the terminal application identifies the data elements of the card application according to their predetermined position in an elementary file. Thus, the first 16 bytes of EF11 are identified as being the

Application Preferred Name, the next 2 bytes are the Application Version Number, and so on.

- The terminal displays the Application Preferred Name to inform the cardholder about the card application that is currently effective. The terminal application performs some checks. For example, the Application Version Number in the card must be equal to the Application Version Number of the terminal application. The Application Expiration Date read from the card must be smaller than the current date in the terminal (card not expired). If all these verifications are passed, the terminal continues processing; otherwise the card session is aborted.

- The terminal creates a message *M0* containing the cardholder's financial information stored in the card. The Cardholder Name, the PAN, and the Application Expiration Date of the card are concatenated in *M0*.

- If the transaction amount is less than a threshold limit imposed as a security parameter by the acquirer, the transaction is processed off-line, without the intervention of the IH. Thus, the transaction is accepted if the value of the MAC-based dynamic authenticator produced by the card (*mac_card*) is correct. More details of this assessment process is postponed until Section 3.3.4. The validity of the dynamic authenticator proves the authenticity of the card and the fact that the card is not counterfeit, which obviously is a big step forward compared to magnetic stripe cards.

- When the transaction amount is above a threshold limit, the terminal sends on-line to the IH the financial data captured from the card (*M0*), the data characterizing the business environment of the POS terminal (*M1*), the ATC, and the dynamic authenticator computed by the card *mac_card*. The IH checks whether the dynamic authenticator is valid or not in the same way that this checking is performed off-line by the terminal. The IH, however, can perform supplementary verifications compared to the off-line case, which increase the security of the authorization process. Thus, the issuer can verify whether the balance of the account indicated by the PAN has enough funds for supporting the transaction. The issuer can also verify whether the card was blacklisted, for reasons of being reported stolen, or having compromised keys, etc. If all these verifications are passed, the issuer informs the POS terminal about the outcome of the authorization, approving or denying the transaction.

3.3.4 Symmetric cryptographic technology

The security protection in the transaction profile described in Section 3.3.3 is deliberately oversimplified. It serves solely for the presentation of concepts related to the choice of an appropriate cryptographic technology for either proprietary or interoperable design solutions.

The only security service foreseen in this transaction profile is card authentication. The MAC-based dynamic data authentication, as explained in Appendix D, Section D.7.1, is the security mechanism implementing this security service.

Since the scheme is proprietary, the payment system operator can easily coordinate the whole key management process for both issuers and acquirers in the framework of symmetric cryptographic techniques.

- Using a secure key distribution channel established in advance, each issuer receives an IMK. The payment system operator derives IMK from its master key (MK). The issuer identifier serves as the diversification information *Diversification_Info* (i.e., $IMK = F_2(MK)[Issuer\ Identifier]$) (see Appendix E, Section E.5). As it was explained in Section 3.3.2, during the card personalization stage, the issuer uses IMK to produce the key K_a, which is a symmetric key for computing the dynamic authenticator.

- The payment system operator provides acquirers with a security application module (SAM) that stores the MK. The SAM is a tamper-resistant chip, which is not embedded in a plastic card but rather is directly plugged into a specialized connector inside the terminal. Note that since this chip contains the MK, its tamper resistance is an essential assumption for the security of the payment system operator.

In the remainder of the section we concentrate only on step 5 of the transaction profile described in Section 3.3.3. We zoom in on both the processing performed by the card to produce the dynamic authenticator *mac_card* as well as on its verification by the terminal, with the help of its SAM, in case the authorization is granted off-line.

Figure 3.8 outlines the computation of the dynamic authenticator by the card application.

The terminal prepares the C-APDU with a header (CLA, INS, P1, and P2) corresponding to the internal authenticate command. The body of the C-APDU contains the message $M1 = R \| amount \| TerminalID \| TimeDate$. *M1* is constructed as the concatenation from left to right of the random number *R*,

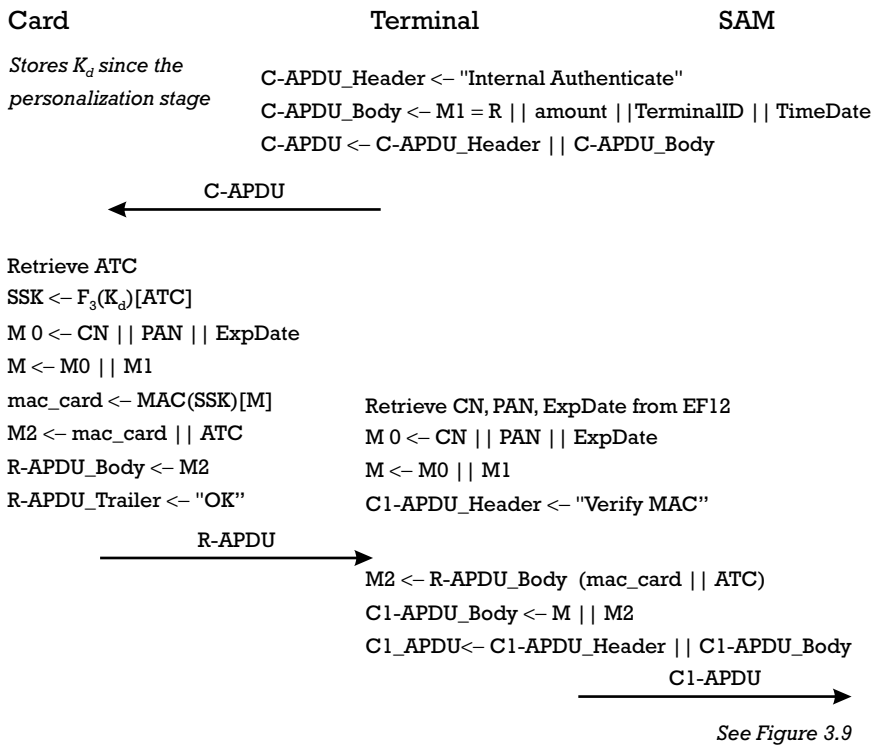

Figure 3.8 Computation of the dynamic authenticator.

the amount of the transaction *amount*, the identifier of the terminal *TerminalID*, and the time/date when the transaction took place *TimeDate*.

After receiving this C-APDU, the card application performs the following processing:

- Retrieve the current ATC and use it as a diversifier for obtaining the session key SSK from the card's unique key K_d (i.e., $SSK = F_3(K_d)[ATC]$).
- Compute a message $M0 = CN \| PAN \| ExpDate$. This message is the concatenation from left to right of the Cardholder Name (CN), the PAN, and the Application Expiration Date of the card.
- Retrieve the message *M1* from the C-APDU body and construct the message *M* as the concatenation of *M0* and *M1* (i.e., $M = M0 \| M1$).
- Compute the dynamic authenticator as *mac_card* = MAC (SSK) [*M*].
- Compute the R-APDU body as the $M2 = mac\_card \| ATC$. Return R-APDU.

Figure 3.9 outlines the verification of the dynamic authenticator by the terminal application with the support of the SAM.

After receiving the R-APDU, the terminal can verify off-line the correctness of the dynamic authenticator *mac_card* received from the card, using the SAM. In this case the SAM can be regarded as the issuer's remote agent validating the dynamic authenticator. To this end the terminal constructs *M0* in the same way as the card did, using the data elements CN, the *PAN*, and the Application Expiration Date previously read from the card. The terminal computes the message *M* concatenating *M0* and *M1*. The terminal

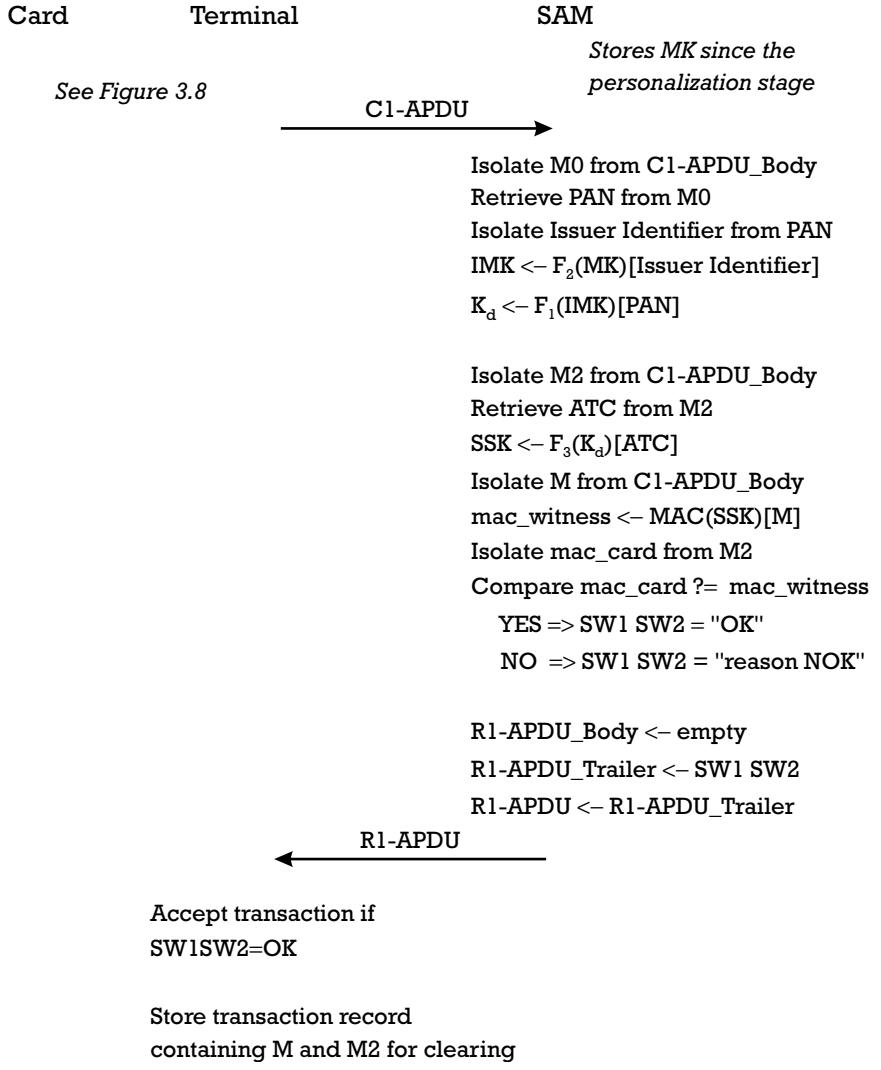

Figure 3.9 Verification of the dynamic authenticator.

prepares another C1-APDU, this time addressed to the SAM. Its header (CLA, INS, P1, and P2) corresponds to the *Verify MAC* command supported by the SAM. The body of this C1-APDU contains the message $M = M0 \parallel M1$ concatenated with the message $M2 = mac\_card \parallel ATC$.

After receiving the C1-APDU, the SAM performs the verification of the dynamic authenticator *mac_card*, following the steps listed below:

- Retrieve the *PAN* from *M0* and isolate the issuer identifier. Use it as a diversifier to obtain the IMK as $IMK = F_2(MK)[Issuer\ Identifier]$, where MK was stored in the SAM since its personalization.
- Using the PAN as a diversifier, derive the unique key of the card K_d, used for the computation of the dynamic authenticator, from the IMK (i.e., $K_d = F_1(IMK)[PAN]$).
- Retrieve the ATC from *M2* and use it as a diversifier for deriving the session key SSK from the unique key of the card K_d (i.e., $SSK = F_3(K_d)[ATC]$).
- Compute the dynamic authenticator as *mac_witness* = MAC (SSK) [M].
- Retrieve the dynamic authenticator *mac_card* computed by the card from *M2* and compare it with the recomputed value *mac_witness*.
- If the two values are equal, position the SW1 and SW2 status words in the trailer of the R1-APDU as OK. Otherwise, position them as NOK. Return R1-APDU.

After receiving the outcome of the dynamic authenticator verification in R1-APDU, the terminal decides whether to approve (SW1SW2="OK") or deny (SW1SW2="NOK") the transaction. The terminal keeps a transaction record (M, M2) in its permanent memory. The record will be sent to the acquirer for the clearing process.

If the terminal decides that the authorization is performed on-line by the IH, the authorization request message (1100) will transport $M = M0 \parallel M1$ concatenated with the message $M2 = mac\_card \parallel ATC$. After receiving these messages, the security module of the IH will perform the same processing for verifying the dynamic authenticator as the processing described for the SAM.

As one can see, in the case of a proprietary payment application, which can authorize off-line transactions involving small amounts, symmetric key cryptographic techniques are appropriate for implementing security mechanisms. In this case the payment system operator controls the whole key

management for both issuers and acquirers, which allows an easy and cost-effective operation of symmetric key cryptographic algorithms. The immediate consequence is that the card does not need to implement asymmetric cryptographic algorithms, and therefore, a cryptographic coprocessor for long arithmetic computation is not needed in its hardware architecture (see Appendix D, Section D.1.2). This keeps the price of chip cards low. The use of the SAM in the structure of the terminal allows the off-line verification of the MAC-based dynamic authenticator. The SAM increases the cost of the terminal, which is the price to pay for off-line authorization of transactions involving small amounts. If the payment system operator decides that all the authorizations must be performed on-line, indifferent of the transaction amount, the presence of the SAM in the terminal is no longer needed. In this case, the verification of the dynamic authenticator is directly performed by the issuer, which simplifies the design of the terminal and its cost.

We have argued that the use of symmetric key cryptography is rather cheap for securing proprietary payment schemes, at least from the point of view of issuers. This does not mean, however, that public key cryptographic techniques are ruled out for securing the off-line authorization of transactions in proprietary payment schemes. With the advance of chip technology, it can be foreseen that the emphasis of security computations will shift towards public key enabled chips, which will render unnecessary the presence of a SAM in the hardware structure of a terminal.

3.4 Interoperable payment application

The design principles explained in the previous section are not suitable for interoperability. The following business case for an interoperable payment application is now analyzed.

The proprietary card application described in Section 3.3 is referred to as *C1*. The card hosting *C1* is issued by the issuer *I1* and is accepted at a terminal managed by the acquirer *A1*, running the terminal application *T1*. The whole payment scheme is managed by the payment system operator denoted *O1*.

Assume that a cardholder has an ICC storing a card application *C2*, providing the same functionality as *C1*. However, the issuer *I2* that manages the ICC containing *C2* is not a subscriber of the payment system operator *O1*. The issuer *I2* is a subscriber of the payment system operator *O2*, which did not establish any business agreement with *O1*. The payment system operator *O2* made its own design for the card application *C2* and for the terminal application *T2*. This basically means that:

- The rules of encoding the data elements into data objects adopted in *C2* could be different than the rules adopted in *C1*.
- It could be that the rules of encoding data elements into data objects adopted by both *C1* and *C2* are the same—for example, according to ISO/IEC 7816-6 [7]. There is a high probability, however, that the mapping of data objects in elementary files is different from one card application to the other, since there is no standard that regulates this matter. Then the implicit identification of data objects in the two card applications is different.
- The file organization in *C2* is different than the file organization in *C1*, since the file tree structure and the file identifiers adopted by the file organization in *C2* are probably different than in the file organization adopted by *C1*.
- Both payment applications use interindustry commands as defined in ISO/IEC 7816-4 [5]. Because of the differences, however, in mapping data objects in files and in the file organization, the set of data objects to be transmitted with each command and the set of data objects expected to be received with each response are different from one card application to another. This determines two different transaction profiles for the two payment applications, which finally means two distinct terminal applications *T1* and *T2*.
- It is also possible that the formulas for computing the dynamic authenticator differ from one card application to another, while the cryptographic keys involved in this computation are certainly proprietary to each payment system operator.

In case the acquirer *A1* would like to broaden its financial services to cardholders of the issuer *I2*, then *A1* should establish a separate business relationship with the payment system operator *O2*. Following this agreement, the acquirer *A1* loads in its terminals another distinct terminal application *T2*, which is proprietary to *O2*. Moreover, considering that symmetric cryptographic technology is used, the terminal should be able to accommodate in addition to the security application module *SAM1* used by *T1*, a supplementary security application module *SAM2*, which is exclusively used by *T2*. Another possibility would be to cumulate the security functions and the corresponding cryptographic parameters of both *SAM1* and *SAM2* into one single SAM, with the condition that the payment system operators *O1* and *O2* have established a business relationship in this sense. In practice, this

alternative is almost ruled out both by concurrency reasons between operators and for reasons determined by logistic problems related to key management and personalization of the SAM. As more and more system operators propose proprietary payment applications to acquirers, the management of the terminal applications and SAM(s) would become very difficult and the terminal more and more expensive.

A possible solution for interoperability would be that payment system operators create a consortium that specifies coproprietary card and terminal applications, with closed design solutions. As a result, everyone interested in being interoperable with this closed system would adhere to a memorandum of understanding proposed by the initial consortium. This policy, however, is not appropriate for the world of banking and financial services. Payment system operators, issuers, and acquirers would like to independently decide how the payment application would best match their interests.

A consortium comprised of Europay, MasterCard, and Visa (which is referred to as the EMV™ consortium) proposed an interoperable and open solution. In the framework of their solution each payment system operator, issuer, and acquirer can still customize a card/terminal application to its own business needs, providing they respect the basic negotiation mechanisms proposed by the EMV™ specifications. The rest of this section explains the principles of how this can be achieved.

3.4.1 Self-determined encoding of data elements

Instead of adopting a predefined fixed format for encoding data elements into data objects and an implicit identification of these data objects, the solution adopted by the EMV™ is to explicitly identify each data object. This is achieved with a tag, which can be regarded as a unique identification label. The data object has also attached the information about the length of the data element it encodes, such that there is no need of specifying beforehand a fixed length for each data element supported by an application. Thus, a data element is encoded following the tag-length-value (TLV) convention, described in the Basic Encoding Rules (BER) contained in the ISO/IEC 8825 standard [12]. Only the value field of the EMV™ data object actually conveys the useful information, while the tag and length fields convey the identification information. The BER-TLV encoding of the data elements in the card application *C1* is listed in Table 3.3.

The BER-TLV representation is suitable from the point of view of interoperability. Since every EMV™ data element is completely characterized by the tag and the length fields, the EMV™ terminal application can identify each data element and retrieve the conveyed information in the

Table 3.3
BER-TLV Encoding of Data Elements

Name	Format	Tag	Length
Application Preferred Name	an 1-16	9F12	1–16
Application Version Number	b	9F08	2
Application Expiration Date	n6 (YYMMDD)	5F24	3
Application PAN	cn 19 var. up to 19	5A	Var. up to 10
Cardholder Name	ans 2-26	5F20	2–26
Issuer's operator, phone number	n16	To be defined	8
MAC-based dynamic authenticator (called Application Cryptogram in EMV™)	b	9F26	8
Signed Static Application Data	b	93	The length of the RSA modulus of the issuer (for further details on RSA, see Appendix F)
Signed Dynamic Application Data	b	9F4B	The length of the RSA modulus of the card

value field, indifferent of their "position" in the card (which EF and on which position). Therefore, the EMV™ terminal application has no need to know in advance the structure of the EMV™ files in the ICC to retrieve financial data needed for the completion of a payment transaction. As it will become evident in the next paragraph, it is sufficient that the terminal application knows the references of all the publicly available elementary files of the card application and the indexes of all the retrievable records from these files. The terminal application, however, has no need of previous information about how data is organized in these records. This allows complete freedom for the issuer of the EMV™ cards about the modality of mapping data objects into elementary files. No business relationship has to be established in advance between the issuer of the ICC and the acquirer responsible for the terminal, except that they are members of the same payment system operator/card association. The mapping of data objects in application elementary files (AEF) is illustrated in Figure 3.10.

An elementary file can be regarded as a sack where the data objects can be located in any position. Once the elementary file is read in the terminal,

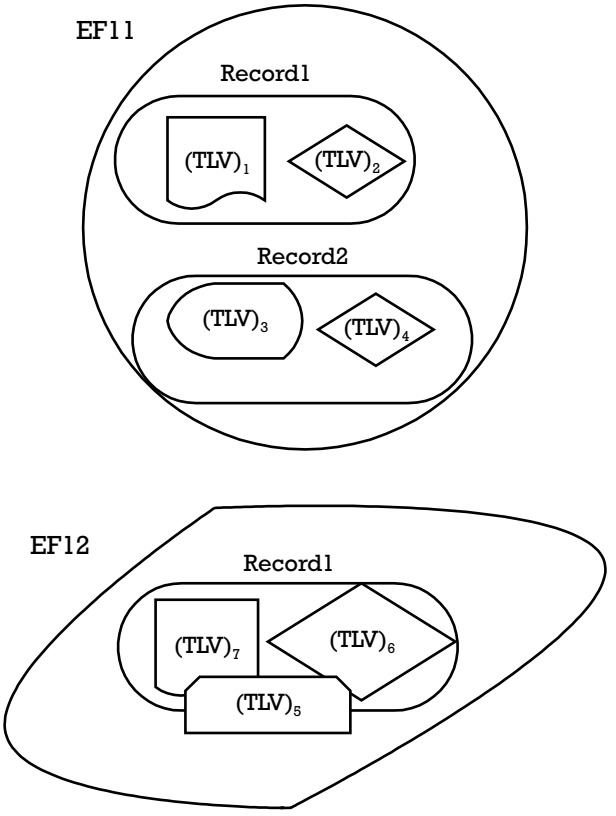

Figure 3.10 EMV™ mapping of data objects in elementary files.

the sack is emptied. The terminal recognizes data objects in the heap according to their tags and not according to their relative position in the elementary file from which they were downloaded.

The price paid for interoperability is a lower efficiency of the BER-TLV encoding. Every data element needs more bytes for its representation because of the addition of the tag and length fields besides the actual information conveyed in the value field of the data object.

3.4.2 Customized file system organization

The file system of an EMV™ card is also compatible with ISO/IEC 7816-4 [5]. A possible EMV™ file system is presented in Figure 3.11.

The file system is divided into application definition files (ADF) and directory definition files (DDF), allowing several card applications to be simultaneously accommodated in the card. A separate ADF corresponds to each card application present in the card. An ADF is referenced with an AID (see Section 3.2.2.1). Each ADF encompasses one or more AEF(s). An AEF

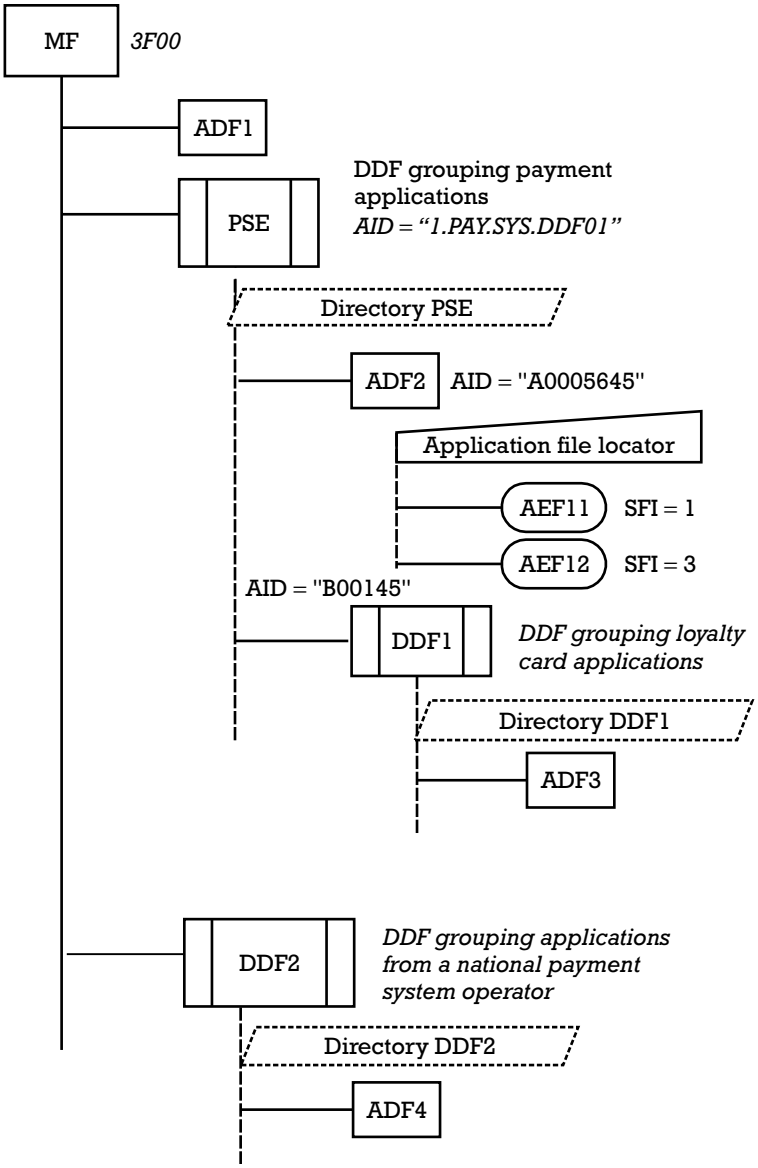

Figure 3.11
File system in an EMV™ card.

is a linear variable file containing the public information of the card application available to the counterpart terminal application. From this perspective an ADF can be regarded as an application data container. Inside the ADF each AEF is referenced with a SFI, which is a number in the range 1 to 30 (see Section 3.2.2.2). The SFI can be used as the handler of the AEF, once the ADF container is selected and known to the terminal. This handler can be directly used by the card commands performing file operations.

A DDF encompasses a group of related ADF(s). Each DDF in the card's file system is also referenced with an AID.

- The nature of the applications can be the criteria for grouping the ADF(s) in a DDF. For example, all the payment card applications in an EMV™ card could be gathered in the same DDF. The DDF that groups them is called the payment system environment (PSE) and has a special AID represented by the string 1PAY.SYS.DDF01, which is a reserved AID.

- The ADF(s) can be also grouped according to the payment system operator that proposed them. In the example of Figure 3.11, DDF2 gathers all the payment card applications of a national payment system operator.

A DDF can also include other hierarchical inferior DDF(s). For example, the PSE can further contain a DDF dedicated to loyalty card applications, denoted DDF1 in Figure 3.11. The DDF can be seen as a container of card applications.

The organization of files in an EMV™ card is more flexible, such that the EMV™ terminal is not compelled to know this organization in advance in order to perform a transaction profile. The terminal has to be aware only about the AID of the DDF applications containers in the card and of the ADF data containers that are not included under a DDF. Thus, the acquirer has to set up in the terminal a list of all the acceptable applications (ADFs) or of all the acceptable groups of applications (DDFs).

- Once the selection of a DDF applications container is performed, the terminal can find the table of contents of the DDF. This table of contents is organized in a directory file, containing as entry points the AID of all ADF data containers and the AID of all the other hierarchical inferior DDF applications containers. For example, the directory file at the level of the PSE contains the AID of ADF2 and DDF1, and the directory file at the level of DDF1 contains the AID of ADF3. By reading a directory file, the terminal is able to learn the file organization in that DDF.

- Whenever the terminal has selected an ADF container, it is further able to read another table of contents, which this time lists the AEF(s) that are publicly readable from a card application. This table of contents is referred to as an Application File Locator (AFL).

3.4.3 Variable formats for commands and responses

In an EMV™ setup the card application and the terminal application can be designed by different roles, within the limits established by the EMV™ specifications. The roles do not have to agree in advance on the list of meaningful data objects to be transmitted from the terminal application to the card application within a command. These data objects are needed by the card to perform its processing. This means that the set of data objects to be transmitted within a command can be different from one card application to another.

Therefore, the card must instruct the terminal about the data objects acceptable to be transmitted in a command. To this end the card application sends to the terminal application a data object list (DOL). This contains the list of all the tag-length identifiers of the data objects to be included by the terminal application in a command body.

For each command accepting a variable data input, the EMV™ specifications have defined a separate type of DOL, which is transmitted to the terminal application before the invocation of the command. The list of items (TL)1, (TL)2, (TL)3, … included in each DOL type contains compulsory data objects specified by the EMV™ specifications and also chosen data objects of each issuer. The DOL(s) are personalized in the card application by the issuer before the card is operated during the utilization life stage.

The terminal uses the tag-length identifiers (TL) of the data objects in the DOL to retrieve the corresponding objects from its application heap. The data objects in the heap correspond to the current business environment: *amount*, *TerminalID*, *Time/Date*, and so on. The terminal retains the field value of the data objects identified in the DOL and concatenates them in a byte string, which is given as a data input to the corresponding command. The mechanism is depicted in Figure 3.12.

Moreover, the transaction profile is also variable, since the sequence of commands depends on the capabilities of the card concerning the implementation of some basic security mechanisms. Included among these mechanisms are the card authentication method (CAM), the cardholder verification method (CVM), and the decision as to whether the terminal performs risk analysis or if everything must be judged on-line by the issuer.

The Application Interchange Profile (AIP) is the data element stored in the card since the personalization, which instructs the terminal concerning the acceptable sequence of commands from the card's viewpoint.

3.4.4 Asymmetric cryptographic support

In the beginning of Section 3.4 we saw that implementing the off-line card authentication service using symmetric cryptographic techniques requires

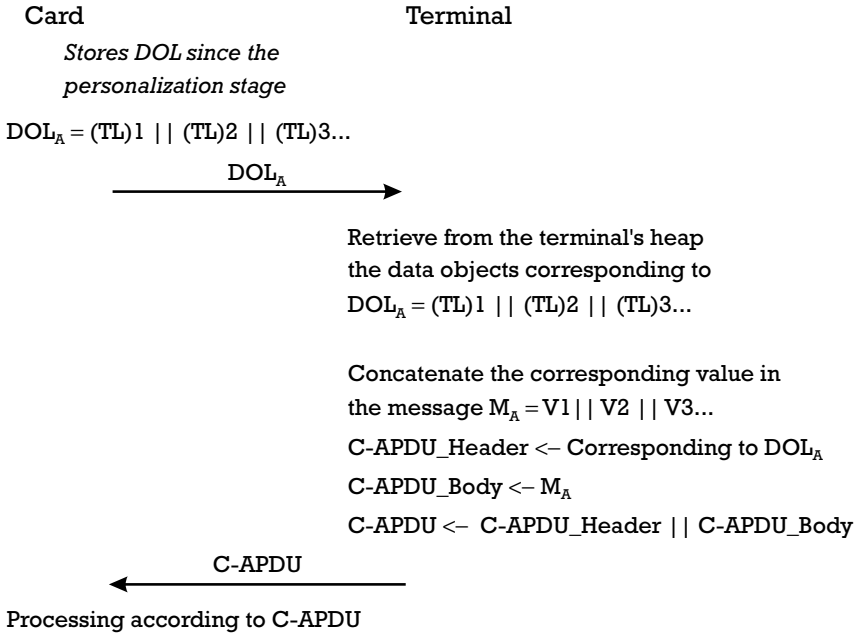

Figure 3.12 Variable command data input with DOL mechanism.

each payment system operator to provide the acquirer with a dedicated SAM. This impacts negatively on the complexity of the terminal and of the key management process.

Openness of design and interoperability imply the use of asymmetric cryptographic techniques for implementing the off-line card authentication service. Thus, in order to prove the authenticity of the financial data personalized in the card, as well as the fact that the card is genuine, instead of using the MAC-based DDA mechanism, one has to use the digital signature–based DDA mechanism, as presented in Appendix D, Section D.7.2. In this case there is no need for the distribution of sensitive secret cryptographic parameters by the payment system operator, which is a considerable advantage. Correspondingly, the hardware structure of the terminal is simplified, as is the key management overhead. The chip card, however, must be able to produce a digital signature, which requires an RSA operation in the case of EMV™ chips. Therefore, the hardware structure of the chip includes a cryptographic coprocessor for speeding up the computations performed by the card (see Appendix D, Section D.1.2). Moreover, there is need for more EEPROM space in the chip card to keep the private key used for signature generation as well as of the corresponding public key with the accompanying issuer certificate to be forwarded to the terminal for signature verification. These extra facilities are expensive both in terms of computation power and permanent storage space. They significantly increase the cost of

the chip card supporting asymmetric cryptography when compared to the chip card supporting only symmetric cryptography. When considering also that the latter card is several times more expensive than the magnetic stripe card, we can see that the former card is around 10 times more expensive than a magnetic stripe card. One can also understand a card manager's reluctance in asking the issuer's administration board for a dollar amount with an added zero at the end (read $1,000,000 instead of $100,000) for paying for the chip migration. As one can see, in relative terms the effort of the issuer increases spectacularly.

The normal reaction of the issuer's administration board would be to cut it in half. In the given circumstances, the card manager remembers past experiences with magnetic stripe cards. In that case the card authentication service was implemented with a MAC-based static data authentication (SDA) mechanism (see Section 2.5.3 and Appendix D, Section D.6.1). The issuer computes the static authenticator and writes it on the magnetic track during the personalization stage. Since the static authenticator in this case is computed with symmetric cryptographic techniques, the same limitations on openness and interoperability would be encountered as explained in the beginning of Section 3.4. Consequently, the EMV™ proposes the cheap solution that mirrors somehow the security protection with static authenticator but in an interoperable way. The issuer can compute this time a static authenticator using the signature-based SDA mechanism (see Appendix D, Section D.6.2). In this case the chip card would compute nothing (no need for a coprocessor) but only store some more bytes corresponding to the signature-based static authenticator. However, the security is also drastically reduced if the EMV™ transaction is concluded off-line and no on-line support is demanded from the issuer. The static authenticator would prove the authenticity of the financial data personalized in the card but would provide no protection against counterfeit. There is the impression that cloning the public information of the chip is more difficult than cloning the magnetic stripe. Cloning this public information, however, is still feasible for a hacker appropriately equipped (more details on this attack in Section 7.7.4).

Thus, while spending $400,000 for chip cards supporting symmetric cryptography on top of the costs of magnetic stripe implementation, the issuer loses the benefit of high security against counterfeit in small value transactions concluded off-line. Moreover, the issuer will not be able to implement the asymmetric enciphered PIN cardholder verification method (see Appendix D, Section D.5.5). This method would improve the security of the cardholder's PIN at the point of service, which is a very sensitive asset. Finally, the issuer is not able to implement on a multiapplication chip card

other "heavy" cryptography card applications, like the interoperable electronic purse CEPS, electronic brokerage, and electronic administration applications for tax paying. Thus, it appears more and more that it is better for chip migration to be done with support of asymmetric cryptographic techniques.

It is also important to note that the payment system operator escaped from the burden of organizing symmetric key generation and distribution processes, but it must operate a public key infrastructure instead. This is an equally difficult task, if not even more difficult. The operator, however, is motivated by the same hope of being able to diversify its services towards its subscribers.

References

[1] Rankl, W., and W. Effing, *Smart Card Handbook*, Chichester, England: John Wiley and Sons, 1997.

[2] Chan, E, "Fraud, a Common Virus in Asia," *Cards Now*, March/April 2001.

[3] Stern, C., "Micro-Thief That 'Steals' Credit Cards," *Sunday Mirror Magazine*, January 28, 2001.

[4] ISO/IEC 7816-3, "Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 3: Electronic Signals and Transmission Protocols," 1997.

[5] ISO/IEC 7816-4, "Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 4: Interindustry Commands for Interchange," 1995.

[6] ISO/IEC 7816-5, "Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 5: Numbering System and Registration Procedure for Application Identifiers," 1994.

[7] ISO/IEC 7816-6, "Identification Cards—Integrated Circuit(s) Cards with Contacts—Part 4: Interindustry Data Elements," 1996.

[8] Hassler, V., et al., *Java Card for E-Payment Applications*, Norwood, MA: Artech House, 2002.

[9] Sun, Java Card 2.1.1, Platform specifications, http://java.sun.com/products/javacard.

[10] Maosco Ltd., *Multos Overview*, http://www.multos.com/multpres.ihtml.

[11] Vedder, K., and F. Weikmann, "Smart Cards—Requirements, Properties, and Applications," in B. Preneel and V. Rijmen (eds.), *State of the Art in Applied Cryptography*, Springer LNCS 1528, 1998, pp. 307–331.

[12] ISO/IEC 8825, "Information Technology—Open Systems Interconnection—Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)," 1990.