T.C.
BAHÇEŞEHİR UNIVERSITY

The Graduate School of Natural and Applied Sciences
Computer Engineering Graduate Program

SMARTCARD PERSONALIZATION WITH CRYPTO ALGORITHMS in EMV STANDARD

Master Thesis

Mehmet Murat TANDOĞAN

Assoc.Prof.Dr.Adem KARAHOCA
ISTANBUL, JUNE, 2010

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

Graduate School of Natural and Applied Sciences

Computer Engineering Graduate Program

Name of the thesis: **Smartcard Personalization with Crypto Algorithms in EMV Standard.**

Name/Last Name of the Student: Mehmet Murat TANDOĞAN

Date of Thesis Defense:

The thesis has been approved by the Institute of Graduate School in Sciences.

Asst. Prof. F. Tunç BOZBURA

Director

Signature

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Head of Department

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members

Assoc. Prof. Adem KARAHOCA (Supervisor)

Prof. Dr. Nizamettin AYDIN

Asst. Prof. Yalçın ÇEKİÇ

ACKNOWLEDGEMENTS

Firstly, I want to thank to Assoc.Prof.Dr.Adem KARAHOCA for their continuous encouragement.

I also want to thank my girlfriend, Sebla ÜNLÜDERE for her infinite moral supports.

Last but, not least, I wish to thank to my family for their love and unlimited support in every stage of my life.

JUNE, 2010
Mehmet Murat TANDOĞAN

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ÖZET

KRİPTO ALGORİTMALARI KULLANARAK, EMV STANDARTLARINA UYGUN MİKRO İŞLEMCİLİ AKILLI KART KİŞİSELLEŞTİRME

Mehmet Murat TANDOĞAN
Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Yüksek Lisans Programı

Tez Danışmanı
Doç.Dr.Adem KARAHOCA
İSTANBUL, Haziran, 2010

Akıllı kart, kredi kartı boyutunda, mikro işlemci içeren bir plastik karttır. Bu mikro işlemciler RFID (temassız) ve temaslı olarak ikiye ayrılmaktadırlar. Gömülü mikro işlemci sayesinde, akıllı kartlar, çok miktarda veriyi, yüksek güvenlik tedbirleri altında saklayabilirler. Yüksek hafıza ihtiyacını ve işlemci kapasitesini, bilgi güvenliği ile birleştiren akıllı kartlar "akıllı"dır, çünkü taşıdığı bilgiye erişimi sınırlandırırlar. Bu sınırlandırma işlemi, çeşitli kripto algoritmalarının yanısıra, kart üreticisinin işlemciye fabrikasyon olarak verdiği bir takım yetkiler ve erişim standartları ile doğru orantılıdır.

Akıllı kartlara; EMV standartlarına uyularak, kişi ile ilgili özel bilgilerin kripto algoritmaları kullanılarak yüklenmesine kart kişiselleştirme denir. Teknolojinin ilerlemesi ve akıllı kartların vageçilmez hale gelmesi ile birlikte, akıllı kartların güvenlik problemleri ortaya çıkmakta ve EMV standardının güvenlik önlemleri günden güne arttırılmaktadır. Akıllı kartlar güvenlik, kullanım kolaylığı gibi sağladıkları avantajlarla çipli, manyetik bantlı veya bantsız olarak günümüzde telekomünikasyon, bankacılık, toplu taşıma, sağlık gibi farklı sektörlerde müşteri kartı, kimlik kartı, telefon kartı, tanıtım kartı, müşteri kartı, promosyon gibi uygulamalar için yaygın olarak kullanılmaktadır.

Bu tezin amacı; kripto algoritmaları kullanarak, çalışır vaziyette örnek bir program yazılıp; çeşitli kripto algoritmaları ile (DES, 3DES) akıllı kartlara EMV standartlarına uygun olarak çeşitli bilgilerin yüklenmesi, karta erişimin test edilmesi ve akıllı kartların PIN numarasını kırma işleminin araştırılmasıdır. Projede 2 adet ACOS 2 işletim sistemli kontak akıllı kart ve kart okuyucusu ve yazılım dilleri olarak Microsoft .NET C#, Microsoft Visual Basic 6.0 kullanılmıştır.

ABSTRACT

SMART CARD PERSONALIZATION WITH CRYPTO ALGORITHMS IN EMV STANDARD

Mehmet Murat TANDOĞAN
Institute of Sciences, Computer Engineering Graduate Program

Supervisor
Assoc. Prof. Dr. Adem KARAHOCA
İSTANBUL, Haziran, 2010

Smart card is a credit card sized plastic card embodying a microprocessor. These microprocessors are dividen into two groups as RFID (contactless) and contact. Smart cards can keep a big amount of data under high security steps by the agency of embedded microprocessor. The cards those integrate high memory need and microprocessor capacity to information safety are smart cards, because they limit access to the information theya re carrying.

Loading the personal informations in accordance with EMV standards and by using crypto algorithms to the smart cards is called personalization. In conjunktion with improvement of the technology and becoming irrevocable of the smart cards, problems with the smart card security occures and security measures of EMV standards are being improved day by day. Nowadays smart cards are widely being used in telecommunication, banking, public transportation and health sectors by its advantages such as security and usage easyness as customer card, phone card, personalization card, advertising card and promotions.

The aim of this thesis is developing a sample program working by using crypto algorithms and loading of various informations according to the EMV standards, testing of card access and investigation of the process of breaking the PIN numbers of the smart cards. In this project, two contact smart cards with ACOS2 operating system, a card reader, Microsoft .NET, C# and Visual Basic 6.0 as software languages.

LIST OF SYMBOLS / ABBREVIATIONS

3DES	Triple DES (data encryption standard)
3GPP	3rd Generation Partnership Project
ADK	Additional decryption key
ADN	abbreviated dialling number
AES	Advanced Encryption Standard
AID	application identifier
API	application programming interface
ARM	Advanced RISC Machine
ARR	access rule reference
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
AT	attention
ATR	answer to reset
AUX1, AUX2	Auxiliary 1, Auxiliary 2
BAFA	Bundesamt f¨ur Wirtschaft und Ausfuhrkontrolle (German Federal Office of Economics and Export Control)
BCD	binary coded digit
BNA	Bundesnetzagentur (German Federal Network Agency)
BSI	Bundesamt f¨ur Sicherheit in der Informationstechnik (German Federal Office for Information Security)
CCS	cryptographic checksum
CDMA	code division multiple access
CEN	Comit´e Europ´een de Normalisation (European Committee for Standardization)
CHV	card holder verification information
CICC	contactless integrated chip card
CLA	class
CLK	clock
CPU	central processing unit
CRC	cyclic redundancy code
DES	Data Encryption Standard
DF	dedicated file
DO	data object
DPA	differential power analysis
DSA	digital signature algorithm
DSS	Digital Signature Standard
EC	elliptic curve crypto algorithm
ECC	elliptic curve cryptosystem
ECC	error correction code
ECDSA	elliptic curve digital signature algorithm (DSA)
EDC	error detection code
EEPROM	electrical erasable program read-only memory
EF	elementary file
EMV	Europay MasterCard Visa
ETSI	European Telecommunications Standards Institute

Etu	elementary time unit
GND	ground (electrical)
GNU	GNU is not UNIX
GPL	GNU General Public License
GUI	graphical user interface
HMAC	keyed-hash message authentication code (MAC)
HTML	hypertext markup language
I/O	input/output
IBE	identity-based encryption
ICAO	International Civil Aviation Organization
ICC	integrated chip card
ID	identifier
IEC	International Electrotechnical Commission
IFD	interface device
IMSI	international mobile subscriber identity
INS	instruction
IPR	intellectual property rights
ISO	International Organization for Standardization
ITU	International Telecommunications Union
JC	Java Card
JCP	Java Community Process
JCRE	Java Card runtime environment
JIT	just in time
JSR	Java specification request
Lc	length command
Le	length expected
MAC	message authentication code
MD5	Message Digest Algorithm 5
MF	master file
MIPS	microprocessor without interlocked pipeline stages
NOP	no operation
NPU	numeric processing unit
NVM	nonvolatile memory
OCF	open card framework
OCR	optical character recognition
P1, P2, P3	Parameter 1, Parameter 2, Parameter 3
PC/SC	Personal Computer/Smartcard
PCD	proximity coupling device
PGP	Pretty Good Privacy
PIN	personal identification number
PIX	proprietary application identifier extension
PKI	public key infrastructure
PPS	Protocol Parameter Selection
PUK	personal unblocking number
RACE	Research and Development in Advanced Communications Technologies in Europe
RAM	random access memory
Reg TP	Regulierungsbeh¨orde f¨ur Telekommunikation und Post (German regulatory agencies for telecommunication and postal services)
RF	radio frequency

RFC	Request For Comment
RFID	radio frequency identifier
RFU	reserved for future use
RID	registered application provider identifier
RIPEMD RACE	Integrity Primitives Evaluation Message Digest
RISC	reduced instruction set computer
RMI	remote method invocation
RND	random number
ROM	read-only memory
RSA	Rivest, Shamir and Adleman cryptographic algorithm
RST	reset
SAT SIM	Application Toolkit
SATSA	Security and Trust Services API
SECCOS	Secure Chip Card Operating System
SFI	short file identifier
SIM	subscriber identity module
SMS	short message service
SPA	simple power analysis
SPU	standard or proprietary use
SSC	send sequence counter
TDES	Triple DES (data encryption standard)
TETRA	Trans-European Trunked Radio
TLV	tag length value
TSCS	The Smartcard Simulator
UART	universal asynchronous receiver transmitter
UCS	universal character set
UICC	universal integrated chip card
UML	unified modelling language
UMTS	Universal Mobile Telecommunication System
USB	Universal Serial Bus
USIM	universal subscriber identity module
Vcc	supply voltage
VM	virtual machine
XML	extensible markup language
XOR	logical exclusive OR operation

1. INTRODUCTION TO SMARTCARD TECHNOLOGY

1.1. PROBLEM SCOPE

The scope of this study is developing a sample program working by using crypto algorithms and loading/writing of various data (informations) according to the EMV standards, testing of card access and investigation of the process of breaking the PIN numbers of the smart cards. In this project, two contact smart cards with ACOS2 operating system, a card reader, Microsoft.NET, C# and Visual Basic 6.0 as software languages.

Primary needs to develop our application;

- A smart card with microprocessor (with operating system e.g.ACOS2)
- A Smart Card Specification from manufacturer
- Information about operating system and file system
- Information about smart card key management
- Information about EMV standards. (From EMV Books - EMVCo Ltd.)
- 3DES, DES Crypto Algorithms
- Universal smart card commands and functions
- Information about PC/AC protocols
- A smart card reader

It is not possible to personalize smart cards in EMV standards without having information and knowledge on any of these steps. After these steps, developing of the method of cracking the pin code of smart cards in EMV standards is done by focusing on security loosies of smart cards.

1.2. OVERVIEW & BACKGROUND

A smartcard is a credit card-sized device that contains one or more integrated circuits (ICCs) and also may employ one or more of the following machine-readable technologies: magnetic stripe, bar code (linear or two-dimensional), contactless radio frequency transmitters, biometric information, encryption and authentication, or photo identification. The integrated circuit chip (ICC) embedded in the smartcard can act as a microcontroller or

computer. Data are stored in the chip's memory and can be accessed to complete various processing applications. The memory also contains the microcontroller chip operating system (COS), communications software, and can also contain encryption algorithms to make the application software and data unreadable. When used in conjunction with the appropriate applications, smartcards can provide enhanced security and the ability to record, store, and update data. When implemented properly, they can provide interoperability across services or agencies, and enable multiple applications or uses with a single card.

Smartcard technology can enable an organization to become more secure, efficient, and interoperable while delivering strong authentication and security, identity management, data management, customer support, and communications. The ICC, the technology on a card that makes it a "smartcard," provides a number of functions. Smartcard technology is commercially active and therefore provides additional benefits through commercial off-the-shelf (COTS) products and well-established technology standards.

Smartcard technology can address issues surrounding identity management and can also provide the means to eventually re-engineer inefficient processes with a high return on investment (ROI). In the identification of inefficient processes, outdated business practices, and low ROI programs, an organization can eliminate deficiencies, unnecessary costs, and under-used resources through the implementation of smartcard technology. The combination of smartcard technology with web-based applications, electronic commerce, and other business uses of the Internet can improve the quality of life for citizens and employees.[1]

[1] *Catherine Allen, "Smart Cards Part of U.S. Effort in Move to Electronic Banking," in Smart Card Technology International: The Global Journal of Advanced Card Technology, ed. Robin Townsend (London: Global Projects Group, 1995), 193-194.*

1.2.1. What is a Smart Card?

A smartcard is a small, tamperproof computer. The smartcard itself contains a CPU and some non-volatile storage. In most cards, some of the storage is tamperproof while the rest is accessible to any application that can talk to the card. This capability makes it possible for the card to keep some secrets, such as the private keys associated with any certificates it holds. The card itself actually performs its own cryptographic operations.

Smartcards currently come in two forms, contact and contactless:
Contact cards require a reader to facilitate the bidirectional connection. The card must be inserted into a device that touches the contact points on the card, which facilitate communication with the card's chip. Contact cards come in 3-volt and 5-volt models, as do current desktop CPUs. Contact card readers are commonly built into company or vendorowned buildings and assets, cellular phones, handheld devices, stand-alone devices that connect to a computer desktop' s serial or Universal Serial Bus (USB) port, laptop card slots, and keyboards.

Contactless cards use proximity couplers to get information to and from the card's chip. An antenna is wound around the circumference of the card and activated when the card is radiated in a specific distance from the coupler. The configuration of the card's antenna and the coupler facilitate connected states from a couple of centimeters to a couple of feet. The bidirectional transmission is encoded and can be encrypted by using a combination of a card vendor's hard-coded chip algorithms; randomly generated session numbers; and the card holder's certificate, secret key, or personal identification number (PIN). The sophistication of the connection can facilitate separate and discrete connections with multiple cards should they be within range of the coupler. Because contactless cards don't require physical contact with a reader, the usability range is expanded tremendously.

International standards govern the physical characteristics of smartcards. For example, the size of a card is covered by International Organization for Standardization (ISO) 7810. ISO 7816 and subsequent standards cover manufacturing parameters, physical and electrical

characteristics, location of the contact points, communication protocols, data storage, and more. Data layout and format, however, can vary from vendor to vendor[2].

1.2.2. CARD CLASSIFICATION

If you were to classify smartcards in the same manner as living beings in biology, you would obtain a tree chart similar to what is shown in Figure 1.1. The top level includes all types of cards, which can have various formats.

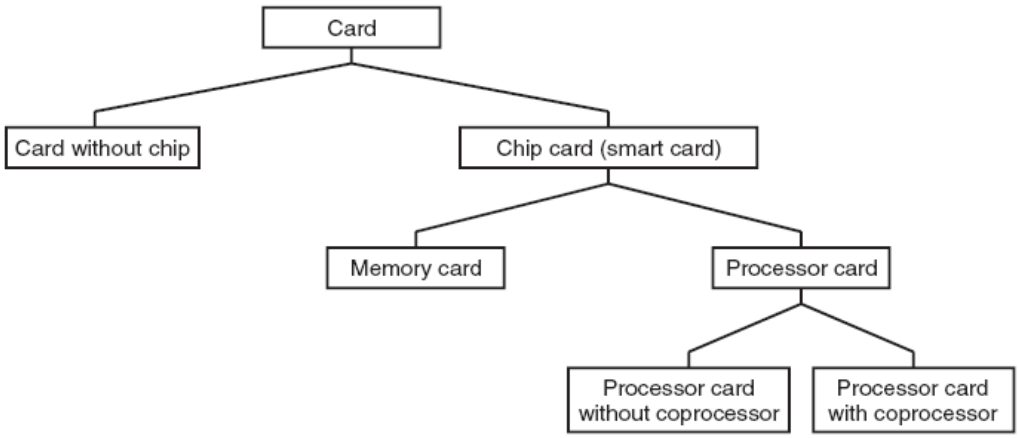


Figure 1.1 *Classification of cards with and without chips*[3]

Cards can be divided into two groups as, cards without chips and cards with chips in Figure 1.2. Logically enough, the latter type are called chip cards, which are also commonly known as smartcards. The chip, which is the essential distinguishing element, can be either a memory chip, in which case the card is called a memory card, or a microcontroller chip, in which case the card is called a processor card. Processor cards can be further subdivided into processor cards with or without coprocessors for executing asymmetric cryptographic algorithms such as RSA (Rivest, Shamir and Adleman) or ECC (elliptic curve cryptosystems).

[2] ***Microsoft TechNet*** *- http://www.microsoft.com/technet/security/guidance/identitymanagement/scard.mspx*
[3] ***Advanced Card Systems Ltd.*** *– China (Brochure)*

This classification provides an adequate overview of the most widely used types of cards. However, it can also be extended to include devices that use smartcard technology. The best-known examples of such devices are 'super smartcards' and tokens. A super smartcard has a direct user interface to the smartcard microcontroller, in the form of additional card elements such as a display and buttons. A token has a different form that is better suited to its intended use than the usual card format. Typical examples include tokens in the form of USB plugs that can be connected directly to a PC. However, the underlying technology is still the same as that of smartcards, with only the appearance being different.

1.2.3. TYPES OF CHIP CARDS

Often the terms "chip card," "integrated circuit card" and "smartcard" are used interchangeably, but they can mean different things. Cards are distinguished both by the type of chip that they contain and by the type of interface that they use to communicate with the reader. [4]

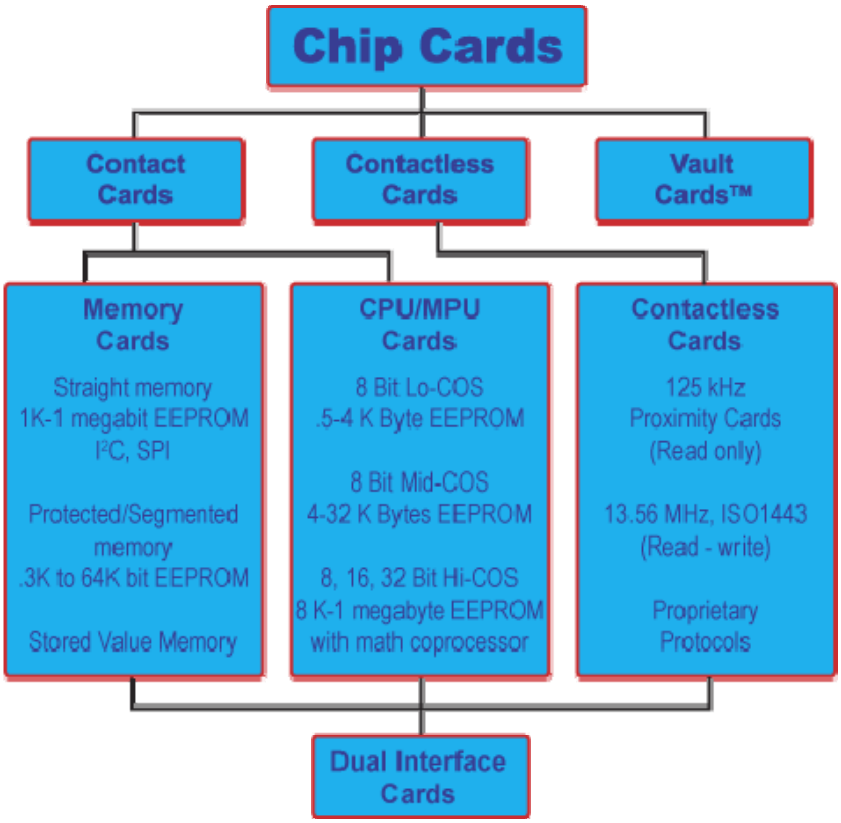


Figure 1.2 *Classification of cards with chips*[5]

[4] *Jack M. Kaplan, Smart Cards: The Global Information Passport (New York: International Thomson Computer Press, 1996), 69-75.*

[5] ***Smartcard Basics*** *- http://www.smartcardbasics.com/images/typesofcards.gif*

There are three different types of chips that can be associated with these cards: memory only, which includes serial-protected memory, wired logic and microcontroller. The terms "memory only," "wired logic" and "microcontroller" refer to the functionality that the chip provides. The following further discusses the types of chip cards.[6]

1.2.3.1. Memory-Only Integrated Circuit Chip Cards

Memory-only cards are "electronic magnetic stripes," and provide little more security than a magnetic stripe card. The two advantages they have over magnetic stripe cards are:

a) They have a higher data capacity (up to 16 kilobits (Kbits) compared with 80 bytes per track),

b) The read/write device is much less expensive. The memory-only chip cards do not contain logic or perform calculations; they simply store data. Serial-protected memory chip cards have a security feature not found in the memory-only chip card; they can contain a hardwired memory that cannot be overwritten.

Early versions of memory-only cards were read-only, low capacity (maximum of 160 units of value), prepaid disposable cards with little security. New versions include prepaid disposable cards that use read/write memory and binary counting schemes that allow the cards to carry more than 20,000 units of value. Many of these cards also have advanced logic-based authentication schemes built into the chip. Other memory-only cards have been developed for re-loadable stored value applications. The cards contain a purse, which can be protected through the use of a personal identification number (PIN) and counters, which limit the number of times the purse can be reloaded[6].

1.2.3.2. Wired Logic Integrated Circuit Card

A wired logic chip card contains a logic-based state machine that provides encryption and authenticated access to the memory and its contents. Wired logic cards provide a static file system supporting multiple applications, with optional encrypted access to memory contents. Their file systems and command set can only be changed by redesigning the logic of the IC. Wired logic-integrated chip cards include contactless variations such as I-Class or MIFARE[6].

---

[6] *Jose Luis Zoreda and Jose Manuel Oton, Smart Cards (Boston: Artech House, Inc., 1994), 5-6.*

1.2.3.3. Secure Microcontroller Integrated Circuit Chip Cards

Microcontroller cards contain a microcontroller, an operating system, and read/write memory that can be updated many times. The secure microcontroller chip card contains and executes logic and calculations and stores data in accordance with its operating system. The microcontroller card is like a miniature PC one can carry in a wallet. All it needs to operate is power and a communication terminal. Contact, contactless and dual-interface microcontroller ICs are available.

There are two primary types of chip card interfaces. These are contact and contactless. The terms "contact" and "contactless" describe the means by which electrical power is supplied to the ICC and by which data is transferred from the ICC to an interface (or card acceptance) device (reader). Cards may offer both contact and contactless interfaces by using two separate chips (sometimes called hybrid cards) or by using a dual-interface chip (sometimes called "combi" cards). *Jose Luis Zoreda and Jose Manuel Oton, Smart Cards (Boston: Artech House, Inc., 1994), 5-6.*

1.2.4. Contact Smartcards

A contact smartcard requires insertion into a smartcard reader with a direct connection to a conductive micromodule on the surface of the card.

A Software Implementation of AES for a Multos Smartcard-Yiannakis loannou pg.24

1.2.5. Contactless Smartcards

Contactless smartcards must only be in near proximity to the reader (generally within 10 centimeters or 3.94 inches) for data exchange to take place. The contactless data exchange takes place over radio frequency (RF) waves. The device that facilitates communication between the card and the reader are RF antennae internal to both the card and the reader.

These are smartcards that employ a radio frequency (RFID) between card and reader without physical insertion of the card. Instead the card is passed along the exterior of the reader and read. Types include proximity cards which are implemented as a read-only technology for building access. These cards function with a limited memory and

communicate at 125 MHz. True read & write contactless cards were first used in transportation for quick decrementing and re-loading of fare values where their lower security was not an issue. They communicate at 13.56 MHz, and conform to the ISO14443 standard. These cards are often straight memory types. They are also gaining popularity in retail stored value, since they can speed-up transactions and not lower transaction processing revenues (i.e. VISA and Mastercard), like traditional smartcards.

Variations of the ISO14443 specification include A, B, and C, which specify chips from either specific or various manufacturers. A = Philips B = everybody else and C = Sony chips. *(A Software Implementation of AES for a Multos Smartcard-Yiannakis loannou)*

1.2.6. Combination Cards

These are hybrids that employ both contact and contactless technology in one card. Combi-cards can also contain two different types of chips in contrast to a Dual-Interface card where a single chip manages both functions[7].

- **Hybrid Smartcards:** A hybrid card contains two chips on the card, one supporting a contact interface and one supporting a contactless interface. The chips contained on the card are generally not connected to each other[7].

- **Dual-Interface Chip Smartcards:** A dual-interface chip card contains a single chip that supports both contact and contactless interfaces. These dual-interface cards provide the functionality of both contact and contactless cards in a single form factor, with designs able to allow the same information to be accessed via contact or contactless readers.[7]

1.3. CARD FORMATS

The most common types of cards in current use have one feature in common, which is a thickness of 0.76 mm. As illustrated in Figure 1.3, all other dimensions can differ. These formats are not arbitrary. Instead, they are specified by international standards or by

[7] ***SmartCard Basics*** *- http://www.smartcardbasics.com/cardtypes.html*

specifications stipulated by major card issuers. This is also important, since at least in case of contact cards they must be able to fit into corresponding terminals or readers.

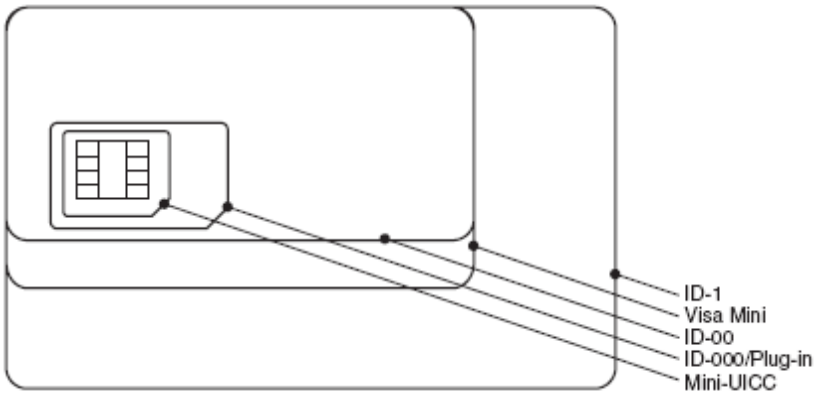


Figure 1.3. *Relative sizes of commonly used card formats*[8].

Typical smartcard formats are summarised in Table 1.1. The most commonly used card format, which is also undoubtedly the best known format, is ID-1. The reason it is so widely used is that practically all credit cards and other forms of payment cards are made in this format. Another name for this format is ID-000. This has become the standard format for cards used in mobile telephones.

The recently defined mini-UICC format is also available for the mobile telecommunications sector.[9]

Table 1.1 *- Summary of typical card formats. All stated dimensions are exclusive of tolerances*[9].

Card Format	Width (mm)	Height (mm)	Use
ID-1	85.6	54	Well known standard format
ID-00	66	33	Standardized for telecommunications, but not used.
Visa Mini	65.6	40	Payment Systems
Plug-in, ID-000	25	15	Telecommunications
Mini-UICC	15	12	Telecommunications

[8] ***GSA U.S.General Services Administration*** *– Goverment SmartCard Handbook*

[9] *Jose Luis Zoreda and Jose Manuel Oton, Smartcards (Boston: Artech House, Inc., 1994), 56-60.*

1.4. CARD ELEMENTS

The card body is usually more than just a carrier for the chip module. It also includes information for the user and card accepters and of course security elements for protection against forgery. Furthermore, the card body is an excellent advertising medium. The card issuers must coordinate all these functions, some of which are mutually contradictory, with their own specific wishes. The ultimate result is the issued card.[10]

1.4.1. Printing and Labeling

A rather wide variety of processes are available for printing and labelling cards. Text elements that are common to all cards of a series are normally applied using offset printing or silkscreen printing. Lasering is widely used for printing individual cards. This consists of using a laser beam to darken the surface of the plastic card body. This process produces irreversible card labelling, but it requires a certain amount of investment in technology. A more economical alternative is thermal transfer printing, which can also be used for colour printing. Digital printing processes for high-quality printing of individual cards are a relatively new development[10].

1.4.2. Embossing

The main advantage of embossing, which is commonly used with credit cards, is that the labelling can be transferred to paper using a simple stamping machine. The embossed section of the card can be restored to its original state by heating the card to a relatively high temperature. For this reason, the check digits at the end of the embossing usually extend into the hologram area. As the hologram will be visibly damaged if the card is heated, this makes it relatively easy to detect manipulation of the embossing[10].

1.4.3. Hologram

Technically sophisticated equipment is necessary to produce the white-light reflection holograms used on cards. As forgers usually do not have access to such equipment, holograms are commonly used on smartcards as security features. Some other reasons for

[10] *Jack M. Kaplan, Smartcards: The Global Information Passport (New York: International Thomson Computer Press, 1996), 72-75.*

using holograms are that they are inexpensive in large quantities, they can be checked directly by users, and the hologram cannot be removed from the smartcard without destroying it. Unfortunately, there is no link between the hologram and the microcontroller, which reduces its advantages from the perspective of the chip.[11]

1.4.4. Signature Panel

The signature panel is located on the rear of the card. It must be erasure-proof so that the signature on the panel cannot be removed without it being noticed. A coloured pattern is often printed on the signature strip, so any attempt to manipulate the signature will cause visible damage to the pattern[11].

1.4.5. Magnetic Stripe

With many types of cards, the only reason to retain the magnetic stripe (with its data storage capacity of a few hundred bytes) is compatibility with a widely distributed terminal infrastructure. However, it will still take a long time before magnetic-stripecards are fully replaced by smartcards, since they are significantly cheaper[11].

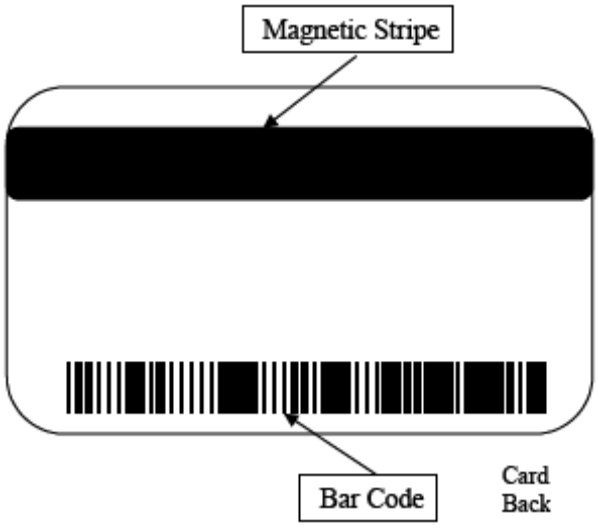
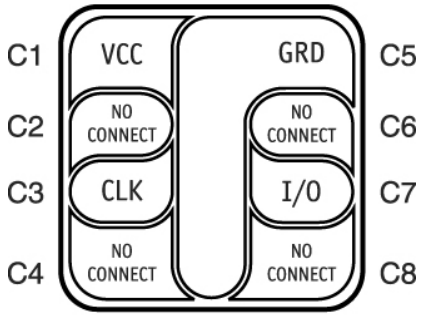


Figure 1.4 *Magstripe Card*

[11] *Jack M. Kaplan, Smartcards: The Global Information Passport (New York: International Thomson Computer Press, 1996), 72-75.*

1.4.6. Chip Module

The chip module is a protective housing for the microcontroller chip, which is fitted to the rear of the module. The module can have six or eight visible contacts on its external surface, although modern smartcards need only five contacts. The other contacts are reserved for future applications. Figure 1.5 shows the signal assignment of the contacts of a chip module.



(http://www.smartcardbasics.com/images/basicmodule.gif)

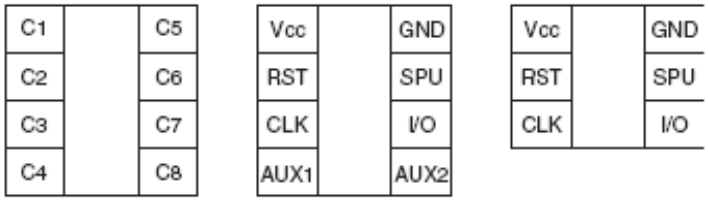


Figure 1.5. *Contact assignments of a smartcard module. Abbreviations;*
Vcc = Supply voltage, RST = Reset, CLK = Clock, AUX1 = Auxiliary 1,
GND = Ground, SPU = Standard or Proprietary Use, I/O = Input/Output, AUX2 = Auxiliary 2

1.4.7. Antenna

Smartcards that communicate without using contacts must have an integrated antenna in the card body. The antenna is a sort of coil consisting of several turns along the outer edge of the entire card. Various methods can be used to produce the antenna. Methods that are used in practice include a coil of thin copper wire embedded in the card body, etched copper tracks, and printed coils.[12]

[12] Jack M. Kaplan, Smartcards: The Global Information Passport (New York: International Thomson Computer Press, 1996), 72-75.

1.5. SMARTCARD MICROCONTROLLERS

The characteristics of a smartcard are largely determined by its microcontroller. Single chip microcontrollers are normally used. A single-chip microcontroller consists of a small silicon chip equipped with all the functions necessary for its intended use. Smartcard microcontrollers are not standard microcontrollers such as those used in coffee machines and toasters, but are instead chips specially adapted for use in smartcards.

Besides all these functional parameters, there is another essential item: security functions. Smartcard microcontrollers are especially hardened against attacks. This includes detecting undervoltage and overvoltage conditions and detecting clock frequencies outside the specified range. These microcontrollers also incorporate light and temperature sensors to enable them to recognize attacks via these routes and respond accordingly.

Besides technologically advanced smartcard microcontrollers, there are also memory chips which are essentially intended to be used as simple data storage devices with fixed logic circuitry designed by the semiconductor manufacturer. Figure 1.5 shows the basic functional groups present on the chip. The ROM (read-only memory) contains data about the chip type. The EEPROM (electrically erasable programmable read-only memory) provides the storage area for a unique chip identification number and data stored in read/write memory. A terminal can store several hundred bytes to a few thousand bytes of data here.

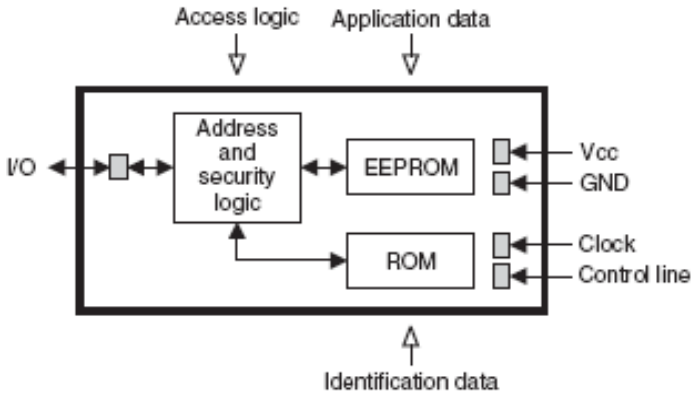


Figure 1.6 *Block diagram of a memory chip for a smartcard with a contact interface.*[13]

[13] *Smartcard Applications: Design Models for using and programming smartcards* W. Rankl 2007 Ltd.

The security logic, which varies according to the chip type, monitors access to the data. For instance, successful verification of a PIN (personal identification number) in the memory chip may be necessary before write access is possible.

Microcontrollers for smartcards have significantly more functionality than simple memory chips, as can be seen from Figure 1.6 on the facing page. The CPU (central processing unit) is a freely programmable control unit that executes the machine instructions of the operating system, which is located in the ROM. The CPU is assisted by a numerical coprocessor (NPU – numeric processing unit) for numerical calculations, particularly those dealing with cryptography. These special processors combine extremely high performance with low power consumption. Operating system extensions and the actual applications and associated data are stored in the EEPROM. Just as in a PC, the RAM (random-access memory) serves as working memory to hold data during operation.

Additional interfaces are integrated into smartcard microcontrollers to expand their range of potential uses. For instance, the commonly used half-duplex bit-serial port can be augmented by a USB interface or a wireless communication interface. Semiconductor manufacturers usually base such developments on existing smartcard microcontrollers, which are upgraded to support the additional interfaces. The result is thus a single-chip microcontroller that can communicate with the outside world via additional interfaces.

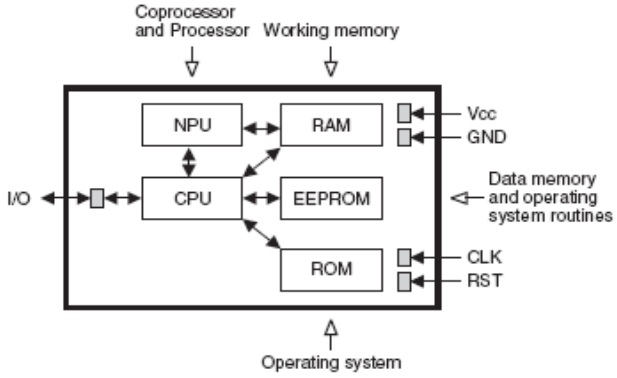


Figure 1.7 *Block diagram of a microcontroller for a smartcard with a contact interface.*[14]

[14] *Programming smartcards* W. Rankl 2007 Ltd

1.5.1. Processor

If you analyse the sales volumes of currently used smartcard microcontrollers, you will find that most of them still have an 8-bit CPU. This is usually a simple 8051 CPU, which has proved itself over the last two decades, along with a few extensions. The processing power of such a CPU is sufficient for all operating systems that do not include an interpreter. However, if the operating system must provide a Java interpreter, there is a distinct preference for microcontrollers with 16-bit processors. Some of these processors are also based on a modified 8051 architecture.

There are also a few smartcard microcontrollers that are based on well-known 32-bit processor families such as ARM 7 or MIPS. The limiting factor for using such highperformance processors is the chip area. There is a more or less direct relationship between chip area and price, and a 32-bit processor occupies a significantly larger area than an 8-bit processor. It is often more economical to invest in optimizing the speed of the software than to use a processor that needs more chip area. This is ultimately a consequence of the fact that smartcards have to be low-cost, mass-production items.

1.5.2. Memory

In addition to a processor, every microcontroller needs several types of memory with differing characteristics. The main type of nonvolatile memory used in smartcard microcontrollers is ROM. If the data located in memory must be modified in operation, electrically erasable memory (EEPROM) is used.

2. SMART CARD STANDARDS

Smartcard standards govern physical properties, communication characteristics, and application identifiers of the embedded chip and data. Almost all standards refer to the ISO 7816-1, ISO 7816-2, and ISO 7816-3 as a base reference.

2.1. ISO (International Standards Organization)

This organization facilitates the creation of voluntary standards through a process that is open to all parties. ISO 7816 is the international standard for integrated-circuit cards (commonly known as smartcards) that use electrical contacts on the card, as well as cards that communicate with readers and terminals without contacts, as with radio frequency (RF/Contactless) technology.[15]

2.1.1. ISO 7816 Summary

This is a quick overview of what the 7816 specifications cover. As these can be in revision at any time, check with ISO for the latest updates. ISO 7816 has six parts. *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.1. ISO 7816-1

Physical Characteristics, 1987; defines the physical dimensions of contact smartcards and their resistance to static electricity, electromagnetic radiation and mechanical stress. It also describes the physical location of an IC card's magnetic stripe and embossing area. *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.2. ISO 7816-2

Dimensions and Location of Contacts, 1988; defines the location, purpose and electrical characteristics of the card's metallic contacts. *(http://en.wikipedia.org/wiki/ISO_7816)*

[15] *ANSI American National Standards Institute. ANSI's address and phone is: 11 West 42nd Street, New York, NY 10036.*

2.1.1.3. ISO 7816-3

Electronic Signals and Transmission Protocols, 1989; defines the voltage and current requirements for the electrical contacts as defined in part 2 and asynchronous half-duplex character transmission protocol (T=0). Amendment 1: 1992, Protocol type T=1, asynchronous half duplex block transmission protocol. *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.4. ISO 7816-4

Inter-industry Commands for Interchange; establishes a set of commands for CPU cards across all industries to provide access, security and transmission of card data. Within this basic kernel, for example, are commands to read, write and update records.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.5. ISO 7816-5

Numbering System and Registration Procedure for Application Identifiers (AID); sets standards for Application Identifiers. An AID has two parts. The first is a Registered Application Provider Identifier (RID) of five bytes that is unique to the vendor. The second part is a variable length field of up to 11 bytes that RIDs can use to identify specific applications. *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.6. ISO 7816-6

Inter -industry data elements; physical transportation of device and transaction data, answer to reset and transmission protocols.

***The specifications permit two transmission protocols*:** Character protocol (T=0) or block protocol (T=1). A card may support either but not both. *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.7. ISO 7816-7

Inter-industry command for Structured Card Query Language (SCQL); This document specifies the concept of a SCQL database (SCQL = Structured Card Query Language based on SQL, see MS ISO 9075), and the related inter-industry enhanced commands.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.8. ISO 7816-8 (commands for security operations)

Created in 1995 and updated in 2004. It specifies interindustry commands for integrated circuit cards (either with contacts or without contacts) that may be used for cryptographic operations. These commands are complementary to and based on the commands listed in ISO/IEC 7816-4.

The choice and conditions of use of cryptographic mechanisms may affect card exportability. The evaluation of the suitability of algorithms and protocols is outside the scope of ISO/IEC 7816-8.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.9. ISO 7816-9 (commands for card management)

Commands for Card Management; specifies a description and coding of the life cycle of cards and related objects, a description and coding of security attributes of card related objects, functions and syntax of additional inter-industry commands, data elements associated with these commands, and a mechanism for initiating card-originated messages.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.10. ISO 7816-10 (electronic signals and answer to reset for syncronous cards)

Electrical signals and answer to reset for synchronous cards; this part of ISO 7816 specifies the power, signal structures, and the structure for the answer to reset between an integrated circuit card(s) with synchronous transmission and an interface device such as a terminal.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.11. ISO 7816-11 (personel verification through biometric methods)

Personal verification through biometric methods; currently a draft.
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.12. ISO 7816-12 (cards with contacts)

ISO/IEC 7816-12 created in 2005.

- The electrical conditions when a USB-ICC is operated by an interface device - for those contact fields that are not used, when the USB interface is applied;
- The USB standard descriptors and the USB-ICC class specific descriptor;
- the data transfer between host and USB-ICC using bulk transfers or control transfers;
- The control transfers which allow two different protocols named version A and version B;
- The (optional) interrupt transfers to indicate asynchronous events;
- Status and error conditions.

ISO/IEC 7816-12 provides two protocols for control transfers. This is to support the protocol T=0 (version A) or to use the transfer on APDU level (version B). *(http://en.wikipedia.org/wiki/ISO_7816)*

2.1.1.13. ISO 7816-13 (application management in multi-application environment)

As of 2006, this document is in development (source) and is supposed to integrate methods from the GlobalPlatform standard, like its Secure Channel Protocols (see this NIST report (in PDF format) for more information).
(http://en.wikipedia.org/wiki/ISO_7816)

2.1.1.14. ISO 7816-15 (Cryptographic information application)

This application contains information on cryptographic functionality. Further, ISO/IEC 7816-15 defines a common syntax (in ASN.1) and format for the cryptographic information and mechanisms to share this information whenever appropriate.

ISO/IEC 7816-15 supports the following capabilities created in 2004:

- Storage of multiple instances of cryptographic information in a card;
- Use of the cryptographic information;
- Retrieval of the cryptographic information;

- Cross-referencing of the cryptographic information with DOs defined in ISO/IEC 7816 when appropriate;
- Different authentication mechanisms;
- Multiple cryptographic algorithms.

(http://en.wikipedia.org/wiki/ISO_7816)

2.2. FIPS (Federal Information Processing Standards)

FIPS are developed by the Computer Security Division with in National Institute of Standards and Technology (NIST). FIPS standards are designed to protect federal assets including computer and telecommunications systems.

(http://en.wikipedia.org/wiki/ISO_7816)

2.2.1. FIPS 140 (1-3)

The security requirements contained in FIPS 140 (1-3) pertain to areas related to the secure design and implementation of a cryptographic module, specifically: cryptographic module specification; cryptographic module ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; design assurance; and mitigation of other attacks.

(http://en.wikipedia.org/wiki/ISO_7816)

2.2.2. FIPS 201

Currently a draft, this specification will cover all aspects of multifunction cards used in identity management systems throughout the U.S. government.

(http://en.wikipedia.org/wiki/ISO_7816)

2.3. EMV (EuroCard/EuroPay, MasterCard, Visa)

EMV is a standard for interoperation of IC cards ("Chip cards") and IC capable POS terminals and ATM' s, for authenticating credit and debit card payments. The name EMV comes from the initial letters of Europay, MasterCard and VISA, the three companies which originally cooperated to develop the standard. Europay International SA was

absorbed into Mastercard in 2002. JCB (formerly Japan Credit Bureau) joined the organization in December 2004, and American Express joined in February 2009. IC card systems based on EMV are being phased in across the world, under names such as "IC Credit" and "Chip and PIN". The EMV specification is also the basis of the Chip Authentication Program, where banks give customers hand-held card readers to perform online authenticated transactions.

The EMV standard defines the interaction at the physical, electrical, data and application levels between IC cards and IC card processing devices for financial transactions. Portions of the standard are heavily based on the IC Chip card interface defined in ISO 7816.[16]

The most widely known implementations of EMV standard are:
VSDC - VISA
MChip - MasterCard
AEIPS - American Express
J Smart – Japan Credit Bureau

2.3.1. Differences and Benefits of EMV

The purpose and goal of the EMV standard is to specify interoperability between EMV compliant IC cards and EMV compliant credit card payment terminals throughout the world. There are two major benefits to moving to smartcard based credit card payment systems: improved security (with associated fraud reduction), and the possibility for finer control of "offline" credit card transaction approvals.

The goals and benefits of EMV:

High level standard on terminal card API: It reduces the cost and time interval of software development (POS, ATM, HSM, etc.). The non EMV payment smartcard has its own crypto protections (RSA, DES) and is based on local private standards.

[16] *http://en.wikipedia.org/wiki/EMV*

EMV financial transactions are more secure against fraud than traditional credit card payments which use the data encoded in a magnetic stripe on the back of the card. This is due to the use of encryption algorithms such as DES, Triple-DES, RSA and SHA to provide authentication of the card to the processing terminal and the transaction processing center.

Although not the only possible method, the majority of implementations of EMV cards and terminals confirm the identity of the cardholder by requiring the entry of a PIN (Personal Identification Number) rather than signing a paper receipt. Wheather or not PIN authentication takes place depends upon the capabilities of the terminal and programming of the card.[17]

2.3.2. Control of the EMV Standard

The first version of EMV standard was published in 1999. Now the standard is defined and managed by the public corporation EMVCo LLC. The current members of EMVCo are JCB International, American Express, MasterCard Worldwide, and Visa, Inc. Each of these organizations owns one quarter of EMVCo and has representatives in the EMVCo organization and EMVCo working groups.

Recognition of compliance with the EMV standard (i.e. device certification) is issued by EMVCo following submission of results of testing performed by an accredited testing house.

EMV Compliance testing has two levels: EMV Level 1 which covers physical, electrical and transport level interfaces, and EMV Level 2 which covers payment application selection and credit financial transaction processing.

After passing a common EMVCo tests the software must be tested to comply with EMV standard (VISA VSDC, MasterCard MChip, etc.)

(EMVCo Ltd.-EMV Book 1)

[17] *http://en.wikipedia.org/wiki/EMV*

List of EMV documents and standards:

Since version 4.0, the official EMV standard documents, that define all the components in an EMV payment system, are published as four "books":

- Application Independent ICC to Terminal Interface Requirement
- Security and Key Management
- Application Specification
- Cardholder, Attendant, and Acquirer Interface Requirements

First EMV standard came into view in 1995 as EMV 2.0. This was upgraded to EMV 3.0 in 1996 with later ammendments to EMV3.1.1 in 1998. This was further ammended to version 4.0 in December 2000.

Version 4.0 became effective in June 2004.
Version, 4.1 became effective in June 2007.
Version EMV 4.2 is in effect since June 2008.[18]

2.4. PC / SC

A Microsoft proposed and implemented standard for cards and readers, called the PC/SC specification. This proposal only applies to CPU cards. They have also built into their CryptoAPI a framework that supports many security mechanisms for cards and systems. PC/SC is now a fairly common middleware interface for PC logon applications. The standard is a highly abstracted set of middleware components that allow for the most common reader card interactions.[19]

2.5. CEN (Comite' Europe' en De Normalisation)

ETSI (European Telecommunications Standards Institute) is focused on telecommunications, as with the GSM SIM for cellular telephones GSM 11.11 and ETSI300045. CEN can

[18] *http://en.wikipedia.org/wiki/EMV*
[19] *Blair Dillaway, "PC/SC Workgroup Specification for PC-ICC Interoperability," Presentation at CardTech/SecurTech '96 West, December 1996.*

be contacted at Rue de Stassart, 36 B-1050 Brussels, Belgium, attention to the Central Secretariat.[22]

2.6. HIPAA

HIPAA means the Health Insurance Portability and Accountability Act. The national standards for implementing a secure electronic health transaction system in the U.S. Example transactions affected by this include claims, enrollment, eligibility, payment and coordination of benefits. Smartcards are governed by the requirements of HIPAA pertaining to data security and patient privacy.[20]

2.7. IC Communication Standards

These existed for non-volatile memories before the chips were adopted for smartcard use. This specifically applies to the I2C and SPI EEPROM interfaces.[22]

2.8. SmartCard Standards

An important characteristic of smartcards is their broad compatibility with a wide variety of informatics infrastructures. These standards serve as basic reference documents for card manufacturers, operating system developers and application developers.

2.8.1. Standarts for Card Bodies

The general physical characteristics of cards are described in the ISO/IEC 7810 standard. It forms the basis for a further set of standards *(including TS 102 221 and EMV Book 1)*, which describe specific details and forms of implementation of the ISO/IEC standard in their introductory sections.

2.8.2. Standarts for Operating Systems

The most important set of standards for smartcard operating systems is the ISO/IEC 7816 family, which describes the essential informatic aspects of smartcards. The basic data trans-

[20, 22] *http://en.wikipedia.org/wiki/EMV*

mission parameters are ATR, PPS, T=0, and T=1. The requirements for contactless data transmission for proximity cards are described in the ISO/IEC 14 443 standard.

ISO/IEC 7816 standard contains a description of the file system, including the file types (MF, DF and EF), file structures (transparent, linear, linear variable, cyclic and TLV coded), and selection options.[5] The essential mechanisms for Secure Messaging are also specified in this standard. The ISO/IEC 7816 standard is also the most important reference for basic smartcard commands. Administrative commands are described in ISO/IEC 7816-9, and commands for cryptographic operations are described in ISO/IEC 7816-8 *(EMV Book 2 – EMVCo. Ltd.)*

2.9. File Management

Managing files is the principal task of a smartcard operating system. File management means not only providing read and write access to files and creating and deleting files, but also granting access privileges and monitoring compliance with access privileges. File management is especially important because most smartcard applications are file-based. File management in smartcards is almost entirely based on the provisions of the ISO/IEC 7816-4 standard. They specify a maximum possible functional scope, which in turn is implemented in actual smartcard operating systems only to the extent necessary.
(GEMPLUS 1999 / EPCOS-EMV Specification)

2.9.1. File Types

Smartcard file structures are always based on a tree structure with a root directory, as illustrated in Figure 2.1. The root directory of a smartcard, which is analogous to the ‘c:’ volume of a PC, is called the MF (master file) and is present only once in the file tree of the smartcard. It has the properties of a directory, which means it can only contain other directories and cannot store data directly.

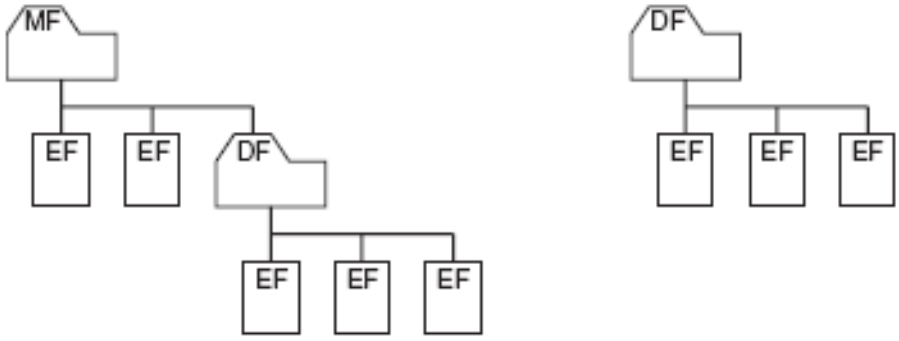


Figure 2.1 *The two possible forms of file-based applications in smartcards. A simple smartcard file system is shown on the left. It contains an MF with application-independent Efs located directly below the MF, along with a DF with application data contained in EFs. A DF without a visible MF is shown on the right. It also contains application data in the form of EFs located below the DF. This sort of DF is also called an ADF.*[21]

The directories of a smartcard are called DFs (dedicated files), and in theory they can be nested indefinitely. Three or four levels are commonly used in actual applications, and smartcard operating systems rarely support more than eight levels. The ADF (application dedicated file) is a special type of DF. It is a DF for a specific application and can be located in the file tree of the smartcard without there being any direct relationship to the root directory. Typically, it holds all the files of a particular application. ADFs are rarely encountered in actual practice.

The actual application data and operating system data are stored in EFs. EFs are always located in directories, and there are two possible types: working EFs and internal EFs. Working EFs are used to store application data that is accessible to the outside world via smartcard commands. By contrast, internal EFs are used by the smartcard operating system to store data for internal purposes.

For example, they can be used to store keys or a seed (initial value) for a random number generator. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

[21] *Advanced Card Systems Ltd. – Chapter 1 – What is a SmartCard ?*

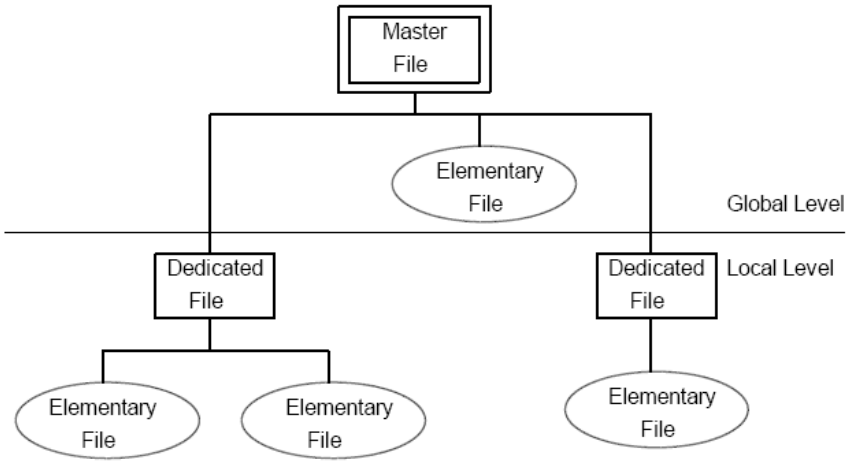


Figure 2.2. *MPCOS-EMV File Hierarchy*[22]

The files in MPCOS-EMV cards are organized into a 2-level hierarchy. The level formed by the Master File with Elementary Files directly beneath it is called the global level. The level formed by Dedicated Files with Elementary Files beneath them is called the local level.

The Master File: The Master File is the root of the MPCOS-EMV file structure. It is the equivalent of the DOS root directory. The Master File (only one per card) can have up to 63 Elementary and Dedicated files in it.

Dedicated Files: Dedicated Files store sets of Elementary Files. In MPCOS-EMV cards, each Dedicated File stores a set of Elementary Files that form an application. Dedicated Files are the equivalent of DOS directories. Each Dedicated File can store up to 63 Elementary Files, but no nested Dedicated Files are allowed.

Elementary Files: Elementary Files are the main component of the MPCOS-EMV file structure. They store application data. Different types of EFs are available in MPCOS-EMV, these are Purse files, secret code files, key files, transaction manager files, transparent files, linear fixed files, linear variable files and cyclic elementary files.

[22] *GEMPLUS 1999 / MPCOS-EMV Specification pg.4*

2.9.2. File Names

As smartcards are always used under the control of a terminal, it is not necessary to make the file names compatible with human needs. Standard file names thus consist of a 2-byte data element called the FID (file identifier). The FID of the MF, which is '3F00', is reserved for this purpose. All other FIDs can be freely chosen. Table 2.1 lists the file names of commonly used types of smartcard files and summarises their key characteristics.

Each directory file (DF) has a supplementary name in addition to its FID, and it can be addressed in the file tree using this supplementary name. This supplementary name is called the DF name, and it usually includes an AID (application identifier). The AID consists of an RID (registered application provider identifier) and a PIX (proprietary application identifier extension). RIDs can be registered officially to ensure that they are unique throughout the world. In this case, the PIX can be used as necessary to further identify a specific DF. This makes it possible to define a unique name for a specific smartcard application, which can then be used to recognize and select it in every smartcard. The EFs provided to hold data are also assigned FIDs, similar to all smartcard files. In addition, each EF has an SFI (short file identifier), which can be provided as a parameter of a read or write command to select the EF directly.[23]

Table 2.1. *Possible file names as specified by ISO/IEC 7816-4* [23(a)].

Data Type	File Name	Size	Value Range
MF (master file)	FID (file identifier)	2 bytes	'3F00'
DF (dedicated file)	FID (file identifier)	2 bytes	0 … 'F … F'
	DF name (usually includes an AID)	1-16 bytes	0 … 'F … F'
	AID (RID || PIX)	5-16 bytes	According to AID Definition
EF (elementary file)	FID (file identifier)	2 bytes	0 … 'FFFF'
	SFI (short file identifier)	5 bits	1 … '30'

[23, 26(a)] *GEMPLUS 1999 / EPCOS-EMV Specification*

2.9.3. File Structures

Smartcard data files (EFs) have internal structures. This means that the data stored in the files can be arranged in various ways. Five different structures are available, as illustrate in Figure 2.3.

In the transparent structure, the data items are arranged as a series of bytes (byte string). The commands READ BINARY and UPDATE BINARY can be used to read data from or write data to this file structure using parameters that specify an integral number of bytes and an offset from the start of the file. This EF structure is a general-purpose structure that can be put to a wide variety of uses.

Besides the transparent file structure, there are three record-oriented file structures. EFs with a linear fixed file structure can be used to store equal-length records. The linear variable file structure allows the records to have different lengths. If records with different lengths must be stored in a smartcard, the amount of memory space required will be less if a linear variable EF is used than if a linear fixed EF is used. These two file structures are typically used to store personal data such as addresses or telephone numbers. The cyclic file structure extends the linear file structure to include a pointer that indicates which record was most recently written. This structure is thus ideal for a variety of log file applications.

The records of all record-oriented files can be read and written using the READ RECORD and UPDATE RECORD commands. Normally, it is only possible to read or write complete records although relatively recent operating systems also support access to partial records.

The fifth type of file structure enables data objects to be stored in a TLV structure. In such a structure, each data object is identified by tag (T) and length (L) elements, which are followed by the actual data or value (V). This file structure can also be used to store nested data objects. Data objects can be read and stored using the GET DATA and PUT DATA commands.

Types of smartcard files and summarises their key characteristics.

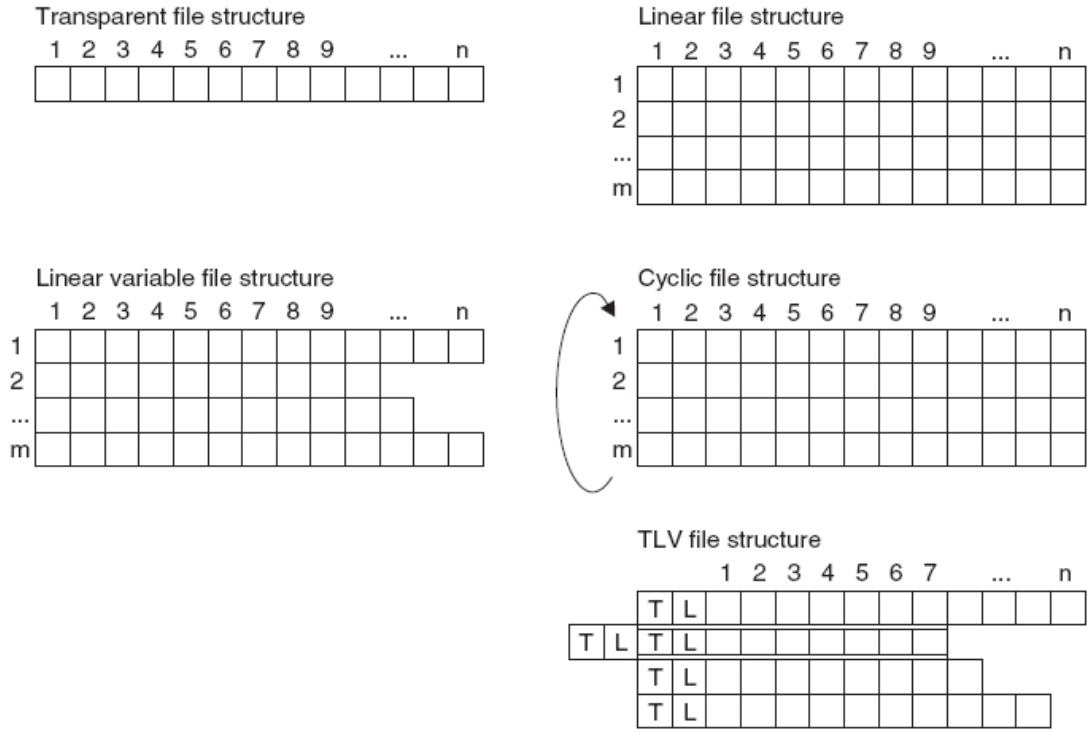


***Figure 2.3**. The five possible structures of data files (EFs) used in smartcards. Each cell in the diagrams represents a data byte.*[24]

***Table 2.2** File Structures and File Sizes*

File Structure	**Typical File Size**	
Transparent	*Total Size*	*1-33 023 bytes*
Linear	Record length	1-255 bytes
	Number of Records	1-254
Linear Variable	Record Length	1-255 bytes
	Number of Records	1-254
Cyclic	Record length	1-255 bytes
	Number of Records	1-254
TLV	Data object size	Not specified (typically 65 535 bytes)
	Number of data objects	Not specified (typically 255)

2.9.4. File Attributes

Smart files in smartcards can also have various attributes, depending on the specific operating system. The best-known set of attributes is shareable and not shareable. These

[24] *GEMPLUS 1999 / EPCOS-EMV Specification*

attributes can be used to specify for each file whether it permits concurrent read or write access via multiple logical channels. There are many other possible file attributes, but they are not standardized. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.9.5. File Selection

The smartcard SELECT command is used to explicitly select a file. A file must always be selected before it can be accessed with the usual commands such as READ BINARY or UPDATE BINARY.

One of the available identifiers (FID, DF Name or AID) must be used for selection, depending on the file type (MF, DF or EF). These identifiers do not have to be unique in the directory and file structure of a smartcard. Consequently, the selection options depend on the currently selected file. Figure 2.4 illustrates the selection methods that can normally be used in the directory and file structure.

Selection using a path name enables fast selection across several DFs with a single command. With this method, the path to the file to be selected is passed to the smartcard as a command parameter. This path can be referenced to the MF or to the currently selected file. This is the simplest selection option, and above all, it is the option that requires the least amount of transaction time. The MF can be selected in a similar manner. It can be selected from anywhere in the entire file tree using a single command.

The four commonly used read and write commands (READ BINARY, UPDATE BINARY, READ RECORD and UPDATE RECORD) also support file selection during command transaction (implicit selection). This eliminates the need to use SELECT to select the desired file before issuing the actual read or write command. This function is called implicit file selection, and it is quite useful for reducing file access times.[25]

[25] *(GEMPLUS 1999 / EPCOS-EMV Specification)*

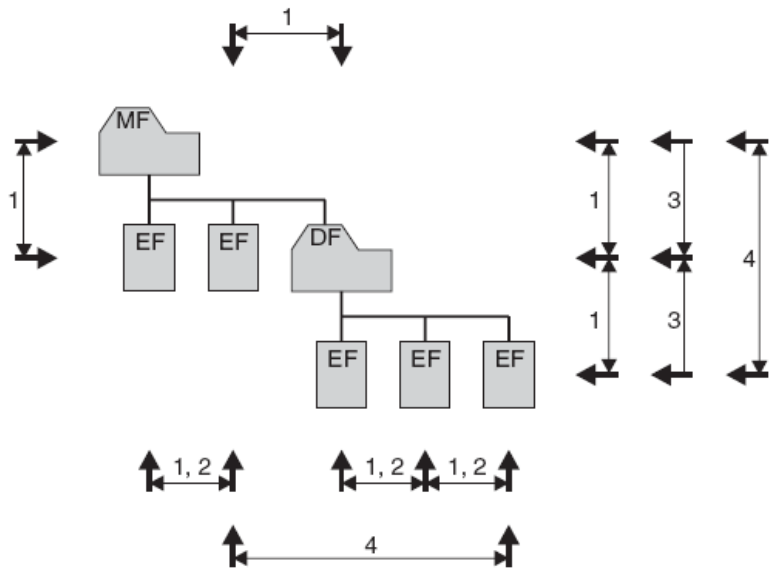


Figure 2.4 *File selection options for smartcards. Option 1 is explicit selection using an FID (file identifier); option 2 is implicit file selection using an SFI (short file identifier); option 3 is selection using a DF name; option 4 is selection using an FID (file identifier) and a path parameter.*[25]

2.9.6. Access Conditions

Access conditions associated with the files defined in a file system are an essential component of the file system. They specify which conditions must be satisfied to enable read or write access to the files. These conditions could be, for example, successful PIN verification or successful authentication of the terminal by the smartcard.

Two different methods are commonly used in smartcards for technical implementation of access conditions: state-based access conditions and rule-based access conditions. The first method has been used for more than a decade in large systems, such as the SIMs used in GSM mobile telecommunication systems. Rule-based access conditions were first published as a standard1 in the late 1990s. They are actually just a generalization and extension of the state-based method. As a result, all aspects of state-based access conditions can be reproduced using rule-based conditions.

(GEMPLUS 1999 / EPCOS-EMV Specification)

2.9.6.1. State-Based Access Conditions

In the case of state-based access conditions, each form of access (read or write) is only possible if a certain state has been attained, independent of other forms of access. The EFADN (abbreviated dialling number) file of a SIM can be used here as a typical example. This file can only be read using the READ RECORD command if PIN 1 has previously been correctly verified by the smartcard.

Nearly all file-based smartcard applications can be implemented with relative ease using state-based access conditions. However, a growing number of smartcard operating systems support the rule-based method, which is more future-proof and significantly more flexible. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.9.6.2. Rule-Based Access Conditions

Rule-based access conditions in smartcards are based on assigning all files (DFs and EFs) references to a record-oriented file containing sets of access rules. This file is assigned the name EF_{ARR} (access rule reference), and each reference is simply composed of the FID of the EF_{ARR} and a record number that addresses the appropriate set of rules. The FID of EF_{ARR} is freely selectable.

Each record in EF_{ARR} contains a set of rules for the various forms of access, such as read and write. As directory files can also be assigned references to an EF_{ARR}, it is also possible to define rules for creating and deleting files.

With rule-based access conditions, it is even possible to specify that certain files can only be accessed using Secure Messaging. The ISO/IEC 7816-9 standard forms the basis for the coding and the available functionality, but you should always consult the specifications of the smartcard operating system being used, since the standard provides many options and there are large differences between individual operating systems. The operating principle of rule-based access is illustrated in Figure 2.5.

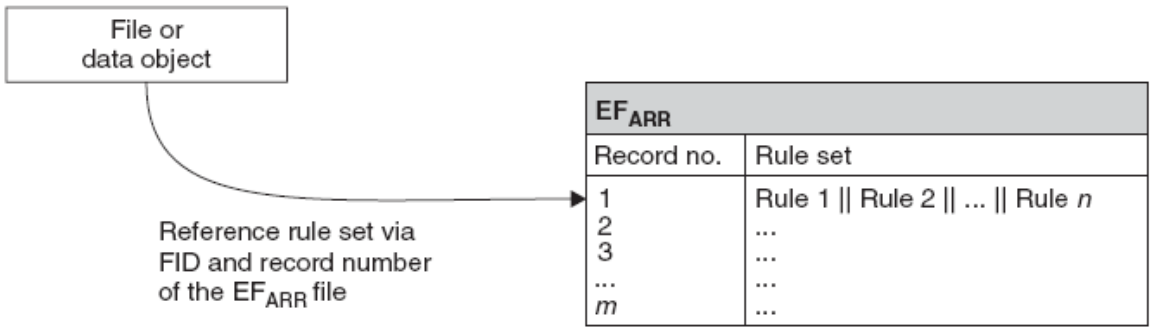


Figure 2.5 *Operating principle of using an EF*ARR *to manage rule-based access conditions for files and data objects.*[26]

All commonly encountered requirements for access to files and data objects in smartcard applications can be implemented using rule-based access conditions. Although this method is not especially simple, it is very powerful. As a comment regarding security, we can note here that it is essential to ensure that write accesses to EFARR can only be performed by authorized entities. Otherwise, the entire security of an application can be effectively by passed.

A mistake in connection with EFARR that can nearly be regarded as classic must be mentioned here. If it is possible to freely delete and create files in the directory containing EFARR, the following simple but highly effective attack is possible. The attacker first uses DELETE to delete EFARR and then uses CREATE to create a new EFARR in which all read and write conditions for the files that reference this file are set to 'always'. After this, the attacker can use standard commands to read all EFs containing application data, and of course the attacker can also alter the contents of these files. Although this is essentially a primitive form of attack, it shows quite clearly that even a sophisticated method such as rule-based access requires suitably careful planning.

(GEMPLUS 1999 / EPCOS-EMV Specification)

2.9.7. File Life Cycle

In the ideal case, it is possible to create, use and then delete files in a smartcard file system whenever so desired. In addition, the amount of free memory available to the file system is

[26] *(GEMPLUS 1999 / EPCOS-EMV Specification)*

ideally just as large after completion of this cycle as at the beginning. The life cycle of files, including all possible options, is illustrated in Figure 2.6.

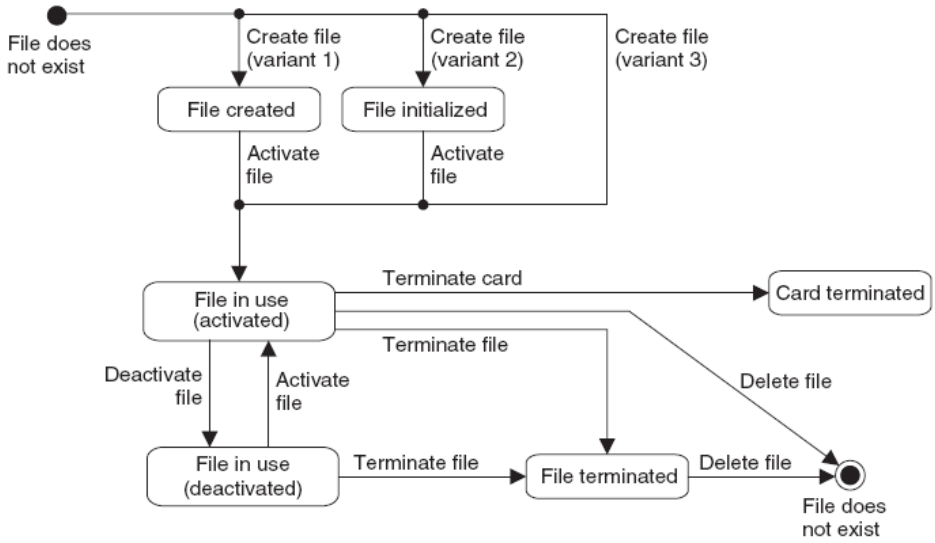


Figure 2.6 *States and associated state transitions during the entire life cycle of a file, as specified by ISO/IEC 7816-9*

All these options are actually available in large smartcard operating systems. On the other hand, simple operating systems often have restrictions in this regard. For instance, simple operating systems often do not allow files to be deleted once they have been created or if they do allow files to be deleted, the amount of available free memory may be reduced by several bytes for each pass through the described life cycle. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.10. EMV Commands

Aside from file management, commands are the most important functionality that a smartcard operating system provides to the outside world. Table 2.3 provides a summary of standard smartcard commands, and Table 2.4 provides a selected list of the most commonly encountered return codes sent by smartcards in response to commands received from a terminal. With regard to the exact coding of individual commands, you must always refer to the specifications of the smartcard operating system being used. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.10.1. EMV Administration Commands (commands for file operations)

The commands for file operations include SELECT, which is used to select a specific file, and READ BINARY and READ RECORD, which are used to read data from files having various structures. By contrast, UPDATE BINARY and UPDATE RECORD are the commands for writing data to files. The search commands SEARCH BINARY and SEARCH RECORD can be used to search for specific values in the EFs of the associated directory and file structure. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.10.1.1. Commands for Data Objects

Application data can be stored in data objects and/or files. GET DATA and PUT DATA read data from data objects and write data to data objects.

2.10.1.2. Commands for Security Functions

The best-known security function command is VERIFY, which is used to verify PINs. GET CHALLENGE requests a random number for a subsequent EXTERNAL AUTHENTICATE command, which is used to authenticate the outside world with respect to the smartcard. By contrast, INTERNAL AUTHENTICATE can be used to authenticate a smartcard with respect to the rest of the world by using a challenge–response process. MUTUAL AUTHENTICATION can be used to authenticate the smartcard and the outside world with respect to each other in a single operation. *(GEMPLUS 1999 / EPCOS-EMV Specification)*

2.10.2. EMV Payment Commands (for file management)

The commands for file management are used for administrative purposes to manage the directory files (DFs) and data files (EFs) in the file tree of a smartcard. This includes using CREATE FILE to create new files, APPEND RECORD to enlarge files, and DELETE FILE to delete existing files. The ACTIVATE FILE and DEACTIVATE FILE commands block and unblock files. The TERMINATE DF and TERMINATE EF commands permanently block files without deleting them from the file tree.

(GEMPLUS 1999 / EPCOS-EMV Specification)

2.10.3. EMV Commands & Descriptions

List of the most important smart commands defined by ISO/IEC 7816-4, -8, -9 and Open Platform.

***Table 2.3**. Administration commands*[27]

Administration Commands Table

Function Class	Name	Description
File	Select File	Selects an elemantary or a dedicated file for use in transactions. (Select a file operation)
	Write Binary	Writes data to an Elementary File by performing a logical OR operation between the current value of the write area and the value being written.
	Read Binary	Reads data from transparent elemantary files.
	Read Record	Reads data from structured file (record-oriented file).
	Update Binary	Updates data in an Elemantary File.
	Update Record	Updates data from a structured file.
	Search Binary	Search data from transparent elemantary file.
	Search Record	Search data from a structured file. (record-orianted file).
	Select File Key	Computes temporary administration key, and an authentication cryptogram.

[27] *(GEMPLUS 1999 / EPCOS-EMV Specification) and Smart Visa Programming.pdf (EMVco.)*

Function Class	Name	Description
File Management	Create File	Creates an Elementary or Dedicated File. (EF or DF)
	Append Record	Appends a new record to a structured file. (Create new record in a record- oriented file)
	Activate File	Reversibly unblock file.
	Deactivate File	Reversibly block file.
	Terminate DF/EF	Permanently block a file (DF or EF)
	Delete File	Delete a File DF or EF.

Function Class	Name	Description
Data Objects	Get Data	Read TLV-coded data objects.
	Put Data	Write TLV-coded data objects.

Function Class	Name	Description
Security	Verify	Compares the value submitted to the secret code number in the secret code file for the currently selected dedicated file. (Verify transferred data.)
	Get Challenge	Generates an eight-byte random number.(Request a random number (e.g. for a subsequent EXTERNAL AUTHENTICATE)
	Internal Authenticate	Causes the card the compute a cryptogram for verification by the outside the world.
	External Authenticate	Causes the card to check a cryptogram sent from the outside world.
	Mutual Authentication	Mutual authentication of the smartcard and the outside the world
	Perform Security Operation	Execute a cryptographic algorithm in the smartcard.
	Manage Security Environment	Manage security command parameters.
	Freezze Access Conditions	Locks or localizes a file access condition.

Function Class	Name	Description
Program Code Management	Load	Load a code-based application.
	Install	Install a code-based application.
	Put Key	Load a key for a code-based application.
	Set Status	Write state information for the life cycle of the smartcard or an application.
	Set Card Status	Sets the Personalization flag to 1 once personalization has been completed and sets other card parameters such as the size of the data units.
	Get Status	Read State information about a security domain, load file or application.
	Set Secret Code	Unlocks or changes a secret code in the local EFsc.
	Delete	Delete an object.

Function Class	Name	Description
Data Transmission	Get Response	Retrieves and erases the data prepared in RAM by the card in response to the previous command. (Request data for the T=0 transmission protocol from the smartcard.)
	Get Info	Retvieves information about the card.
	Switch Protocol	Switches the card to another protocol or speed.

Table 2.4 *Payment commands*[28]

Payment Commands Table

Function Class	Name	Description
Payment	Cancel Debit	Cancels the previous debit performed by a terminal and optionally replaces it by a new debit.
	Credit	Credits a purse.
	Debit	Debits a purse.
	Read Balance	Reads the specified purse balance value.
	Select Purse & Key	Selects the specified purse and key, then generates a new temporary payment transaction key and an authentication cryptogram.

[28] *(GEMPLUS 1999 / EPCOS-EMV Specification) and Smart Visa Programming.pdf (EMVco.)*

Set Option	Sets the following Sign command options. - Use current purse balance in the certificate calculation. - Clear the RAM parameters after executing the Sign command.
Sign	Generates a certificate for the previous transaction.

2.10.4. Return & Error Codes Meanings (Status Codes)

Table 2.5 *Error Codes*

Group	SW1 / SW2	Meaning
Normal Processing	'9000'	Process executed successfully.
	'61xx'	Processing completed successfully. xx data bytes are available in response and can be retrieved using GET RESPONSE.
Warning Processing	'62xx'	Data in nonvolatile memory not modified. See SW2 for details.
	'63xx'	Data in nonvolatile memory modified; see SW2 for details.
Execution Error	'64xx'	Data in nonvolatile memory not modified; see SW2 for details.
	'65xx'	Data in nonvolatile memory modified; see SW2 for details.
	'66xx'	Security-relevant result
Checking Error	'6700'	Incorrect length (no additional information).
	'68xx'	Functions in class byte not supported; see SW2 for details.
	'69xx'	Illegal command; see SW2 for details.
	'6Axx'	Incorrect P1/P2 parameters; see SW2 for details.
	'6B00'	Incorrect P1 or P2 parameter.
	'6Cxx'	Bad Le value; see SW2 for correct number of available data bytes.

Group	SW1 / SW2	Meaning
Checking Error	'6D00'	Command code invalid or not supported.
	'6E00'	Class not supported.
	'6F00'	No spesific diagnosis

2.11. Data Transmission

This master/slave principle pervades all communications with smartcards. After the electrical startup of the smartcard microcontroller, the terminal sends a reset signal to the smartcard, which responds to this signal with an ATR (answer to reset). This can optionally be followed by a PPS (protocol parameter selection), which transfers a set of parameters that modify the subsequent data transmission process. In this case as well, the smartcard only responds to an explicit request from the terminal. The actual transmission protocol, during which the smartcard only reacts to commands by sending responses, begins after this initialization phase.

There are two types of reset for smartcards: cold reset and warm reset. With a cold reset, the smartcard is started up from the power-down state and reset during this process. By contrast, with a warm reset the smartcard is already powered up and only receives a reset signal from the terminal.

Table 2.6 *Logical sequence of transactions during smartcard startup. The PPS transaction is optional and can be omitted if the parameters of the transmission protocol provided in the ATR will be used unchanged.*

IFD Terminal		**ICC (Smartcard)**
Reset	→	Startup smartcard operating system.
	←	ATR
Optional		
PPS Request	→	PPS processing.
	←	PPS response
Commands		
APDU 1 Command	→	Command processing.
	←	Response APDU 1 Command
…		…

The master – slave relationship also affects the behaviour of the chip hardware and the operating system, as illustrated in table 2.6. After the power-up sequence, the smartcard operating system is started up and an ATR is transmitted. After this, the smartcard enters a low-power sleep mode. It remains in this mode until the terminal transmits a command. The command is received and processed, and the response is sent back to the terminal. The smartcard then enters the sleep mode again and waits for the next command from the terminal, which causes it to return to the active mode. Alternatively, the terminal can initiate the power-down sequence at this point to shut down the smartcard.

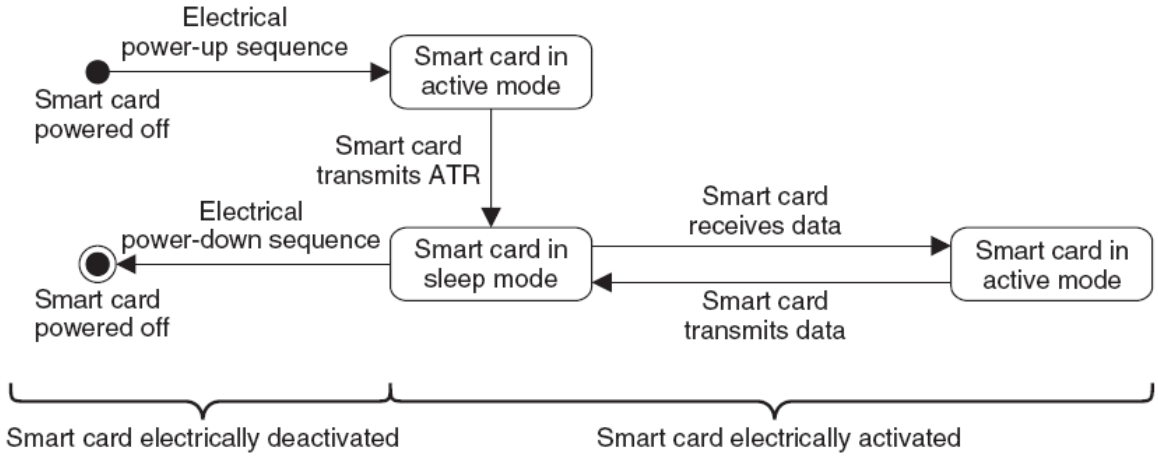


Figure 2.7 *The possible states of a smartcard operating system for transmitting and receiving data. The smartcard remains in the low-power sleep mode until it receives data via the interface. The power-down sequence can be executed at any desired time, but typically, it occurs in sleep mode.*

2.11.1. Answer to Reset (ATR)

The ATR (answer to reset) is the first communication a smartcard sends after detecting a reset. Among other things, the ATR provides the terminal with information about the transmission protocols and data transmission rates supported by the smartcard. The ATR is always transmitted with a divider value of 372, which yields a transmission data rate of 9 600 bps with a clock frequency of 3.5712 MHz.

(Microsoft MSDN http://msdn.microsoft.com/en-us/library/aa924246.aspx)-Automatic Terminal Recognition

2.11.2. Transmission Protocols

The transmission protocols define the communication processes between the terminal and the smartcard in case of successful transactions and the mechanisms to be used to handle detected transmission errors.

The most commonly used protocols for chip cards with memory chips are the ICC protocol and the 2-wire or 3-wire protocol. The T=0 and T=1 transmission protocols, which are commonly used with processor cards, are used almost without exception with contact-type processsor cards. There are already several types of smartcards that support the USB protocol, which is widely used in the PC environment. In the case of contactless microcontroller smartcards, the most widely used protocols are ISO/IEC 14 443 Type A and Type B.

Several abbreviations related to data transmission are commonly used in the processor card realm. A data record at the transmission level is called a TPDU (transport protocol data unit), while a data record at the application level is called an APDU (application protocol data unit). TPDUs and APDUs are defined for the commands sent to smartcards and the associated responses. A command APDU consists of a command header and a command body. The header is mandatory, but the body is optional. A response APDU consists of a response body and a response trailer. Only the trailer is mandatory in the response APDU.

A command APDU consists of four bytes designated as follows: Class (CLA), Instruction (INS), Parameter 1 (P1) and Parameter 2 (P2). The principle that the class byte should indicate the standard in which the command in question is specified is adhered to in most cases. The instruction byte defines the actual command, and the two parameters (P1 and P2) provide additional information about the command.

The command body can contain a maximum of three data elements. The first, Lc (length command), contains the length of the data in the command APDU, while Le (length expected) contains the length of the data requested from the smartcard, which is to be retur-ned in the response APDU.
Four different combinations are permitted for the command APDU. Each combination is called a case. There are only two variants for the response APDU. The T=0 or T=1 trans-

mission protocol, which is located below the application layer, looks after communicating these rigidly defined APDUs between the terminal and the smartcard.

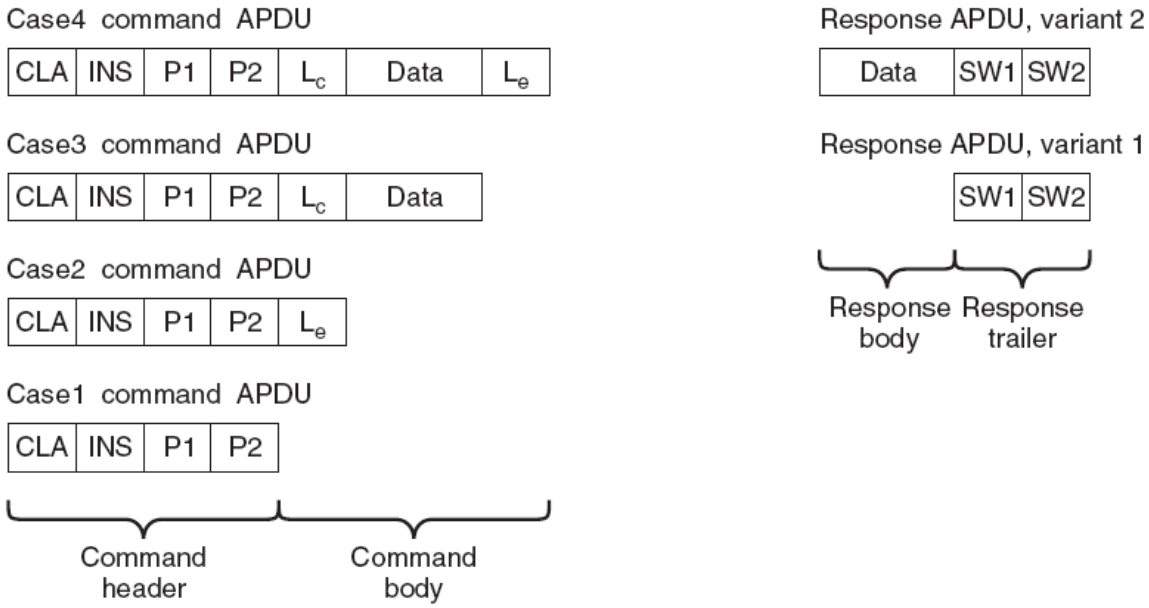


Figure 2.8 *The four different cases of command APDUs and the two different variants of response APDUs.*[29]

2.11.2.1. T=0 Transmission Protocol for Contact Cards

The T=0 transmission protocol is the oldest and most widely used protocol for smartcards. It is a byte-oriented transmission protocol with relatively poor layer separation. As a result, Case 4 commands in Figure 2.8 are not possible with T=0. Instead, the terminal must use the GET RESPONSE command to retrieve data to be provided to the terminal by the smartcard. However, this has not significantly restricted the use of the T=0 protocol, which is the standard protocol for the world's largest smartcard application: the SIMs and USIMs used in GSM and UMTS mobile telecommunication systems[29].

2.11.2.2. T=1 Transmission Protocol for Contact Cards

The block-oriented T=1 protocol has distinct layer separation, so all four cases of command APDUs can be used with this protocol. T=1 has a significantly more complicated structure than T=0, but it is also significantly more robust, thanks to its processes for detecting and resending blocks that contain transmission errors. T=1 is often used with payment cards

[29] *Advanced Card Systems Ltd. ACOS3 SmartCard Technical Specification*

and ID cards. It is indisputably a more modern protocol than T=0, but its advantages relative to T=0 are not large enough to threaten T=0 with becoming irrelevant[29].

2.11.2.3. USB Transmission Protocol for Contact Cards

The data transmission rate of T=0 or T=1 rarely exceeds 115 kbps in practice. This is too low for smartcards with large data memories. This is one of the reasons why the USB protocol (Universal Serial Bus) is slowly becoming established in the smartcard world. The second main reason is that USB provides compatibility with the PC environment. USB smartcards that support the 1.5 Mbps data rate of low-speed USB and even the 12 Mbps data rate of full-speed USB. *(Advanced Card Systems Ltd. ACOS3 SmartCard Technical Specification)*

2.11.2.4. Contactless Transmission Protocol

ISO/IEC 14 443 specifies the properties of contactless smartcards for use at a maximum distance of 10 cm from a terminal. Such cards are called proximity cards and they operate on the principle of inductive coupling via an RF magnetic field with a frequency of 13.56 MHz that is generated by the terminal or PCD (proximity coupling device).

Two different transmission techniques can be used for communication, since agreement on a single technique could not be reached during the preparation of the standard. They are called ISO/IEC 14 443 Type A and Type B and are mutually incompatible. However, commonly used terminals for contactless smartcards, as well as many types of smartcard microcontrollers, support both transmission techniques. *(Advanced Card Systems Ltd. ACOS3 SmartCard Technical Specification)*

2.11.3. Secure Messaging

For some applications, it is necessary to cryptographically secure data transmission to the smartcard to prevent eavesdropping and manipulation. This sort of security for smartcards is called Secure Messaging. It involves either adding an MAC (message authentication code) to each APDU or fully encrypting each APDU. It is also possible to use send sequence counters (SSCs) for the command and response APDUs to prevent successful playback of previous messages. Secure Messaging is a technically elegant solution that

provides transparent communication of APDUs and is highly configurable via parameters, but this comes at the price of complexity. *(Advanced Card Systems Ltd. ACOS3 SmartCard Technical Specification)*

2.12. Special Operating System Functions

In addition to file management functions, commands and data transmission, smartcard operating systems offer a range of special functions that can be used to develop applications. The available functions very depending on the hardware of the selected smartcard microcontroller and the operating system, so you should always compare the information provided here against the functional scope of the smartcard you intend to use before starting to create a specific application. *(Advanced Card Systems Ltd. ACOS3 SmartCard Technical Specification)*

2.12.1. Cryptographic Functions

The basic cryptographic functions of smartcards encompass the entire range of current cryptographic algorithms. Table 2.7 provides an overview. The basic functions are usually not directly available to the outside world at the interface, but are instead incorporated into commands that provide more abstract functions based on these functions.

One of these functions is encrypting and decrypting data. This can often be done at the level of performance that is suitable for real-time processing of audio or video data. Another function abstracted from the basic algorithms is authentication of entities, which is usually performed using a symmetric cryptographic algorithm. For compatibility reasons, DES (Data Encryption Standard) and Triple DES are always provided for this purpose, but the trend is clearly heading toward AES (Advanced Encryption Standard) with all three defined key lengths, which is inherently stronger than DES.

Table 2.7 *Types of Crypto Algorithms*

Types of Algorithms	**Algorithms**
Symmetric Cryptographic Algorithms	AES (128 bit, 196 bit, 256 bit) DES (56 bit), TDES (112 bit) IDEA (128 bit)
Asymmetric Cryptographic Algorithms	DSA ECDSA (160 bit, 256 bit) RSA (1024 bit, 2048 bit)
Types of Algorithms	**Algorithms**
Hash Algorithms	HMAC MD5 RIPEMD-160 SHA-1 and SHA-256
Key generation for symmetric cryptographic algorithms	Various
Key generation for symmetric cryptographic algorithms	Various
Random number generators	Various
Error detection codes	SHA-1 and SHA-256

2.13. Data Implementation

It is decisively important to always maintain a good overview of the data in a smartcard system. One way to do this is to generate and maintain a data dictionary of all the data in all components of the system. In the simplest case, this dictionary can consist of a table generated using a word processing program, but relatively complex database applications are often used for this purpose in complex systems. Table 2.8 shows an example of a typical entry in a data dictionary.

Table 2.8 *Data elements for a typical access control card and the associated read and write conditions for the administrative and operational phases. 'ADM1' is the administration PIN for card personalization and 'ADM2' is the administration PIN of the system operator.*

Life cycle Phase	Administrator Phase		Operational Phase	
Access	**Read**	**Write**	**Read**	**Write**
Card Number	Always	ADM1	Always	Never
Seed For PRN Generator	Never	ADM1	Never	Never
Current Random Number	Never	ADM1	Always	Never
PIN	Never	ADM1	Never	PIN or PUK
PIN error counter	Always	ADM1	Always	Never
Access privileges	Never	ADM1	PIN	ADM2
Access protocol	Never	ADM1	PIN	Never

2.14. Implementation of Files

The vast majority of smartcard applications are file-based applications consisting of a certain number of files (EFs – elementary files) with corresponding access conditions, all located in a directory (DF – dedicated file). The most important task for generating an application thus consists of specifying the files and associated access conditions.

2.14.1. Access Conditions

The next step is to define a systematic set of access conditions (access privileges). These conditions essentially relate to user identification using PIN verification and unilateral or mutual authentication of the smartcard and/or components of the outside world. Enter the results in the previously mentioned list of data elements. Next, you can systematically group the individual data elements into separate files, which form the basis for the filebased application. As part of this activity, you can also specify the file structures of the individual files. Table 2.10 shows some of the data elements of Table 2.9 assigned to several files.

Rule-based access conditions provide significantly more freedom for specifying file access privileges. However, this also creates significantly more complexity and thus more opportunities to make mistakes. A prerequisite for this type of access control is an EF_{ARR} (access rule reference) file in the DF of the application. Each record of the EF_{ARR} file contains a set of rules for accessing a particular file. Table 5.4 shows some typical access rules that could be placed in an EF_{ARR} file for the data elements and files listed in the table below.

Table 2.9 *Assignment of some of the data elements listed in the table below to files according to the specified read and write privileges.*

Data Element	**File**	**EF_{ARR} Rule**
Card Number	$EF_{Cardnumber}$	SE1, Rule Set 2
		SE2, Rule Set 1
Random Number Generator	$EF_{RNDSeed}$	SE1, Rule Set 3
Seed		SE2, Rule Set 4
PIN	EF_{PIN}	SE1, Rule Set 3
		SE2, Rule Set 2
Access Privileges	EF_{Priv1}	SE1, Rule Set 3
		SE2, Rule Set 3
Access Protocols	EF_{Prot}	SE1, Rule Set 3
		SE2, Rule Set 5

A set of rules must be generated corresponding to the previously generated list of files and associated accesses conditions and then distribute them among the appropriate records of the EF_{ARR} file. In the interest of simplicity, it is appropriate to note here that you should be economical when generating access rules.

All entities involved in the entire life cycle of the smartcard must be taken into account when defining access privilege groups. Initialization and personalization by the card manufacturer occur at the beginning of the smartcard life cycle. They are followed by an administrative phase with an application operator. It is certainly possible for several applications belonging to different operators to be present in a single smartcard. This must be reflected in the access rules. Specific privileges are usually necessary for the smartcard user, and possibly also for the card owner, although these privileges can often be combined. The access conditions for the EF_{ARR} file must be chosen carefully because this file governs all accesses to the files of the smartcard application. If the rules in EF_{ARR} can be modified, the entire security scheme can be bypassed. Consequently, write accesses to EF_{ARR} must be restricted to the administrative level and users must never be granted write access. If it is possible to foresee that the specified access rules will be adequate for any files to be created at some later date, write access to EF_{ARR} can also be set to ‘Never’.

Of course, it must be impossible to delete EF_{ARR}, as otherwise the EF_{ARR} file at the next higher level would become applicable and the access rules defined in that file could lead to security problems.

Table 2.10 *Example of the typical content of an EF_{ARR} file for a system with two different security environments (SEs): one for the administrative phase (SE1) and the other for the operational phase (SE2)*

SE1, Rule Set 1	READ: Always, UPDATE: Never Rule for readable data that cannot be modified during personalization. A typical example of such data is the code that identifies the microcontroller type and associated memory sizes.
SE1, Rule Set 2	READ: always, UPDATE: ADM1, CREATE: ADM1, DELETE: never Combined rule for file access and file management. The file access rule applies to data that must be read and modified during personalization such as name, address, date of birth and the like. Read access is necessary for verifying correct personalization in a subsequent step. The file management rule only allows data entry, since data deletion is not necessary during personalization.
SE1, Rule set 3	READ: never, UPDATE: ADM1 Rule for non-readable data that can be written during personalization. An example of such data is a seed value for a random number generator or a key for encrypting keys stored in the card.
SE2, Rule set 1	READ: always, UPDATE: never, CREATE: never, DELETE: never Combined rule for file access and file management. The file access rule applies to data that can be read freely but can never be modified after being stored. An example is the card number. The file management rule excludes creating new files and deleting files.
SE2, Rule set 2	READ: PIN, UPDATE: PIN, CREATE: ADM2, DELETE: ADM2 Combined rule for file access and file management. The file access rule applies to user data that can be read and modified after successful PIN verification. The file

	management rule permits creating and deleting files after successful verification of the PIN (ADM2) of the administrative entity.
SE2, Rule set 3	READ: PIN, UPDATE: ADM2, CREATE: ADM2, DELETE: ADM2 Combined rule for file access and file management. The file access rule applies to data that can only be read by the user and can only be modified by the system operator. An example of such data is the access privileges in a card used for computer access. The file management rule permits creating and deleting files after successful verification of the PIN (ADM2) of the administrative entity.
SE2, Rule set 4	READ: never, UPDATE: never This rule applies to data that is only used internally by the smartcard operating system and cannot be read or written by the outside world.
SE2, Rule set 5	READ: PIN, UPDATE: never This rule applies to data that can be read after successful verification but can only be written internally by the smartcard operating system.

Besides the access rules for the data files, a variety of other conditions must be defined for each application and entered in the EF_{ARR} file. They are the conditions for creating (CRE-ATE), deleting (DELETE), resizing (RESIZE), blocking (INVALIDATE), unbloc-king (REHABILITATE), and permanently blocking (LOCK) data files (EFs) and di-rectories (DFs).

For security reasons, the access conditions should always be specified as conservatively as possible. However, you must take care to ensure that suitable tests can still be performed after completion of the manufacturing phase in order to ensure correct personalization. These tests are usually based on reading or using personalization data (for example, for an authentication).[30]

[30] *EPCOS-EMV Product Overview version 1.0*

Similar considerations apply to accesses that are necessary for analysing complaints about cards in the field. The access privileges should at least be sufficiently lenient to make analysis of the problem possible, but they should not create any opportunities for attacks.

2.14.2. File Names

There are few restrictions on the file names (FIDs – file identifiers) for data files. The reserved FIDs specified in ISO/IEC 7816-4 are '3F00' for the root directory (MF) (master file), '3FFF' for selecting a file using a path name, and 'FFFF' for future use. From practical experience, it is a good idea to use the same upper byte for all FIDs assigned to a set of related files. The lower byte can then take the form of an incrementing number. For example, you could assign FIDs in the range 'A001'–'A004' to the files of an access control application and FIDs in the range 'B001'–'B008' to the files of a payment application in the same smartcard.

2.15. PIN Management

Numeric codes have been used for many years to authenticate card users. Only a simple ten-digit numeric keypad is needed to enter the codes and numbers are also suitable in terms of the ability of the general population to remember them.

However, this subject requires attention to more than just the technical aspects. You also have to take the behaviour and preferences of the users into consideration. Smartcards are used in all reaches of society, so only well established and widely accepted methods, such as PIN entry, should be employed.

This is also the reason for the widely used PIN code length of four digits. Although the theoretical security of PIN codes increases with the number of digits, the practical security reaches a maximum at four digits. If a larger number of digits is used, more users will either write the PIN code on the card or keep it in a handy location near the card. The number of cards that are blocked because of incorrect PIN entries also increases in proportion to the length of the PIN code, with a corresponding decline in user satisfaction and significantly increased administrative costs.

The PIN error counter normally blocks the application in the smartcard after three incorrect PIN entries. This is also regarded as tolerable with regard to security. In the case of longer PIN codes for special functions; the maximum value of the error counter before blocking occurs can be increased to as much as 10 for some applications.

The reset function can be implemented individually in each card by using a personal unblocking number (PUK). This requires the user of the smartcard to enter the PUK and his new PIN in the smartcard in a single session. A new PIN is necessary because the user has obviously forgotten his previous PIN.

2.16. Key Management

Key management for smartcard systems encompasses an enormous variety of options. Examples that can be found in actual practice range from a single key for all system functions to highly complicated key management schemes with 30 or more derived keys for each smartcard. The reasons for this wide range of variation can be found in the individual applications and the number of smartcards in the field for the specific applications.

The primary objectives of good key management are protecting the system against attackers and providing a good fallback position in the event of a successful attack. Consequently, simple smartcard systems that are not especially attractive targets of attack usually have correspondingly simple key management. The most elaborate forms of key management are used in electronic purse systems and smartcard systems for pay TV, both of which are unquestionably exposed to the most severe forms of attack.

Technically sophisticated key management schemes employ a different key for each function, which is called key diversification. The key for each function is called the master key for that function. On the basis of this master key, individual keys (derived keys) can be derived for each smartcard and supported function. Dynamic keys and session-specific keys can in turn be generated from the derived keys. These dynamic keys and session keys

are ultimately used by the cryptographic algorithms for the actual functions. Figure 2.9 shows this in graphic form.

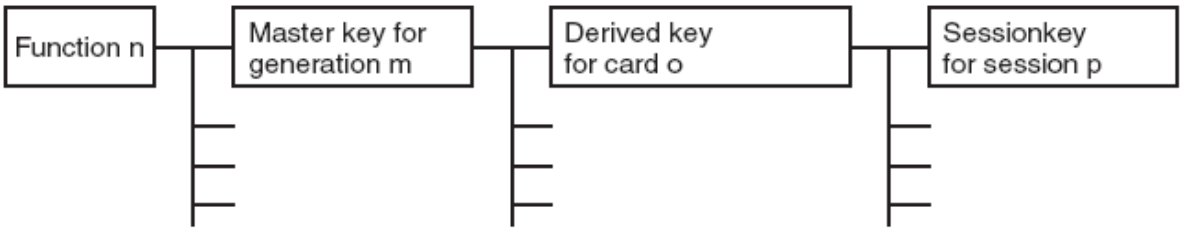


Figure 2.9 *Key hierarchy of an elaborate key management system, such as is used in electronic purse card systems, with a separate key hierarchy for each smartcard function. The number of key generations stored in individual smartcards is typically five or less.*

This means that an attacker must work his way along the entire chain, from the session specific keys through the derived keys to the master key, to fully break the cryptographic protection of the smartcard system for a particular function. To make things even more difficult for attackers, several generations of keys can be stored in each smartcard so that the system can switch from one generation to the next at regular intervals or as necessary. Alternatively, means can be provided to download new keys to the smartcards from the background system. The best way to do this is to use an asymmetric cryptographic algorithm such as RSA.

3. MATERIAL & METHODS

3.1. System Analysis

While doing the system analysis, it is a very important fact that the smart card to be used to be compatible with the EMV applications. Made a point of choosing a card that has public producer specifications and easy to provide, because producers conceal the EMV compatible smartcard standards for the security causes. Programming specifications of these cards can only be getten in only mass card purchases with privacy aggreements. Otherwise the specifications are not public. For security reasons the card must work with algorithyms such as DES and 3DES. The ACOS2 operating system is a commonly used operating system. USB card readers to read the card are obtained with the smart card.

3.1.1. Requirements Analysis

Principally it is needed to be known and obeyed the cardinal standards to be able to program a smartcard with operating system. For this process, the document that the producer of the smart card has allready published must beread in details, then the commands and rules must be known clearly. It must be matching with the EMV standards of the cards producer. Matching EMV standards means highly secure in other words. For programming the card, a USB card reader that should be able to read the card in our hand is needed. USB card readers communicate with the software by using PC/AC protocole. Clearly understanding the key management subject that the card producer has written is a must for the usage of the card key and the terminal key. All the processes are suitable to the ISO 7816 norms.

ACOS2 Microprocessor Card and ACOS2 Card's Specification (Smart card operating system requirement / Issuer Specific Requirement)

PC / AC Smart Card Reader, EMV Specification, VISA Specification, DES / 3DES Algorithm, Key Management, Smart card commands, APDU commands and means, Microsoft Visual Basic 6.0 or Microsoft Visual Studio .NET.

Smart Card Personalization

Smart card personalization describes the general procedure in the personalization of an ACOS2 smart card. While the card personalization may be carried out in separate processing steps, the personalization process generally requires the execution of the steps described below.

The personalization of a new ACOS2 smart card is suggested to be carried out according to the following sequence:

1) Power up and reset the card.

2) Submit the default Issuer Code IC (the code is communicated to the card issuer by ACS; the code may be different for different batches of cards supplied).

3) Select the Personalization File (File ID = FF 02H) and write the required settings to the Option Register and the parameter N_OF_FILE. **Caution: Do not accidentally set the Personalization Bit and do not change the Security Option Register at this stage!**

4) Perform a card reset. After the reset, ACOS2 reads the Personalization File and accepts the new value of N_OF_FILE and the option bits stored in the Option Register.

5) Submit the default Issuer Code IC.

6) Select the User File Management File (File ID = FF 04H) and write the File Definition Blocks for the required User Files (WRITE RECORD command) with the security attributes set to 'Free Access'.

7) Select the individual User Files and initialize the data in the files as required (WRITE RECORD command).

8) Select the User File Management File (File ID = FF 04H) and write the required security attributes for all User Files (WRITE RECORD command). Verify the contents of the User

File Management File (READ RECORD command). **Caution: Do not accidentally change the other parameters in the File Definition Blocks.**

9) If applicable, select the Account File (File ID = FF 05H) and initialize the relevant data in the Account File (WRITE RECORD command). Verify the contents of the Account File (READ RECORD command).

10) If applicable, select the Account Security File (File ID = FF 06H) and initialize the account processing keys (WRITE RECORD command). Verify the contents of the Account Security File (READ RECORD command).

11) Select the Security File (File ID = FF 03H) and initialize all keys and codes (WRITE RECORD command). Verify the contents of the Security File (READ RECORD command)

12) Select the Personalization File (File ID = FF 02H) and initialize the Security Option Register and the remaining bytes of the Personalization File. **Set the Personalization Bit** (WRITE RECORD command). Verify the contents of the Personalization File (READ RECORD command). **Caution: Do not accidentally change the value of the Option Register and N_OF_FILE.**

13) Perform a card reset. The chip life cycle stage as indicated in the ATR should be 'User Stage'.

14) The correct personalization can be verified by submitting the secret codes and keys programmed in the card (AUTHENTICATE, SUBMIT CODE commands) and reading/writing the allocated data files and executing the Account commands.

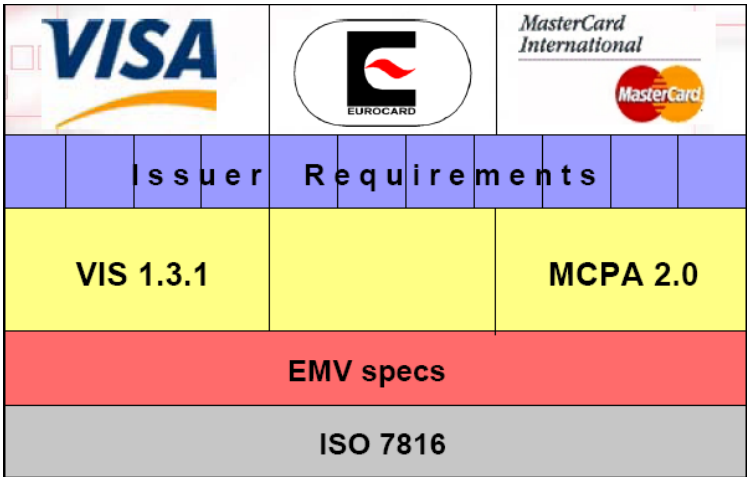


Figure 3.1 *Requirements*

(Smart)CPU card - 8bits/16 bits, 8051 or 6805 core

- ROM 3Kbytes to 32 Kbytes
- RAM ~100 bytes to 1 Kbytes
- EEPROM 512 bytes to 32 Kbytes

Smart card has four main elements, Central Process Unit(CPU), memory, input/output and Interface Device(IFD). Generally, smart card CPU is an 8 bit microcontroller. There are three types of memory inside smart card. Read Only Memory(ROM), Electrically Erasable Programmable Read Only Memory(EEPROM) and Random Access Memory(RAM). Smart card operating system and basic software are stored in the ROM. The EEPROM is used to install and run the application. The RAM is used to perform calculation process.

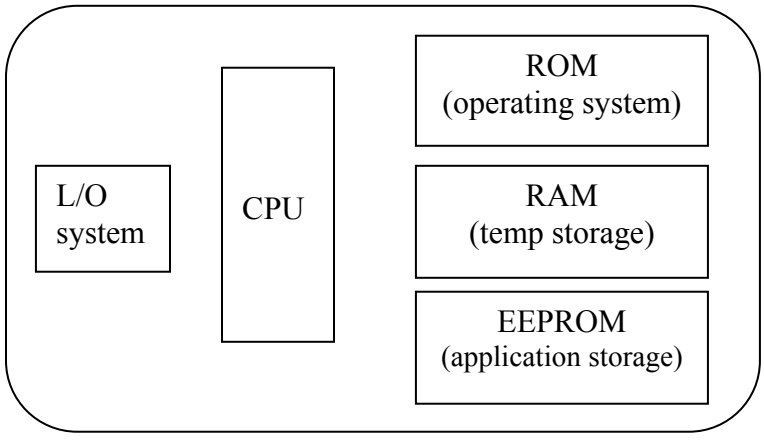


Figure 3.2 *Smart card elements*

EMV defines

-Electromechanical characteristics

-Logical interface and Transmission protocols

-Data Elements & commands

-Application selection

-Security aspects

EMV does not define

-Physical data structure

-Operating system

-Personalization procedure

VIS 1.3.1:

VISA options of EMV specifications. VIS is sufficient to develop a chip card application.

VIS defines

-Data elements and functions (from EMV)

-Card Risk Management processing

-Calculation of cryptograms

-Additional VISA specific commands and data elements

VIS does not define

-Proprietary processes, data & commands

-Operating system

-Personalization procedure

APDU Format

Application Protocol Data Unit(APDU) is a command message which is send from the application layer to the smart card and response message being sent from smart card to the application layer. Communication between smart card and card reader is performed using APDU message. There are two kinds of APDU, Command APDU and Response APDU.

Smart Card always waits for a "Command APDU" from a terminal. It then executes the action specified in the APDU and replies to the terminal with a Response APDU.

ISO-IN Command

CLA	INS	P1	P2	L_{in}	$Data_{in}$

ISO-OUT Command

CLA	INS	P1	P2	L_{out}

ACOS2 Security Mechanism:

a) Passive Authentication: VERIFY command with PIN password.

b) Active Authentication:

INTERNAL AUTHENTICATION with challenge
EXTERNAL AUTHENTICATION with response to challange

c) Data Authentication:

READ, WRITE, UPDATE command with secured messaging
Protecting Access Channel

d) Data Enchipherment: READ, WRITE, UPDATE command with ciphered data

COS Security

At implementation level
At command definition level

File Header – MF / DF Header

Byte 0 File descriptor byte
Byte 1 – 2 File ID

Byte 3 – 4 File size allocated

Byte 5 DF state AND mask

Byte 6 DF body size

Byte 7 – 8 Create – delete access

Byte 9 – 10 File size remaining

Byte 11 Current DF headers checksum

File Header – Transparent / TLV / Variable Record File

Byte 0 File descriptor byte

Byte 1 – 2 File ID

Byte 3 – 4 File size allocated

Byte 5 – 6 Read Access

Byte 7 – 8 Update Access

File Header – Linear / Cyclic Record File

Byte 0 File descriptor byte

Byte 1 – 2 File ID

Byte 3 – 4 Number of record, Record length

Byte 5 – 6 Read Access

Byte 7 – 8 Update Access

Security Policy

DF Access Condition (Create, Delete)

EF Access Condition (Read, Update)

B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**Description**
1	-	-	-	-	-	-	-	1=Ciphered
-	1	-	-	-	-	-	-	1=MAC
-	-	Level	-	-	-	-	-	0=key in current DF, 1=parent DF
-	-	-	x	x	x	x	x	11111 indicates that key is session key else indicates key number in the key file

B7	B6	B5	B4	B3	B2	B1	B0	Description
x	x	x	-	-	-	-	-	Access Logic
-	-	-	x	x	x	x	x	Access State

Each key record contains the following fields:

Byte 0, bit 7-5	ACTIVE_LOGIC
Byte 0, bit 4-0	ACTIVE_STATE
Byte 1, bit 4-0	NEXT_STATE
Byte 1, bit 7-5	RFU
Byte 2-3	Key capability
Byte 4, 5	max error/ usage counter
Byte 6, 7	error / usage counter
Byte 8	XX key content

Active Logic:

000 – Always

001 – Less Than (<)

010 – Less or Equal (<=)

011 – Equal (==)

100 – Greater or Equal (>==)

101 – Greater (>)

110 – Not Equal (! =)

111 – Never

State:

- COS has a state {0,1,2..31}
- State is defined by a 5 bits field
- State = 0 is the power-on default state (ALWAYS)
- State = 31 is the NEVER (LOCKED) state
- State is changed by a secret code presentation or key authentication
- Active Logic, Active State set the pre-condition to use a secret code / key

- Next State of secret code / key change to state machine
- If the state machine matches the Access, access is authorized.

Cryptographic Security:

Symmetrical e.g. DES (or 3DES)

- Good for many-to-one and one-to-one security (e.g. bank – customers)
- Simple key management
- Cannot achieve non-repudiation

Asymmetrical (public key) e.g. RSA, ECC

- Good for many-to-many security (e.g. electronic mail, electronic commerce)
- Complex key management infra-structure
- Public key compliments DES, not replace DES

DES Data Encryption:

- Symmetrical key algorithm
- Manipulate data in 8 bytes block
- Only known attack is exhaustive key search, 2 to the power of 56 computations
- 2 million years for today's PC @1ms per computation or a few hours with special designed hardware, parallel processing
- Security can be increased using triple DES

DES / 3DES:

- Single DES uses single length key (8 bytes), K(8)
- 3DES uses double length key (16 bytes), K(16) = KL(8) | KR(8) or KA(8) | KB(8)
- If the left and right part are the same, 3DES reduces to single DES
- Allows smooth migration from single DES to 3DES
- Least significant bit of each byte not used

3.1.2. Design

My application connects to smart card with PC / AC protocol. After determination of smart card key and terminal key, smart card formatted according to these keys.

Through the formatting process;

AA 11
BB 22
CC 33
} User Files

a) Creates the user file with keys.
b) Accordingly, string value data can be written in and read from all the files.

Read Record: The read record comment can be executed only if a file has been already selected with SELECT FILE command.

Record No: One byte logical record number
Length: Number of data bytes to be read from the record, max. 32
Data: Record data, Length bytes

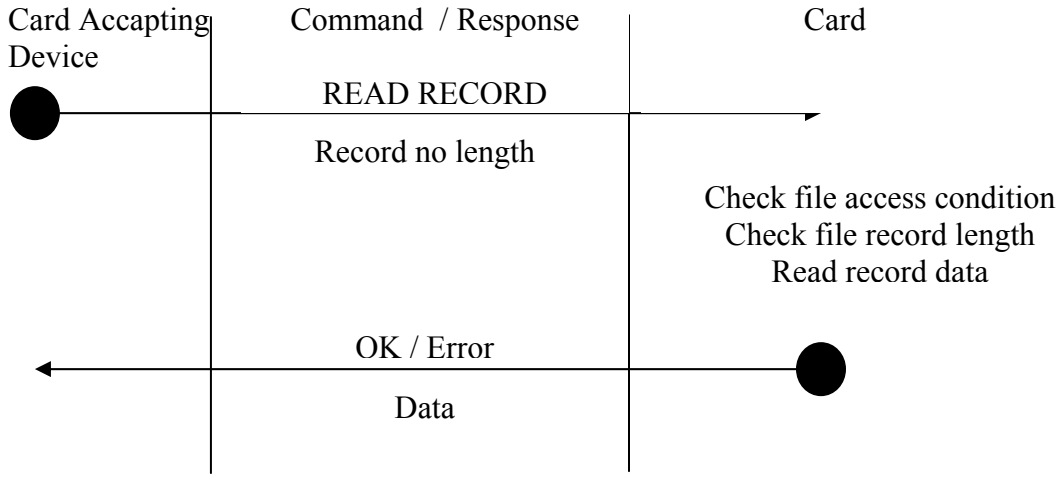


Figure 3.3 *Using Read Record (Select File)*

Write Record: The write record comment can be executed only if a file has been already selected with SELECT FILE command.

Record No: One byte logical record number

Data: Data bytes to be written to the record

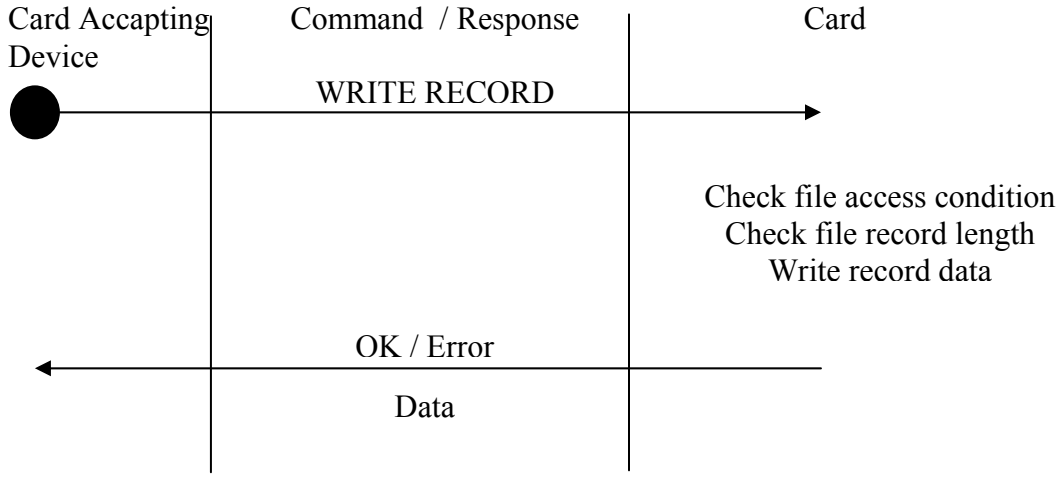


Figure 3.4*Using Write Record with Select File*

Account Processing Keys:

Record No	Byte 1							Byte 8
1	K_D							
2	K_{CR}							
3	K_{CF}							
4	K_{RD}							

Figure 3.5 *Key Storage for DES (keys are 8 byte long)*

K_D : The DEBIT key, used in the computation of the MAC for the DEBIT command.

K_{CR} : The CREDIT key, used in the computation of the MAC for the CREDIT command.

K_{CF} : The CERTIFY key, used in the computation of the MAC with the INQUIRE ACCOUNT command.

K_{RD} : The REVOKE DEBIT

Record No	Byte 1							Byte 8
1	Right half of K_D							
2	Right half of K_{CR}							
3	Right half of K_{CF}							
4	Right half of K_{RD}							
5	Left half of K_D							
6	Left half of K_{CR}							
7	Left half of K_{CF}							
8	Left half of K_{RD}							

Figure 3.6*Key Storage for 3DES (keys are 16 byte long)*

Secret Codes: ACOS2 provides some secret codes. Five Application Codes (AC), One Issuer Code (IC), One PIN Code (PIN)

Five Application Codes (AC1 ,….. AC5) are available to control the access to the data stored in data files. (Each Application Code is 8 bytes long). Issuer Code is provided to control access to data files and to privileged card functions; it is 8 bytes long. The PIN Code is provided to control access to data files. The PIN is 8 bytes long. The PIN is presented to the card with the SUBMIT CODE command.

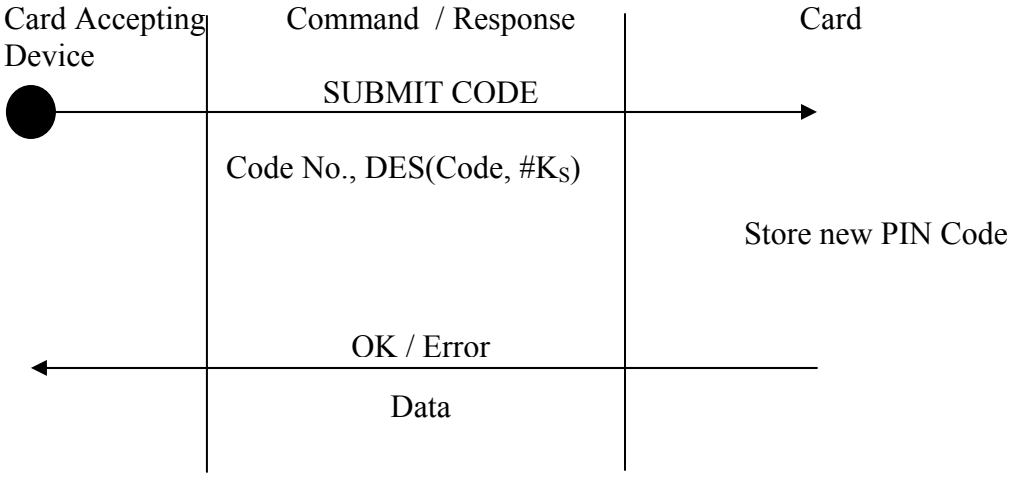


Figure 3.7 *Secret code submission and Error Counters*

Code No. Reference to the particular code that is submitted with the command:
1 ... 5 = Application Codes AC1...AC5 6 = PIN 7 = Issuer Code IC
Other values for Code No. are not allowed and will be rejected by the card.

Code The 8 bytes secret code to be submitted.

K_S The current session key

Change PIN Code:

The PIN code can be changed in the user stage with the command CHANGE PIN if the option bit PIN_ALT is set. My program a new PIN code in the card, the current PIN code must have been submitted first. For security reasons, the CHANGE PIN command can only be executed immediately after a Mutual Authentication process. No other command must have been executed between the Mutual Authentication and the CHANGE PIN command.

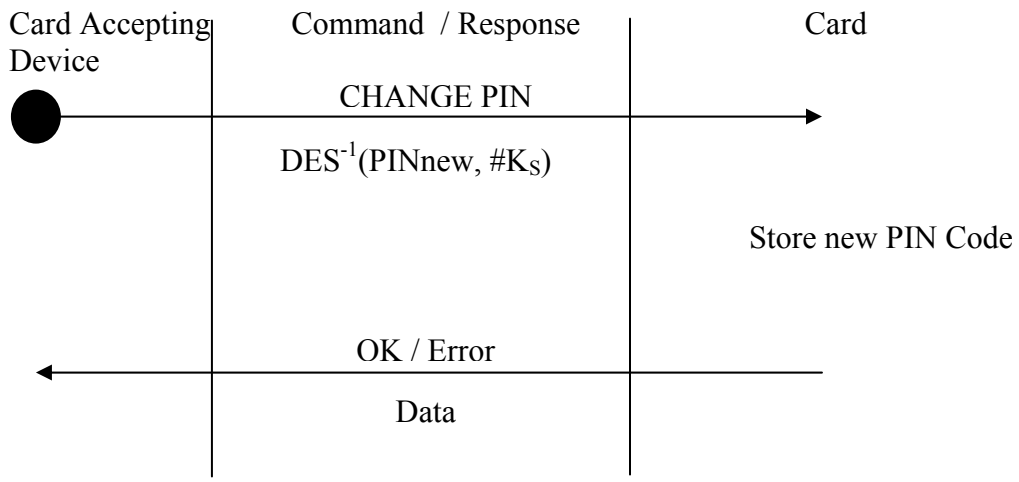


Figure 3.8 *Change PIN Code.*

PINnew : The new PIN code

K_S : The current session key

3.1.3. Development

Firstly, I must develop **mutual authentication**, **read / write file**, **account transaction processing** in EMV standard. Secondly, I must connect to smart card reader using hContext handle and obtain valid hCard handle, read and write data with APDU commands (sending data to smart card reader with PC/AC protocol.)

Mutual Authentication and Session Key based on Random Numbers:

The Mutual Authentication is based on the exchange and mutual verification of secret keys between the Card and the Card Accepting Device. The key exchange is performed in a secure way by use of random numbers and DES data encryption.

ACOS2 maintains a dedicated pair of data encryption/decryption keys for the Mutual Authentication, K_T, called Terminal Key, and K_C, called Card Key.

ACOS2 also provides a generator for the random numbers used in the Mutual Authentication process, RND_C, called Card Random Number. The session key is the final result of the Mutual Authentication process.

Account Transaction Processing: The account has four keys. Credit Key (K_{CR}), Debit Key (K_D), Certify Key (K_{CF}), Revoke Debit Key (K_{RD}). The keys are stored in the account security file. The keys are used in the calculation and verification of MAC cryptographic checksums on commands and data exchanged between the card and the Card Accepting Device in the Account processing. All keys are 8 bytes long. Debit Key, Credit Key and Revoke Debit Key have each associated an error counter CNT Kxx to count and limit the number of consecutive unsuccessful executions of the transaction commands.

Four different transaction types can be executed on the Account Data Structure under security conditions:

• INQUIRE ACCOUNT : The card returns the current balance value together with other relevant account information and a MAC cryptographic checksum on the relevant data.

• DEBIT : The balance in the Account is decreased by the specified amount

• REVOKE DEBIT : A REVOKE DEBIT is only possible after a DEBIT transaction and applies always to the immediately preceding DEBIT transaction.

• CREDIT: In a CREDIT transaction, the balance in the Account is increased by the specified amount.

The Account Data Structure can be read as a record oriented file in the Manufacturing Stage, in the Personalization Stage and in the User Stage after presentation of the Issuer Code IC. In the normal User Stage, a WRITE access to the Account is possible only through the special Account processing commands. WRITE RECORD access is possible after presentation of the Issuer Code IC.

3.1.4. Implementation

User File Management & Read/Write Record

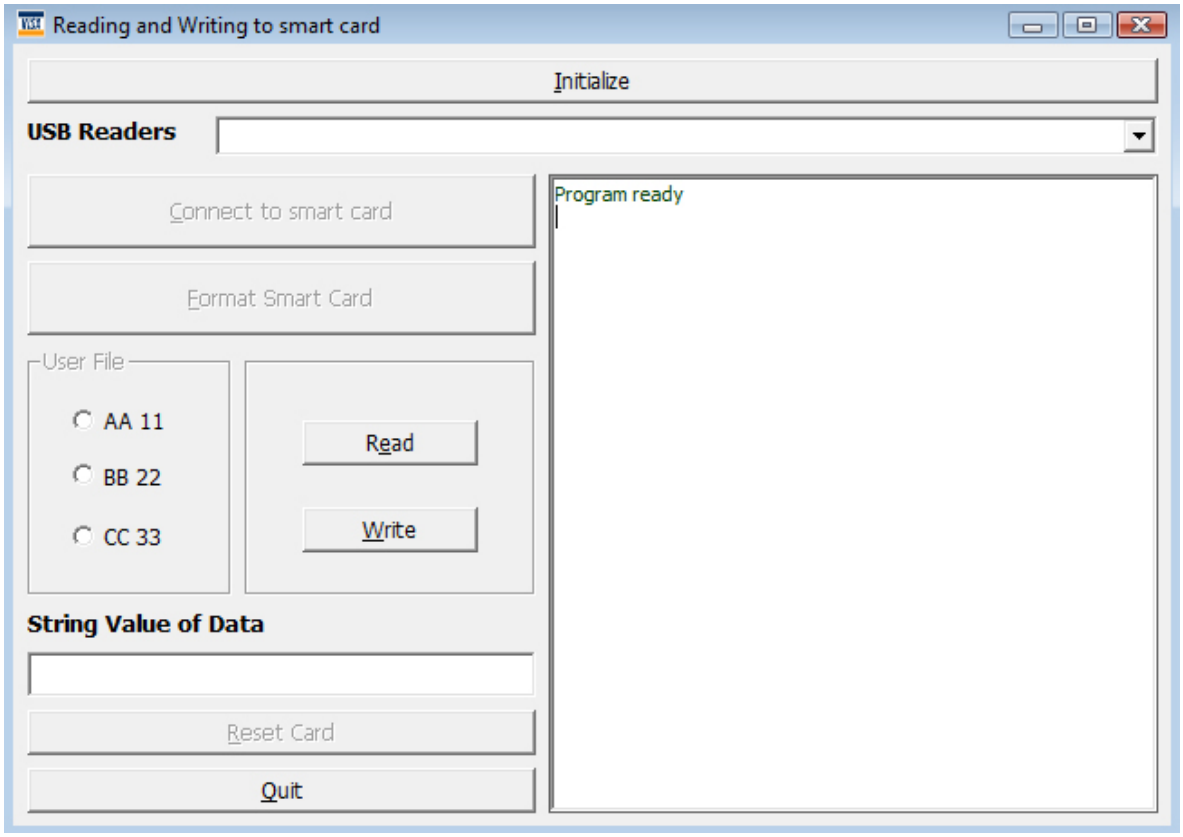


Figure 3.9 *Software Screenshot 1*

Clicking the "Initialize" button lists all USB card readers using PC/AC protocoll in a combo box. Then, we select the reader mounted to the smart card click on the "connect" button. After all the software connects to the smartcard. Clicking the " Format Card" button formats the card and AA11, BB22, CC33 user files are created for usage. The intended u-ser file (AA11, BB22, CC33) is to be choosen from option buttons and the string that is to be written in it is determined. When the string data in the textbox "String Value of Data" is entered with the keyboard and the write button is clicked the string value that is entered with keyboard is written to the selected file. Now if we choose a user file randomly and click on the read button, the string value will be read from the file and shown in the "String Value of Data" text box.

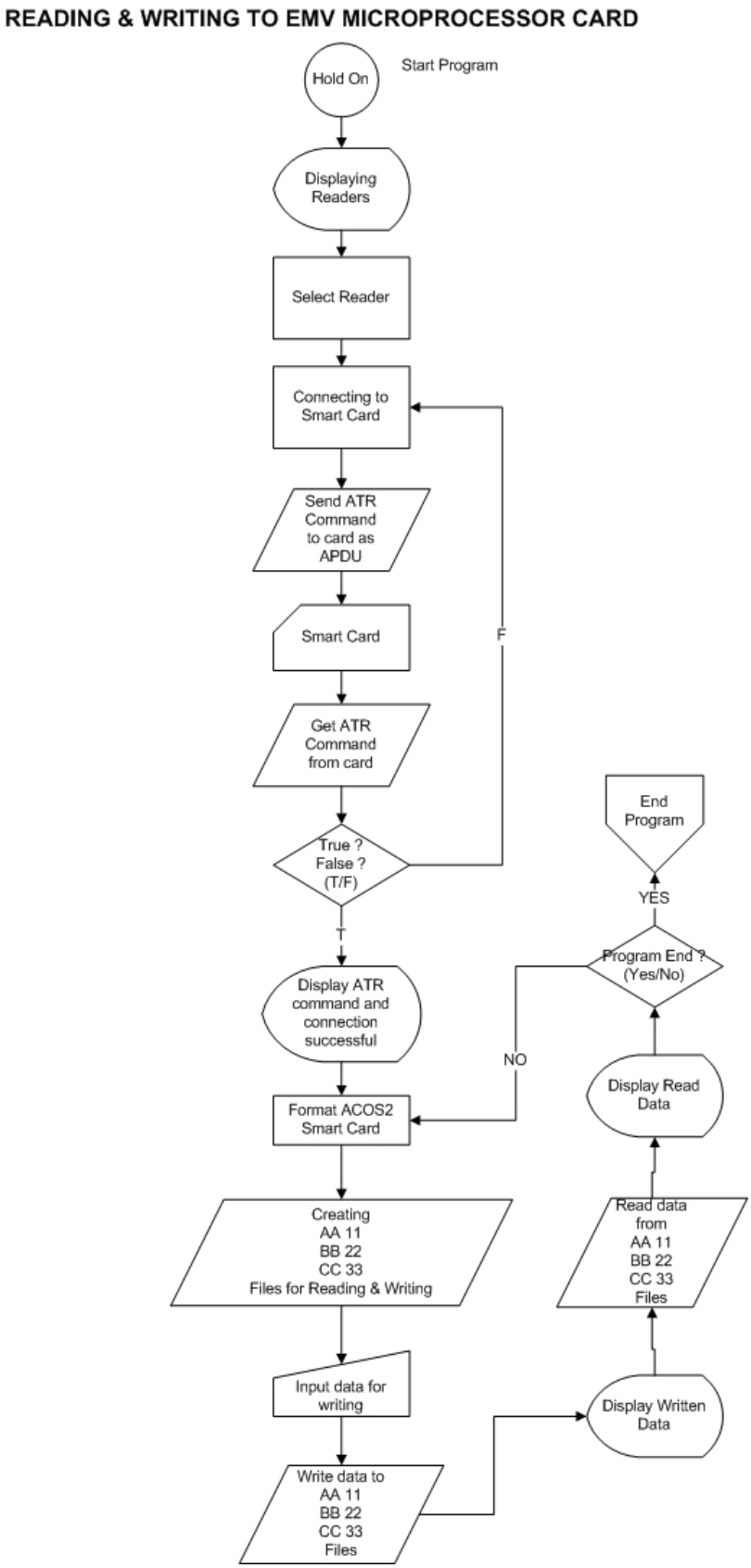


Figure 3.10 *Reading & Writing to EMV Microprocessor Card*

- Processes on Reading and Writing to EMV Microprocessor Card
- Program is in Ready/Standby mode
- Listing process of PC/AC protocole and USB readers
- Selection process of the USB reader
- Connecting to the smartcard
- Routing the APDV command to the card for the ATR command
- Obtainment of the ATR commend by the program
- Printing the ATR commend process to the screen if the comming ATR commend is true, if not the program goes back to the “Connecting to the Smartcard” process.
- Formatting process of the card with ACOS2 operating system.
- Creation of AA11, BB22, CC33 user files for reading and Writing process.
- Data enterance with keyboard for writing string data to the files.
- The process of writing data to the user files.
- Taping process of the data written in the file.
- Reading process of the string data from the intended User File
- If the program is resetted, end the program, if not you can format the ACOS2 card or it can wait in the standby mode.

MUTUAL AUTHENTICATION PROCESS

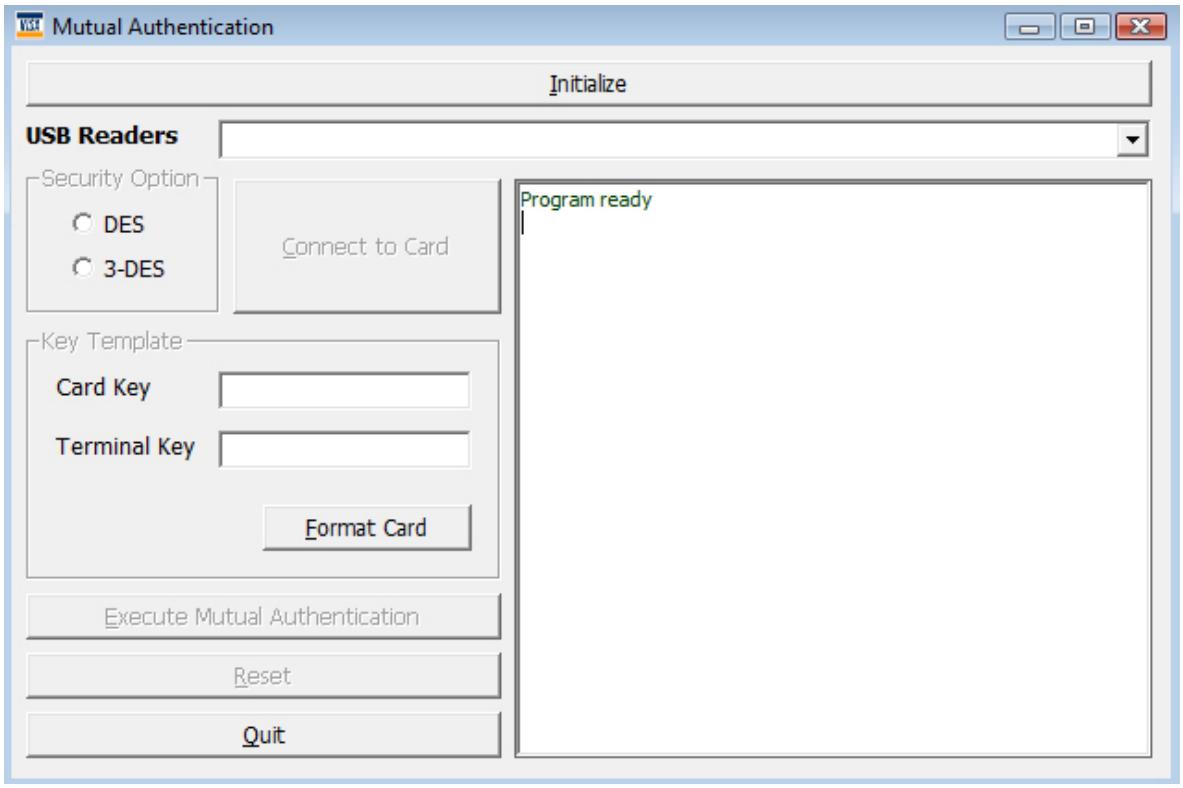


Figure 3.11*Software Screenshot 2*

Clicking the "Initialize" button lists all USB card readers using PC/AC protocoll in a combo box. Then, we select the reader mounted to the smart card click on the "connect" button. After all the software connects to the smartcard. After that, the cryipto algorithym (DES or 3DES) is selected from security option part. In the key template part the card key and terminal key is written and after all the "Format ACOS" button is clicked, so the card is formatted with the choosen cryipto algorithym. It must be used with entering the card key and terminal key, otherwise it will not work. Then, if we entered the right terminal key, by clicking on "Execute MA" button we are able to be logged in. Card – program connection is set whenn clicked on "Reset" button. The "Quit" button makes us quit the program.

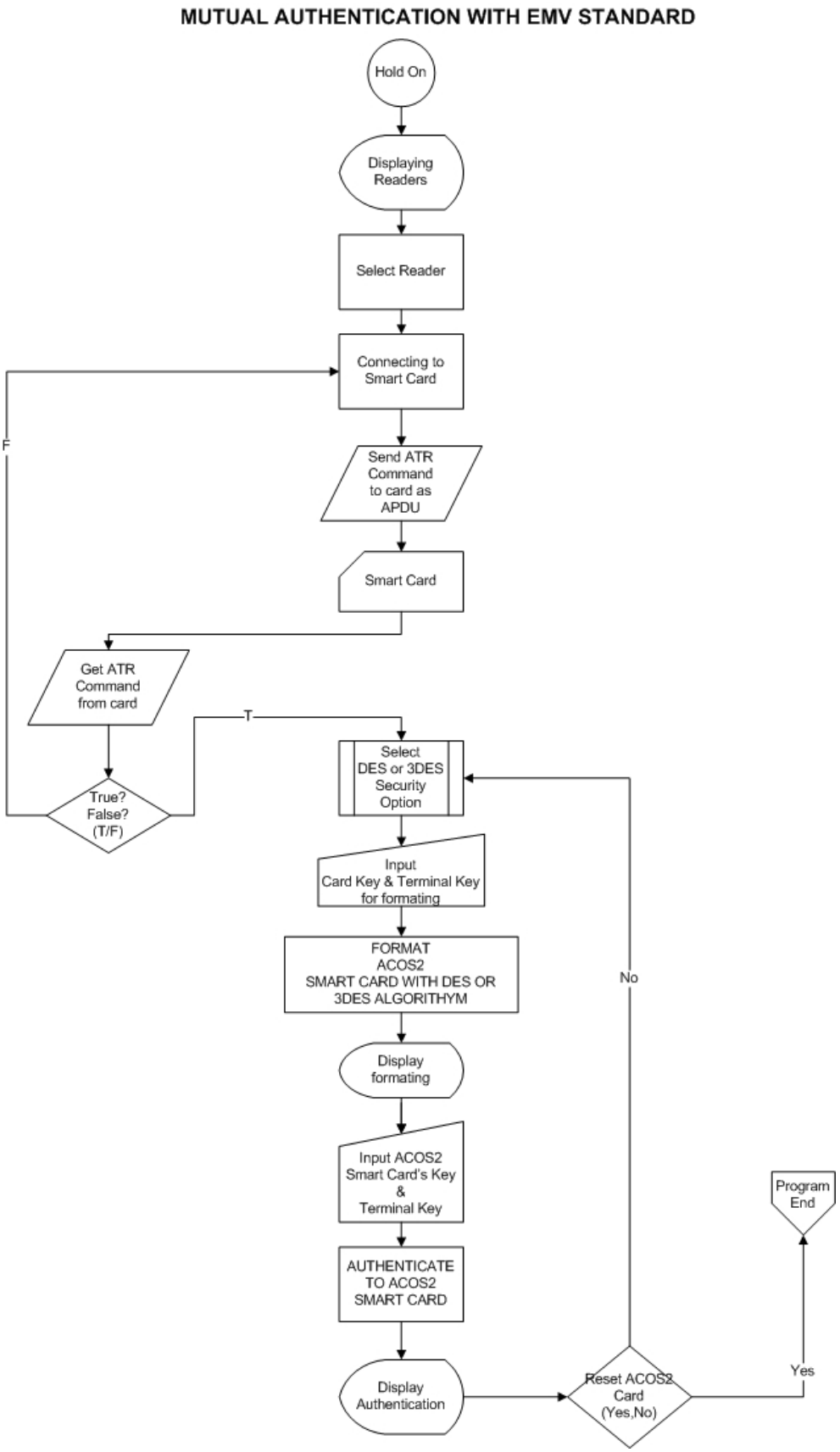


Figure 3.12 ***Mutual Authentication with EMV Standard***

- Program is in the stand-by mode.
- Listing of the USB readers working with PC/AC protocole in the screen.
- Selection of the readers
- Setting up the connection with the smartcard.
- Sending the APDU command to the card for the ATR number
- Getting the ATR number from the card
- If the ATR number is correct, the process begins as selection of the algorithyms DES or 3DES, if not turn back to the process of connection with the smartcard.
- Enterance of card and terminal keys with the keyboard.
- Formatting the card that is with ACOS2 operating system with cryipto algorithym DES or 3DES
- Execution of the formatting process
- Entering the card and terminal keys by the keyboard
- Mutual authentication to the card
- Showing the values on the program screen.
- If the reset button is clicked the connection between the card abd program is to be ended, if not the program is to be stand by in the idle mode for the formatting or authentication processes.

ACCOUNT TRANSACTION PROCESS

Clicking the “Initialize” button lists all card readers in a combo box. Then, we select the reader mounted to the smart card click on the “connect” button. After all, the software connects to the smartcard. After that, the cryipto algorithym (DES or 3DES) is selected from security option part. In the security keys part the credit key, debit key, certify key and revoke debit key is written and after all the “Format Card” button is clicked, so the card is formatted with the choosen cryipto algorithym.

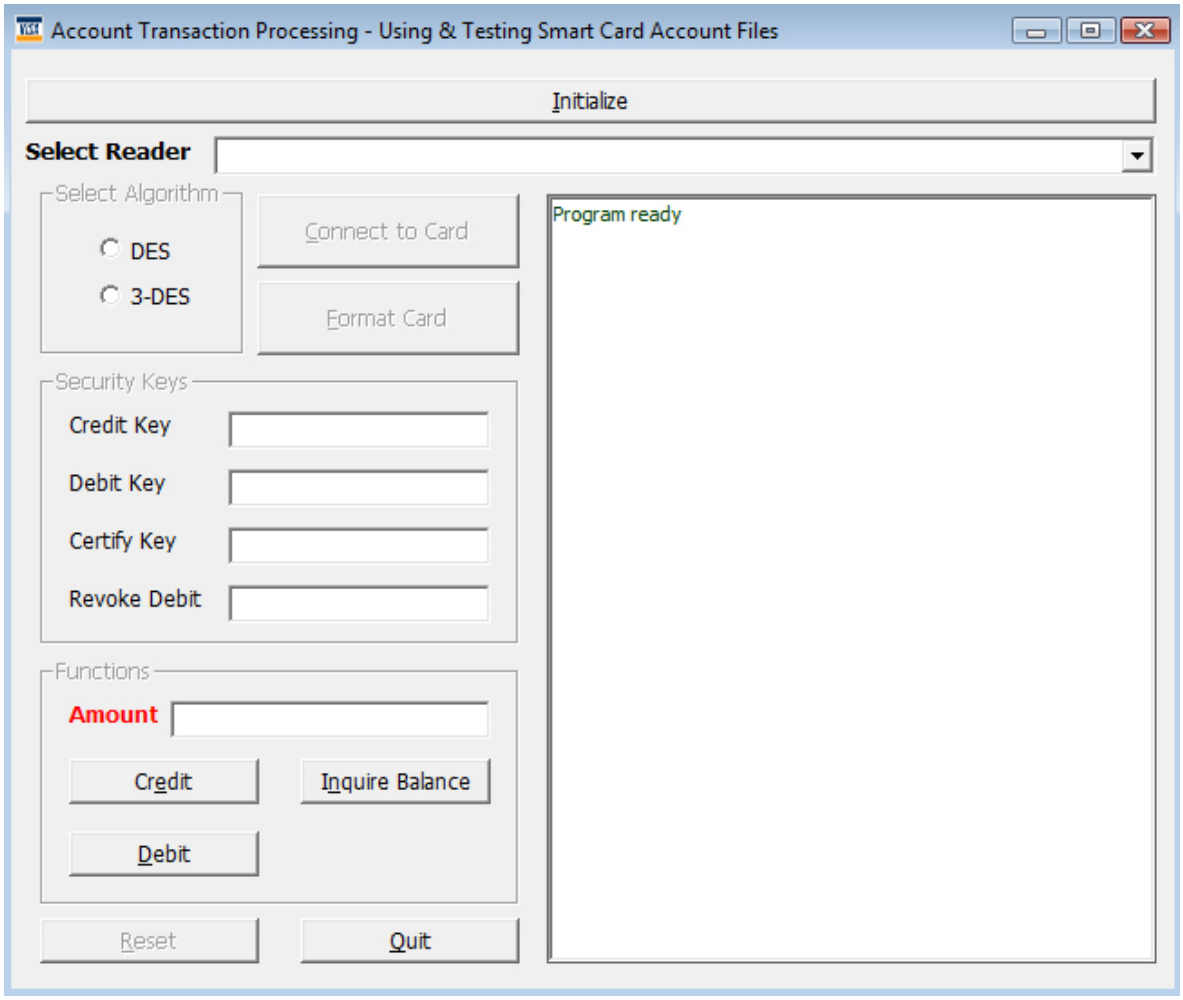


Figure 3.13 Software Screenshot 3

After the formatting process; the value that is wanted to be loaded to the card should be written and after all the "Credit" button should be clicked. So, the value that is written in the "Amount" textbox represents the balans of the card. Before loading balance to the card, we need to be sure that the security keys that were entered while formatting processes are written. They can withdraw money with the "Debit" button and with the "Inquire balance" button and the remaining balance in the card can be displayed.

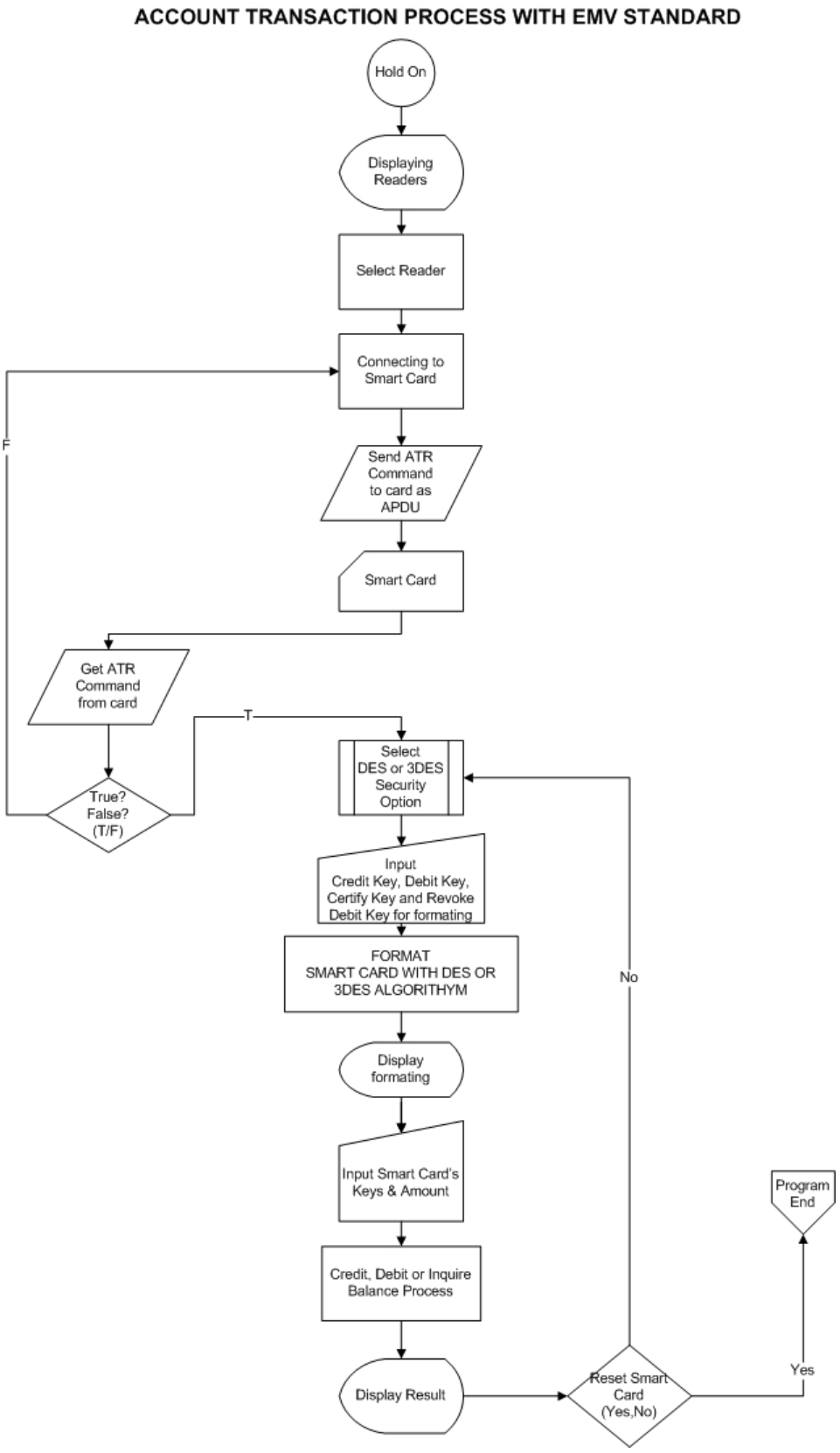


Figure 3.14 Account Transaction Process

- Program is in the stand-by mode.
- Listing of the USB readers working with PC/AC protocole in the screen.
- Selection of the readers
- Setting up the connection with the smartcard.
- Sending the APDU command to the card for the ATR number
- Getting the ATR number from the card
- If the ATR number is correct, the process begins as selection of the algorithyms DES or 3DES, if not turn back to the process of connection with the smartcard.
- Enterance of credit, debit, certify and revoke debit keys with the keyboard.
- Formatting the card with cryipto algorithym DES or 3DES
- Execution of the formatting process
- Entering the credit, debit, certify and revoke debit keys by the keyboard
- Mutual authentication to the card
- Showing the values on the program screen.
- If the reset button is clicked the connection between the card abd program is to be ended, if not the program is to be stand by in the idle mode for the formatting or authentication processes.

4. TEST RESULTS & FINDINGS

Table 4.1 *Test Values Table (ms)*

Test Results	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Connection	100	95	97	94	98	99	96	95	101	99
Login	100	90	92	91	93	98	101	95	99	97
Format DES	2000	1800	1870	1786	1883	1798	1766	1756	1743	1755
Format 3DES	2500	2600	2536	1591	1547	1596	1572	1589	1572	1584
Read File DES	1000	971	987	969	977	985	975	986	961	982
Write File DES	1300	1277	1294	1291	1269	1276	1265	1280	1269	1244
Read File 3DES	1300	1283	1288	1271	1255	1283	1271	1269	1277	1274
Write File 3DES	1800	1742	1763	1786	1792	1783	1776	1786	1785	1783

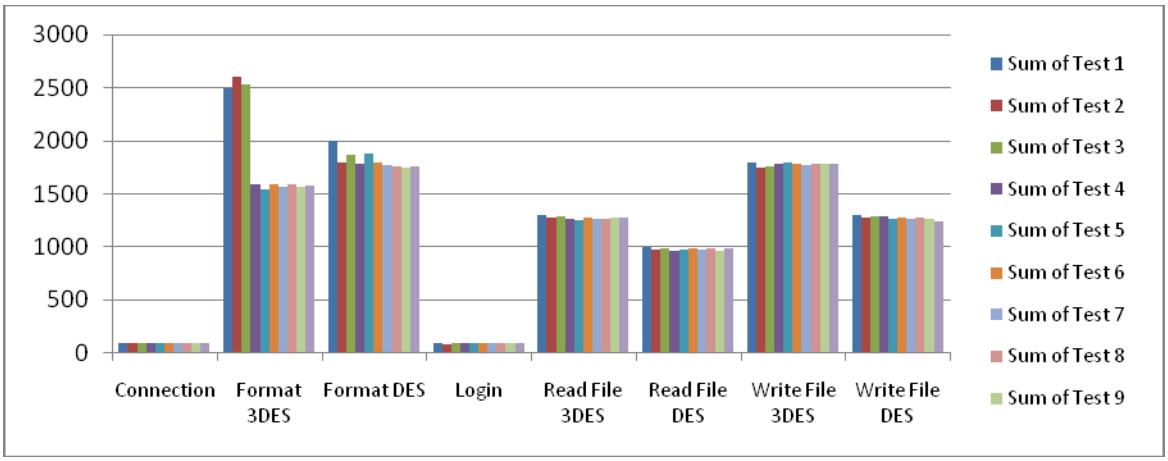


Figure 4.1 *Test Results*

Connection: Connecting to Card Reader.

Login: Logining to smart card sector(s).

Format (DES): Formating smart card sectors with DES algorithym.

Format (3DES): Formating smart card sectors with 3DES algorithym.

Read File DES: Reading File from smart card sector(s) with DES decription.

Write File DES: Writing file into smart card sectors with DES encryption.

Read File 3DES: Reading file from smard card sectors with 3DES decription.

Write File 3DES: Writing file into smart card sectors with 3DES encryption.

5. CONCLUSION & FUTURE WORKS

Smart card operating systems have improved security stepsand limitations. Access to every files and sectors in are depended on the commands and permissions of the operating system and data are written by crypto algorithms of data such as DES and 3DES to establish the PIN code, operating system that the smart card uses, VISA specification, file structure and sector structure must be known.

In addition to these, the file that the PIN code is written and access to that file must be known. The most important issue is the smart card format to be known. It is almost impossible to crack the PIN code without knowing the card format.

I will develop softwares for smart card personalization with crypto algorithms in EMV standards after completed my CARD SOFT Design & CARD SOFT Production softwares.

REFERENCES

Abelson H., Anderson R., Bellovin SM., Benaloh J., Blaze M., Diffei W., Gilmore J., Neumann P.G., Rivest R.L., Schiller J.I., and Schneier B. 1997. The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption. www.crypto.com.

Anderson RJ. 2001 Security Engineering. John Wiley & Sons.

Anderson RJ. and Needham RM. 1995 Programming satan's computer, Computer Science Today. www.computersciencetoday.com

Anderson, R. Why cryptosystems fail. Communications of the ACM, 37(11), Nov. 1994.

Anderson, R. M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic processors – a survey. Proceedings of the IEEE,94(2), Feb. 2006. Invited paper.

Ausfuhrliste (Export Control List) 2004 Ausfuhrliste: Anlage AL zur Außenwirtschaftsverordnung, 25 May 2004.Bundesamt f¨ur Wirtschaft und Ausfuhrkontrolle.

AWG 2004 Außenwirtschaftsgesetz (Foreign Trade Act), 23 December 2004.

AWV 2001 Außenwirtschaftsverordnung (Foreign Trade Ordinance), 2 July 2001.

Bond, M. and Zielinski, P. Decimalisation table attacks for PIN cracking. Technical report (UCAM-CL-TR-560), Computer Laboratory, University of Cambridge, 2003.

Bond, M. and Zielinski, P. Encrypted? Randomised? Compromised? (When cryptographically secured data is not secure). In Workshop on Cryptographic Algorithms and their Uses, Gold Coast, Australia, July 2004.

Biham, E. and Shamir, A. Differential Cryptanalysis of DES-like Cryptosystems. In A. J. Menezes and S. A. Vanstone, editors, Advances in Cryptology — CRYPTO '90, volume LNCS 537, pages 2–21, Berlin, Germany, 1991. Springer-Verlag.

BAFA – Bundesamt fur Wirtschaft und Ausfuhrkontrolle (Federal Office of Economics and Export Control). www.bafa.de.

BDSG 2001 Bundesdatenschutzgesetz (Federal Data Protection Act), 11 May 2001. BfD – Bundesbeauftragte f¨ur den Datenschutz (Federal Commissioner for Data Protection). www.bfd.bund.de.

Berkman, O. and Ostrovsky, M. The unbearable lightness of PIN cracking. In Financial Cryptography and Data Security (FC), Scarborough, Trinidad and Tobago, Feb. 2007.

BNA – Bundesnetzagentur (Federal Network Agency). www.bundesnetzagentur.de.

Boehm BW. 1981 Software Engineering Economics.Prentice Hall.

Bond, M. Phantom withdrawals: On-line resources for victims of ATM fraud. http://www.phantomwithdrawals.com.

Bond, M. Understanding security APIs. Ph.D. Thesis, Computer Laboratory, University of Cambridge, 2004.

Bond, M. Attacks on cryptoprocessor transaction sets. In Workshop on Cryptographic Hardware and Embedded Systems (CHES), Paris, France, May 2001.

Bono S, Green M, Stubblefield A, Juels A, Rubin A, and Szydlo M 2005 Security Analysis of a Cryptographically-Enabled RFID Device. The Johns Hopkins
University Information Security Institute, Baltimore.

BSI – Bundesamt f¨ur Sicherheit in der Informationstechnik(Federal Office for Information Security). www.bsi.de.

Chen Z. 2000 Java Card Technology for Smartcards.Addison Wesley.

Clulow, J. The design and analysis of cryptographic APIs for security devices. Masters Thesis, University of Natal, Durban, South Africa, 2003.

CLUSIF 2002. An Overview of Cyber-Crime in 2001. Club de la S´ecurit´e des Syst´emes d'Information Franc¸ais, Paris.

Criteria 2005 Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung ('Notice Regarding Electronic Signatures Compliant with the Signature Act and the Signature Ordinance') (summary of suitable algorithms). Bundesnetzagentur (Federal Network Agency)

Crypto 2002 Cryptography and Liberty: An International Survey of Encryption Policy. Electronic Privacy Information Center, Washington, DC.

CWA 2004 Application Interface for Smartcards used as Secure Signature Creation Devices. CWA 14890.

Datenschutz (Data Protection) 1996 Anforderungen zur informationstechnischen Sicherheit bei Chipkarten ('Requirements for Information Security with Regard to Chip Cards'). The Data Protection Commissioner of Hamburg, Hamburg.

Drimer, D. Murdoch, S. and Anderson, R. Thinking inside the box: System-level failures of tamper proofing. In IEEE Symposium on Security and Privacy (to appear), May 2008. Also avialable as a technical report (UCAM-CL-TR-711) at http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-711.html

ECR 2000 Council Regulation (EC) No 1334/2000 of 22 June 2000 setting up a Community regime for the control of exports of dual-use items and technology.

Finkenzeller F. 1999 RFID Handbook. John Wiley & Sons.

Gamma E, Helm R, and Johnson RE 1994 Design Patterns Elements of Reusable Object-Oriented Software. Addison-Wesley.

Garstka H 2003 Informationelle Selbstbestimmung und Datenschutz, in Schulzki-Haddouti C B¨urgerrechte im Netz. Bundeszentrale f¨ur politische Bildung, Bonn.

Global Platform 2003 Open Platform: Card Specification, Version 2.1.1. www.globalplatform.org.
Haghiri Y and Tarantino T 2002 Smartcard Manufacturing: A Practical Guide. John Wiley & Sons.

Holloway, R. University of London - Information Security Group. September 2006. MSc in Information Security Smart Card Centre Laboratory - A Software Implementation of AES for a Multos Smart Card.

Hassler V., Manninger M, Gordeev M, and Muller M 2002 Java Card for E-Payment Applications. Artech House, London.

Hunt A. and Thomas D. 1999 The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley.

Horster P. and Fox D. (ed.) 1999 Datenschutz und Datensicherheit. Vieweg Verlag, Braunschweig.

ITGH 2004 IT-Grundschutzhandbuch. Bundesanzeiger-Verlag, Cologne.

ITU X.509:2000 Information Technology: Open Systems Interconnection: The Directory Authentication Framework. www.itu.int.

Lamport L 1981 Password authentication with insecure communication, Communications of the ACM 24, 11. ACM (Association for Computing Machinery), San Diego, CA.

Liggesmeyer P 2002 Software-Qualit¨at. Spektrum Verlag, Heidelberg.

McConnell S 2002 Code Complete, 2nd edn. Barnes & Noble.

Menezes AJ, van Oorschot PC, and Vanstone SA 1997 Handbook of Applied Cryptography. CRC Press, Boca Raton, FL.

Mannan, M. and Oorschot, P. Using a personal device to strengthen password authentication from an untrusted computer. In Financial Cryptography and Data Security (FC), Scarborough, Trinidad and Tobago, Feb. 2007.

Nirmalananthan, Anusha. Microsoft Corporation. October 2002.
Smart Card Technical Articles - The Smart Card Cryptographic Service Provider Cookbook

Poschmann, A., Leander, G., Schramm, K. and Paar, C. 2006. A Family of Light-Weight Block Ciphers Based on DES Suited for RFID Applications.

Poschmann, A., Leander, G., Schramm,K. and Paar, C. Horst G¨ortz Institute for IT-Security, Ruhr-University Bochum, Germany. 2007. New Light-Weight Crypto Algorithms for RFID

Ross, B. Jackson, R. Miyake, B. Boneh, D. and Mitchell, J.C. Stronger password authentication using browser extensions. In USENIX Security, 2005.

Schaar P. 2002 Datenschutz im Internet. C.H. Beck, Munich.

Schneier B. 1996 Angewandte Kryptographie. John Wiley & Sons.

Spillner A. and Linz T. 2003 Basiswissen Softwaretest. Dpunkt Verlag, Heidelberg.

Ostrovsky, O.V. Vulnerabilities in the financial PIN processing API. Masters Thesis, Tel Aviv University, 2006.

EMV Book 1 2004 EMV Integrated Circuit Card Specification for Payment Systems, Book 1: Application Independent ICC to Terminal Interface Requirements, Version 4.1. www.emvco.com.

EMV Book 2 2004 EMV Integrated Circuit Card Specification for Payment Systems, Book 2: Security and Key Management, Version 4.1. www.emvco.com.

EMV Book 3 2004 EMV Integrated Circuit Card Specification for Payment Systems, Book 3: Application Specification, Version 4.1. www.emvco.com.

EMV Book 4 2004 EMV Integrated Circuit Card Specification for Payment Systems, Book 4: Cardholder, Attendant and Acquirer Interface Requirements, Version 4.1. www.emvco.com.

EN 1546:2000 Identification Card Systems: Inter-Sector Electronic Purse.

ETSI – European Telecommunications Standards Institute. www.etsi.org.

EU 1995 Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of Such Data. Official Journal of the European Communities, L. 281, 23 November.

JCAPN 2003 Java Card Platform: Application Programming Notes, Version 2.2.1, Sun Microsystems, Santa Clara, CA.

JCAPI 2003 Java Card Platform: Application Programming Interface, Version 2.2.1, Sun Microsystems, Santa Clara, CA.

JCRES 2003 Java Card Platform: Runtime Environment Specification, Version 2.2.1, Sun Microsystems, Santa Clara, CA.

JCVMS 2003 Java Card Platform: Virtual Machine Specification, Version 2.2.1, Sun Microsystems, Santa Clara, CA.
ICAO 2002 ICAO Machine Readable Travel Documents, Part 3: Size 1 and Size 2 Machine Readable Official Travel Documents, 2nd edn, Doc 9303. www.icao.int.

ISO/IEC 7810:2003 Identification Cards: Physical Characteristics.

ISO/IEC 7811-1:2002 Identification Cards: Recording Technique: Part 1 Embossing.

ISO/IEC 7811-2:2001 Identification Cards: Recording Technique: Part 2 Magnetic Stripe: Low Coercivity.

ISO/IEC 7813:2001 Identification Cards: Financial Transaction Cards.

ISO/IEC 7816-3:1997 Identification Cards: Integrated Circuit(s) Cards: Part 3 Cards with Contacts: Electrical Interface and Transmission Protocols.

ISO/IEC 7816-4:2005 Identification Cards: Integrated Circuit Cards: Part 4 Organization, Security and Commands for Interchange.
ISO/IEC 7816-6:2004 Identification Cards: Integrated Circuit Cards: Part 6 Interindustry Data Elements for Interchange.

ISO/IEC 7816-8:2004 Identification Cards: Integrated Circuit Cards: Part 8 Commands for Security Operations.

ISO/IEC 7816-9:2004 Identification Cards: Integrated Circuit Cards: Part 9 Commands for Card Management.

ISO/IEC 7816-12:2005 Identification Cards: Integrated Circuit Cards: Part 12 Cards with Contacts: USB Electrical Interface and Operating Procedures.

ISO/IEC 7816-15:2004 Identification Cards: Integrated Circuit Cards: Part 15 Cryptographic Information Application.

ISO 8402: 1994 Quality Management and Quality Assurance: Vocabulary.

ISO/IEC 8824: 2002 Information Technology: Abstract Syntax Notation One (ASN.1).

ISO/IEC 8825: 2002 Information Technology: ASN.1 Encoding Rules Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

ISO/IEC 14443-4:2001 Identification Cards: Contactless Integrated Circuit(s) Cards: Proximity Cards.

PCSC 2004 PC/SC Interoperability Specification for ICCs and Personal Computer Systems, V 2.00.1
1. www.smartcardsys.com.

PKCS #15 2000 Cryptographic Token Information Format Standard, V 1.1.
www.rsa.com.

Rankl W and Effing W 2002 Smartcard Handbook, 3rd edn. John Wiley & Sons.
RFC 2289 1998 A One-Time Password System.

SATSA 2004 Security and Trust Services API for Java 2 Platform Micro Edition Java Community Process (JCP), Version 1.0. jcp.org.

Open Card 2001 Open Platform Card Specification, Version 2.1, Open Card Foundation.

TS 101 476:2002 Digital cellular telecommunications system (Phase 2+): Subscriber Identity Module Application Programming Interface (SIM API): SIM API for Java CardTM:; Stage 2, V8.5.0. ETSI.

TS 102 221:2005 Smartcards: UICC–Terminal interface: Physical and logical characteristics, Release 6, V6.8.0. ETSI.

TS 102 222:2005 Integrated Circuit Cards (ICC): Administrative commands for telecommunications applications, Release 6, V6.8.0. ETSI.

TS 31.102:2003 3rd Generation Partnership Project: Technical Specification Group Terminals: Characteristics of the USIM application, Release 6, V6.4.0. 3GPP.

TS 51.011:2003 3rd Generation Partnership Project; Technical Specification Group Terminals; Specification of the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface, Release 4, V4.2.0. 3GPP.

TS 51.014:2003 3rd Generation Partnership Project: Technical Specification Group Terminals: Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface, Release 4, V4.3.0. 3GPP.

V Model XT 2004 V-Modell XT. Federal Republic of Germany, www.v-modell-xt.de.

Wassenaar Arrangement: List of Dual Use Goods and Technologies And Munitions List. Vienna. 2004. www.wassenaar.org.

Algorithmic Research (ARX). PrivateServer Switch-HSM. White paper. http://www.arx.com/documents/Switch-HSM.pdf

International Organization for Standardization (ISO). Banking – Personal Identification Number (PIN) management and security – Part 1: Basic principles and requirements for online PIN handling in ATM and POS systems, Apr. 2002. International Standard, ISO 9564-1.

EMVCo, LLC ("EMVCo"). June 2008. EMV Integrated Circuit Card Specifications for Payment Systems – Book 1 - Application Independent ICC to Terminal Interface Requirements Version 4.2

EMVCo, LLC ("EMVCo"). June 2008. EMV Integrated Circuit Card Specifications for Payment Systems – Book 2 – Security and Key Management Version 4.2

EMVCo, LLC ("EMVCo"). June 2008. EMV Integrated Circuit Card Specifications for Payment Systems – Book 3 – Application Specification Version 4.2

EMVCo, LLC ("EMVCo"). June 2008. EMV Integrated Circuit Card Specifications for Payment Systems – Book 4 – Cardholder, Attendant, and Acquirer Interface Requirements Version 4.2

EMVCo, LLC ("EMVCo"). July 2007. EMV Card Personalization Specification Version 1.1

APPENDICES

Appendix I – Creating User Files & Read/Write string data to EMV smart card.

Windscard.dll (windscard module)

```
Public Type SCARD_IO_REQUEST
   dwProtocol As Long
   cbPciLength As Long
End Type

Public Type APDURec
   bCLA As Byte
   bINS As Byte
   bP1 As Byte
   bP2 As Byte
   bP3 As Byte
'   DATA(1 To 255) As Byte
   DataIn(1 To 255) As Byte
   DataOut(1 To 255) As Byte
   SW(1 To 2) As Byte
   IsSend As Boolean
End Type

Public Type SCARD_READERSTATE
   RdrName As String
   UserData As Long
   RdrCurrState As Long
   RdrEventState As Long
   ATRLength As Long
   ATRValue(1 To 36) As Byte
End Type


Global Const SCARD_S_SUCCESS = 0
Global Const SCARD_ATR_LENGTH = 33

'============================================================
'  Memory Card type constants
'============================================================

Global Const CT_MCU = &H0              ' MCU
Global Const CT_IIC_Auto = &H1          ' IIC (Auto Detect Memory Size)
Global Const CT_IIC_1K = &H2            ' IIC (1K)
Global Const CT_IIC_2K = &H3            ' IIC (2K)
Global Const CT_IIC_4K = &H4            ' IIC (4K)
```

```
Global Const CT_IIC_8K = &H5              ' IIC (8K)
Global Const CT_IIC_16K = &H6             ' IIC (16K)
Global Const CT_IIC_32K = &H7             ' IIC (32K)
Global Const CT_IIC_64K = &H8             ' IIC (64K)
Global Const CT_IIC_128K = &H9            ' IIC (128K)
Global Const CT_IIC_256K = &HA            ' IIC (256K)
Global Const CT_IIC_512K = &HB            ' IIC (512K)
Global Const CT_IIC_1024K = &HC           ' IIC (1024K)
Global Const CT_AT88SC153 = &HD           ' AT88SC153
Global Const CT_AT88SC1608 = &HE          ' AT88SC1608
Global Const CT_SLE4418 = &HF             ' SLE4418
Global Const CT_SLE4428 = &H10            ' SLE4428
Global Const CT_SLE4432 = &H11            ' SLE4432
Global Const CT_SLE4442 = &H12            ' SLE4442
Global Const CT_SLE4406 = &H13            ' SLE4406
Global Const CT_SLE4436 = &H14            ' SLE4436
Global Const CT_SLE5536 = &H15            ' SLE5536
Global Const CT_MCUT0 = &H16              ' MCU T=0
Global Const CT_MCUT1 = &H17              ' MCU T=1
Global Const CT_MCU_Auto = &H18           ' MCU Autodetect

'=============================================================
' Context Scope
'=============================================================
Global Const SCARD_SCOPE_USER = 0 ' The context is a user context, and any
                        ' database operations are performed within the
                        ' domain of the user.
Global Const SCARD_SCOPE_TERMINAL = 1 ' The context is that of the current
terminal,
                          ' and any database operations are performed
                          ' within the domain of that terminal.  (The
                          ' calling application must have appropriate
                          ' access permissions for any database actions.)
Global Const SCARD_SCOPE_SYSTEM = 2 ' The context is the system context, and any
                        ' database operations are performed within the
                        ' domain of the system.  (The calling
                        ' application must have appropriate access
                        ' permissions for any database actions.)

'=============================================================
==========
' Context Scope
'=============================================================
==========
Global Const SCARD_STATE_UNAWARE = &H0 ' The application is unaware of the
                              ' current state, and would like to
                              ' know.  The use of this value
```

```
                        ' results in an immediate return
                        ' from state transition monitoring
                        ' services.  This is represented by
                        ' all bits set to zero.
Global Const SCARD_STATE_IGNORE = &H1 ' The application requested that
                        ' this reader be ignored.  No other
                        ' bits will be set.
Global Const SCARD_STATE_CHANGED = &H2 ' This implies that there is a
                        ' difference between the state
                        ' believed by the application, and
                        ' the state known by the Service
                        ' Manager.  When this bit is set,
                        ' the application may assume a
                        ' significant state change has
                        ' occurred on this reader.

Global Const SCARD_STATE_UNKNOWN = &H4 ' This implies that the given
                        ' reader name is not recognized by
                        ' the Service Manager.  If this bit
                        ' is set, then SCARD_STATE_CHANGED
                        ' and SCARD_STATE_IGNORE will also
                        ' be set.
Global Const SCARD_STATE_UNAVAILABLE = &H8 ' This implies that the actual
                        ' state of this reader is not
                        ' available.  If this bit is set,
                        ' then all the following bits are
                        ' clear.
Global Const SCARD_STATE_EMPTY = &H10 ' This implies that there is not
                        ' card in the reader.  If this bit
                        ' is set, all the following bits
                        ' will be clear.
Global Const SCARD_STATE_PRESENT = &H20 ' This implies that there is a card
                        ' in the reader.
Global Const SCARD_STATE_ATRMATCH = &H40 ' This implies that there is a card
                        ' in the reader with an ATR
                        ' matching one of the target cards.
                        ' If this bit is set,
                        ' SCARD_STATE_PRESENT will also be
                        ' set.  This bit is only returned
                        ' on the SCardLocateCard() service.
Global Const SCARD_STATE_EXCLUSIVE = &H80 ' This implies that the card in the
                        ' reader is allocated for exclusive
                        ' use by another application.  If
                        ' this bit is set,
                        ' SCARD_STATE_PRESENT will also be
                        ' set.
Global Const SCARD_STATE_INUSE = &H100 ' This implies that the card in the
```

```
                        ' reader is in use by one or more
                        ' other applications, but may be
                        ' connected to in shared mode.  If
                        ' this bit is set,
                        ' SCARD_STATE_PRESENT will also be
                        ' set.
Global Const SCARD_STATE_MUTE = &H200 ' This implies that the card in the
                        ' reader is unresponsive or not
                        ' supported by the reader or
                        ' software.
Global Const SCARD_STATE_UNPOWERED = &H400 ' This implies that the card in the
                        ' reader has not been powered up.


'=============================================================
Global Const SCARD_SHARE_EXCLUSIVE = 1 ' This application is not willing to share
this
                ' card with other applications.
Global Const SCARD_SHARE_SHARED = 2 ' This application is willing to share this
                ' card with other applications.
Global Const SCARD_SHARE_DIRECT = 3 ' This application demands direct control of
                ' the reader, so it is not available to other
                ' applications.

'=============================================================
'   Disposition
'=============================================================
Global Const SCARD_LEAVE_CARD = 0 ' Don't do anything special on close
Global Const SCARD_RESET_CARD = 1 ' Reset the card on close
Global Const SCARD_UNPOWER_CARD = 2 ' Power down the card on close
Global Const SCARD_EJECT_CARD = 3 ' Eject the card on close

'=============================================================
'   Error Codes
'=============================================================
Global Const SCARD_F_INTERNAL_ERROR = &H80100001
Global Const SCARD_E_CANCELLED = &H80100002
Global Const SCARD_E_INVALID_HANDLE = &H80100003
Global Const SCARD_E_INVALID_PARAMETER = &H80100004
Global Const SCARD_E_INVALID_TARGET = &H80100005
Global Const SCARD_E_NO_MEMORY = &H80100006
Global Const SCARD_F_WAITED_TOO_LONG = &H80100007
Global Const SCARD_E_INSUFFICIENT_BUFFER = &H80100008
Global Const SCARD_E_UNKNOWN_READER = &H80100009
Global Const SCARD_E_TIMEOUT = &H8010000A
Global Const SCARD_E_SHARING_VIOLATION = &H8010000B
Global Const SCARD_E_NO_SMARTCARD = &H8010000C
```

```
Global Const SCARD_E_UNKNOWN_CARD = &H8010000D
Global Const SCARD_E_CANT_DISPOSE = &H8010000E
Global Const SCARD_E_PROTO_MISMATCH = &H8010000F
Global Const SCARD_E_NOT_READY = &H80100010
Global Const SCARD_E_INVALID_VALUE = &H80100011
Global Const SCARD_E_SYSTEM_CANCELLED = &H80100012
Global Const SCARD_F_COMM_ERROR = &H80100013
Global Const SCARD_F_UNKNOWN_ERROR = &H80100014
Global Const SCARD_E_INVALID_ATR = &H80100015
Global Const SCARD_E_NOT_TRANSACTED = &H80100016
Global Const SCARD_E_READER_UNAVAILABLE = &H80100017
Global Const SCARD_P_SHUTDOWN = &H80100018
Global Const SCARD_E_PCI_TOO_SMALL = &H80100019
Global Const SCARD_E_READER_UNSUPPORTED = &H8010001A
Global Const SCARD_E_DUPLICATE_READER = &H8010001B
Global Const SCARD_E_CARD_UNSUPPORTED = &H8010001C
Global Const SCARD_E_NO_SERVICE = &H8010001D
Global Const SCARD_E_SERVICE_STOPPED = &H8010001E
Global Const SCARD_W_UNSUPPORTED_CARD = &H80100065
Global Const SCARD_W_UNRESPONSIVE_CARD = &H80100066
Global Const SCARD_W_UNPOWERED_CARD = &H80100067
Global Const SCARD_W_RESET_CARD = &H80100068
Global Const SCARD_W_REMOVED_CARD = &H80100069

'============================================================
'   Protocol
'============================================================
Global Const SCARD_PROTOCOL_UNDEFINED = &H0          ' There is no active
protocol.
Global Const SCARD_PROTOCOL_T0 = &H1                 ' T=0 is the active protocol.
Global Const SCARD_PROTOCOL_T1 = &H2                 ' T=1 is the active protocol.
Global Const SCARD_PROTOCOL_RAW = &H10000            ' Raw is the active
protocol.
Global Const SCARD_PROTOCOL_DEFAULT = &H80000000     ' Use implicit PTS.

'============================================================
'   Reader State
'============================================================
Global Const SCARD_UNKNOWN = 0    ' This value implies the driver is unaware
                        ' of the current state of the reader.
Global Const SCARD_ABSENT = 1     ' This value implies there is no card in
                        ' the reader.
Global Const SCARD_PRESENT = 2    ' This value implies there is a card is
                        ' present in the reader, but that it has
                        ' not been moved into position for use.
Global Const SCARD_SWALLOWED = 3  ' This value implies there is a card in the
                        ' reader in position for use.  The card is
```

```
                    ' not powered.
Global Const SCARD_POWERED = 4    ' This value implies there is power is
                    ' being provided to the card, but the
                    ' Reader Driver is unaware of the mode of
                    ' the card.
Global Const SCARD_NEGOTIABLE = 5 ' This value implies the card has been
                    ' reset and is awaiting PTS negotiation.
Global Const SCARD_SPECIFIC = 6   ' This value implies the card has been
                    ' reset and specific communication
                    ' protocols have been established.


'=====================================================================
' Prototypes
'=====================================================================
Public Declare Function SCardEstablishContext Lib "Winscard.dll" (ByVal dwScope As
Long, _
                                        ByVal pvReserved1 As Long, _
                                        ByVal pvReserved2 As Long, _
                                        ByRef phContext As Long) As Long

Public Declare Function SCardReleaseContext Lib "Winscard.dll" (ByVal hContext As
Long) As Long

Public Declare Function SCardConnect Lib "Winscard.dll" Alias "SCardConnectA"
(ByVal hContext As Long, _
                                            ByVal szReaderName As String, _
                                            ByVal dwShareMode As Long, _
                                            ByVal dwPrefProtocol As Long, _
                                            ByRef hCard As Long, _
                                            ByRef ActiveProtocol As Long) As Long

Public Declare Function SCardDisconnect Lib "Winscard.dll" (ByVal hCard As Long, _
                                    ByVal Disposistion As Long) As Long

Public Declare Function SCardBeginTransaction Lib "Winscard.dll" (ByVal hCard As
Long) As Long

Public Declare Function SCardEndTransaction Lib "Winscard.dll" (ByVal hCard As Long,
_
                                        ByVal Disposition As Long) As Long

Public Declare Function SCardState Lib "Winscard.dll" (ByVal hCard As Long, _
                                    ByRef State As Long, _
                                    ByRef Protocol As Long, _
                                    ByRef ATR As Byte, _
                                    ByRef ATRLen As Long) As Long
```

```
Public Declare Function SCardStatus Lib "Winscard.dll" Alias "SCardStatusA" (ByVal
hCard As Long, _
                                                ByVal szReaderName As String, _
                                                ByRef pcchReaderLen As Long, _
                                                ByRef State As Long, _
                                                ByRef Protocol As Long, _
                                                ByRef ATR As Byte, _
                                                ByRef ATRLen As Long) As Long

Public Declare Function SCardTransmit Lib "Winscard.dll" (ByVal hCard As Long, _
                                        pioSendRequest As SCARD_IO_REQUEST, _
                                        ByRef SendBuff As Byte, _
                                        ByVal SendBuffLen As Long, _
                                        ByRef pioRecvRequest As SCARD_IO_REQUEST, _
                                        ByRef RecvBuff As Byte, _
                                        ByRef RecvBuffLen As Long) As Long

Public Declare Function SCardListReaders Lib "Winscard.dll" Alias "SCardListReadersA"
(ByVal hContext As Long, _
                                         ByVal mzGroup As String, _
                                         ByVal ReaderList As String, _
                                         ByRef pcchReaders As Long) As Long

Public Declare Function SCardGetStatusChange Lib "Winscard.dll" Alias
"SCardGetStatusChangeA" (ByVal hContext As Long, _
                                        ByVal TimeOut As Long, _
                                        ByRef ReaderState As SCARD_READERSTATE, _
                                        ByVal ReaderCount As Long) As Long

'=====================================================================

Public Sub LoadListToControl(ByVal Ctrl As ComboBox, ByVal ReaderList As String)
Dim sTemp As String
Dim indx As Integer

indx = 1
sTemp = ""
Ctrl.Clear
While (Mid(ReaderList, indx, 1) <> vbNullChar)
   While (Mid(ReaderList, indx, 1) <> vbNullChar)
     sTemp = sTemp + Mid(ReaderList, indx, 1)
     indx = indx + 1
   Wend
   indx = indx + 1
   Ctrl.AddItem sTemp
   sTemp = ""
```

```
Wend

End Sub


Public Function GetScardErrMsg(ByVal ReturnCode As Long) As String
  Select Case ReturnCode
    Case SCARD_E_CANCELLED
    GetScardErrMsg = "The action was canceled by an SCardCancel request."
    Case SCARD_E_CANT_DISPOSE
    GetScardErrMsg = "The system could not dispose of the media in the requested
manner."
    Case SCARD_E_CARD_UNSUPPORTED
    GetScardErrMsg = "The smart card does not meet minimal requirements for support."
    Case SCARD_E_DUPLICATE_READER
    GetScardErrMsg = "The reader driver didn't produce a unique reader name."
    Case SCARD_E_INSUFFICIENT_BUFFER
    GetScardErrMsg = "The data buffer for returned data is too small for the returned data."
    Case SCARD_E_INVALID_ATR
    GetScardErrMsg = "An ATR string obtained from the registry is not a valid ATR string."
    Case SCARD_E_INVALID_HANDLE
    GetScardErrMsg = "The supplied handle was invalid."
    Case SCARD_E_INVALID_PARAMETER
    GetScardErrMsg = "One or more of the supplied parameters could not be properly
interpreted."
    Case SCARD_E_INVALID_TARGET
    GetScardErrMsg = "Registry startup information is missing or invalid."
    Case SCARD_E_INVALID_VALUE
    GetScardErrMsg = "One or more of the supplied parameter values could not be properly
interpreted."
    Case SCARD_E_NOT_READY
    GetScardErrMsg = "The reader or card is not ready to accept commands."
    Case SCARD_E_NOT_TRANSACTED
    GetScardErrMsg = "An attempt was made to end a non-existent transaction."
    Case SCARD_E_NO_MEMORY
    GetScardErrMsg = "Not enough memory available to complete this command."
    Case SCARD_E_NO_SERVICE
    GetScardErrMsg = "The smart card resource manager is not running."
    Case SCARD_E_NO_SMARTCARD
    GetScardErrMsg = "The operation requires a smart card, but no smart card is currently in
the device."
    Case SCARD_E_PCI_TOO_SMALL
    GetScardErrMsg = "The PCI receive buffer was too small."
    Case SCARD_E_PROTO_MISMATCH
    GetScardErrMsg = "The requested protocols are incompatible with the protocol currently
in use with the card."
    Case SCARD_E_READER_UNAVAILABLE
```

```
   GetScardErrMsg = "The specified reader is not currently available for use."
   Case SCARD_E_READER_UNSUPPORTED
   GetScardErrMsg = "The reader driver does not meet minimal requirements for support."
   Case SCARD_E_SERVICE_STOPPED
   GetScardErrMsg = "The smart card resource manager has shut down."
   Case SCARD_E_SHARING_VIOLATION
   GetScardErrMsg = "The smart card cannot be accessed because of other outstanding
connections."
   Case SCARD_E_SYSTEM_CANCELLED
   GetScardErrMsg = "The action was canceled by the system, presumably to log off or
shut down."
   Case SCARD_E_TIMEOUT
   GetScardErrMsg = "The user-specified timeout value has expired."
   Case SCARD_E_UNKNOWN_CARD
   GetScardErrMsg = "The specified smart card name is not recognized."
   Case SCARD_E_UNKNOWN_READER
   GetScardErrMsg = "The specified reader name is not recognized."
   Case SCARD_F_COMM_ERROR
   GetScardErrMsg = "An internal communications error has been detected."
   Case SCARD_F_INTERNAL_ERROR
   GetScardErrMsg = "An internal consistency check failed."
   Case SCARD_F_UNKNOWN_ERROR
   GetScardErrMsg = "An internal error has been detected, but the source is unknown."
   Case SCARD_F_WAITED_TOO_LONG
   GetScardErrMsg = "An internal consistency timer has expired."
   Case SCARD_S_SUCCESS
   GetScardErrMsg = "No error was encountered."
   Case SCARD_W_REMOVED_CARD
   GetScardErrMsg = "The smart card has been removed, so that further communication is
not possible."
   Case SCARD_W_RESET_CARD
   GetScardErrMsg = "The smart card has been reset, so any shared state information is
invalid."
   Case SCARD_W_UNPOWERED_CARD
   GetScardErrMsg = "Power has been removed from the smart card, so that further
communication is not possible."
   Case SCARD_W_UNRESPONSIVE_CARD
   GetScardErrMsg = "The smart card is not responding to a reset."
   Case SCARD_W_UNSUPPORTED_CARD
   GetScardErrMsg = "The reader cannot communicate with the card, due to ATR string
configuration conflicts."
   Case Else
   GetScardErrMsg = "?"
   End Select

End Function
```

Main (Authentication, DES Format, Create File in Card, Read Card, Write Card, Read File, Write File)

```
Option Explicit
Dim retCode, Protocol, hContext, hCard, ReaderCount As Long
Dim sReaderList As String * 256
Dim sReaderGroup As String
Dim ConnActive As Boolean
Dim ioRequest As SCARD_IO_REQUEST
Dim SendLen, RecvLen As Long
Dim SendBuff(0 To 262) As Byte
Dim RecvBuff(0 To 262) As Byte

Const INVALID_SW1SW2 = -450

Private Sub ClearBuffers()

  Dim indx As Long

  For indx = 0 To 262
    RecvBuff(indx) = &H0
    SendBuff(indx) = &H0
  Next indx

End Sub

Private Sub InitMenu()

  cbReader.Clear
  bInit.Enabled = True
  bConnect.Enabled = False
  bFormat.Enabled = False
  bReset.Enabled = False
  fUserFile.Enabled = False
  fFunction.Enabled = False
  mMsg.Text = ""
  tData.Text = ""
  tData.Enabled = False
  rbAA11.Value = False
  rbBB22.Value = False
  rbCC33.Value = False
  Call DisplayOut(0, 0, "Program ready")

End Sub

Private Sub DisplayOut(ByVal mType As Integer, ByVal msgCode As Long, ByVal
PrintText As String)
```

```
  Select Case mType
    Case 0                ' Notifications only
      mMsg.SelColor = &H4000
    Case 1                ' Error Messages
      mMsg.SelColor = vbRed
      PrintText = GetScardErrMsg(retCode)
    Case 2
      mMsg.SelColor = vbBlack
      PrintText = "< " & PrintText
    Case 3
      mMsg.SelColor = vbBlack
      PrintText = "> " & PrintText
  End Select

  mMsg.SelText = PrintText & vbCrLf
  mMsg.SelStart = Len(mMsg.Text)
  mMsg.SelColor = vbBlack

End Sub

Private Sub AddButtons()

  bInit.Enabled = False
  bConnect.Enabled = True
  bReset.Enabled = True

End Sub

Private Function SendAPDUandDisplay(ByVal SendType As Integer, ByVal ApduIn As
String) As Long

  Dim indx As Integer
  Dim tmpStr As String

  ioRequest.dwProtocol = Protocol
  ioRequest.cbPciLength = Len(ioRequest)
  Call DisplayOut(2, 0, ApduIn)
  tmpStr = ""
  RecvLen = 262

  retCode = SCardTransmit(hCard, _
                    ioRequest, _
                    SendBuff(0), _
                    SendLen, _
                    ioRequest, _
                    RecvBuff(0), _
                    RecvLen)
```

```
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    SendAPDUandDisplay = retCode
    Exit Function
  Else
    Select Case SendType
      Case 0                ' Read all data received
        For indx = 0 To RecvLen - 1
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
      Case 1                ' Read ATR after checking SW1/SW2
        For indx = RecvLen - 2 To RecvLen - 1
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
        If tmpStr <> "90 00 " Then
          Call DisplayOut(1, 0, "Return bytes are not acceptable.")
        Else
          tmpStr = "ATR: "
          For indx = 0 To RecvLen - 3
            tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
          Next indx
        End If
      Case 2                ' Read data after checking SW1/SW2
        For indx = RecvLen - 2 To RecvLen - 1
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
        If tmpStr <> "90 00 " Then
          Call DisplayOut(1, 0, "Return bytes are not acceptable.")
        Else
          tmpStr = ""
          For indx = 0 To RecvLen - 3
            tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
          Next indx
        End If
    End Select
    Call DisplayOut(3, 0, tmpStr)
  End If
  SendAPDUandDisplay = retCode

End Function

Private Function SubmitIC() As Long

  Dim indx As Integer
  Dim tmpStr As String
  Call ClearBuffers
  SendBuff(0) = &H80        ' CLA
```

```
  SendBuff(1) = &H20       ' INS
  SendBuff(2) = &H7        ' P1
  SendBuff(3) = &H0        ' P2
  SendBuff(4) = &H8        ' P3
  SendBuff(5) = &H41       ' A
  SendBuff(6) = &H43       ' C
  SendBuff(7) = &H4F       ' O
  SendBuff(8) = &H53       ' S
  SendBuff(9) = &H54       ' T
  SendBuff(10) = &H45      ' E
  SendBuff(11) = &H53      ' S
  SendBuff(12) = &H54      ' T

  SendLen = &HD
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    SubmitIC = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    SubmitIC = INVALID_SW1SW2
    Exit Function
  End If

  SubmitIC = retCode

End Function

Private Function SelectFile(ByVal HiAddr As Byte, ByVal LoAddr As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &HA4       ' INS
  SendBuff(2) = &H0        ' P1
```

```
  SendBuff(3) = &H0         ' P2
  SendBuff(4) = &H2         ' P3
  SendBuff(5) = HiAddr      ' Value of High Byte
  SendBuff(6) = LoAddr      ' Value of Low Byte

  SendLen = &O7
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    SelectFile = retCode
    Exit Function
  End If

  SelectFile = retCode

End Function

Private Function readRecord(ByVal RecNo As Byte, ByVal dataLen As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  ' 1. Read data from card
  Call ClearBuffers
  SendBuff(0) = &H80        ' CLA
  SendBuff(1) = &HB2        ' INS
  SendBuff(2) = RecNo       ' Record No
  SendBuff(3) = &H0         ' P2
  SendBuff(4) = dataLen     ' Length of Data
  SendLen = 5
  RecvLen = SendBuff(4) + 2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    readRecord = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx + SendBuff(4))), "00") & " "
```

```
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    readRecord = INVALID_SW1SW2
    Exit Function
  End If

  readRecord = retCode

End Function

Private Function writeRecord(ByVal caseType As Integer, ByVal RecNo As Byte, ByVal
maxLen As Byte, _
                    ByVal dataLen As Byte, ByRef ApduIn() As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  If caseType = 1 Then   ' If card data is to be erased before writing new data
    ' 1. Re-initialize card values to $00
    Call ClearBuffers
    SendBuff(0) = &H80       ' CLA
    SendBuff(1) = &HD2       ' INS
    SendBuff(2) = RecNo      ' Record No
    SendBuff(3) = &H0        ' P2
    SendBuff(4) = maxLen     ' Length of Data
    For indx = 0 To maxLen - 1
      SendBuff(indx + 5) = &H0
    Next indx
    SendLen = SendBuff(4) + 5
    RecvLen = &H2
    tmpStr = ""
    For indx = 0 To SendLen - 1
      tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
    Next indx
    retCode = SendAPDUandDisplay(0, tmpStr)
    If retCode <> SCARD_S_SUCCESS Then
      writeRecord = retCode
      Exit Function
    End If
    tmpStr = ""
    For indx = 0 To 1
      tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
    Next indx
    If tmpStr <> "90 00 " Then
      Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
      writeRecord = INVALID_SW1SW2
```

```
    Exit Function
   End If
  End If

  ' 2. Write data to card
  Call ClearBuffers
  SendBuff(0) = &H80        ' CLA
  SendBuff(1) = &HD2        ' INS
  SendBuff(2) = RecNo       ' Record No
  SendBuff(3) = &H0         ' P2
  SendBuff(4) = dataLen     ' Length of Data
  For indx = 0 To dataLen - 1
   SendBuff(indx + 5) = ApduIn(indx)
  Next indx
  SendLen = SendBuff(4) + 5
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
   tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
   writeRecord = retCode
   Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
   tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
   Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
   writeRecord = INVALID_SW1SW2
   Exit Function
  End If

  writeRecord = retCode

End Function

Private Sub bConnect_Click()

  If ConnActive Then
   Call DisplayOut(0, 0, "Connection is already active.")
   Exit Sub
  End If

  Call DisplayOut(2, 0, "Invoke SCardConnect")
```

```
  ' 1. Connect to selected reader using hContext handle
  '    and obtain valid hCard handle
  retCode = SCardConnect(hContext, _
                    cbReader.Text, _
                    SCARD_SHARE_EXCLUSIVE, _
                    SCARD_PROTOCOL_T0 Or SCARD_PROTOCOL_T1, _
                    hCard, _
                    Protocol)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    ConnActive = False
    Exit Sub
  Else
    Call DisplayOut(0, 0, "Successful connection to " & cbReader.Text)
  End If

  ConnActive = True
  bFormat.Enabled = True
  fUserFile.Enabled = True
  rbAA11.Value = True
  fFunction.Enabled = True
  tData.Enabled = True
  tData.Text = ""
  tData.MaxLength = 10

End Sub

Private Sub bFormat_Click()

  Dim indx As Integer
  Dim tmpStr As String
  Dim tmpArray(0 To 31) As Byte

  ' 1. Send IC Code
  retCode = SubmitIC()
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If

  ' 2. Select FF 02
  retCode = SelectFile(&HFF, &H2)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
```

```
Next indx
If tmpStr <> "90 00 " Then
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
  Exit Sub
End If


' 3. Write to FF 02
'    This will create 3 User files, no Option registers and
'    Security Option registers defined, Personalization bit
'    is not set
tmpArray(0) = &H0      ' 00    Option registers
tmpArray(1) = &H0      ' 00    Security option register
tmpArray(2) = &H3      ' 03    No of user files
tmpArray(3) = &H0      ' 00    Personalization bit
retCode = writeRecord(0, &H0, &H4, &H4, tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
Call DisplayOut(0, 0, "FF 02 is updated")

' 4. Perform a reset for changes in the ACOS to take effect
retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
retCode = SCardConnect(hContext, _
                cbReader.Text, _
                SCARD_SHARE_EXCLUSIVE, _
                SCARD_PROTOCOL_T0 Or SCARD_PROTOCOL_T1, _
                hCard, _
                Protocol)
If retCode <> SCARD_S_SUCCESS Then
  Call DisplayOut(1, retCode, "")
  ConnActive = False
  Exit Sub
Else
  Call DisplayOut(0, 0, "Card reset is successful.")
End If

' 5. Select FF 04
retCode = SelectFile(&HFF, &H4)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
tmpStr = ""
For indx = 0 To 1
  tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
Next indx
If tmpStr <> "90 00 " Then
```

```
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
  Exit Sub
End If

' 6. Send IC Code
retCode = SubmitIC()
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If

' 7. Write to FF 04
' 7.1. Write to first record of FF 04
tmpArray(0) = &HA       ' 10    Record length
tmpArray(1) = &H3       ' 3     No of records
tmpArray(2) = &H0       ' 00    Read security attribute
tmpArray(3) = &H0       ' 00    Write security attribute
tmpArray(4) = &HAA      ' AA    File identifier
tmpArray(5) = &H11      ' 11    File identifier
retCode = writeRecord(0, &H0, &H6, &H6, tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
Call DisplayOut(0, 0, "User File AA 11 is defined")

' 7.2. Write to second record of FF 04
tmpArray(0) = &H10      ' 16    Record length
tmpArray(1) = &H2       ' 2     No of records
tmpArray(2) = &H0       ' 00    Read security attribute
tmpArray(3) = &H0       ' 00    Write security attribute
tmpArray(4) = &HBB      ' BB    File identifier
tmpArray(5) = &H22      ' 22    File identifier
retCode = writeRecord(0, &H1, &H6, &H6, tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
Call DisplayOut(0, 0, "User File BB 22 is defined")

' 7.3. Write to third record of FF 04
tmpArray(0) = &H20      ' 32    Record length
tmpArray(1) = &H4       ' 4     No of records
tmpArray(2) = &H0       ' 00    Read security attribute
tmpArray(3) = &H0       ' 00    Write security attribute
tmpArray(4) = &HCC      ' CC    File identifier
tmpArray(5) = &H33      ' 33    File identifier
retCode = writeRecord(0, &H2, &H6, &H6, tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
```

```
  End If
  Call DisplayOut(0, 0, "User File CC 33 is defined")

End Sub

Private Sub bInit_Click()

  sReaderList = String(255, vbNullChar)
  ReaderCount = 255

  ' 1. Establish context and obtain hContext handle
  retCode = SCardEstablishContext(SCARD_SCOPE_USER, 0, 0, hContext)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    Exit Sub
  End If

  ' 2. List PC/SC card readers installed in the system
  retCode = SCardListReaders(hContext, sReaderGroup, sReaderList, ReaderCount)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    Exit Sub
  End If
  Call LoadListToControl(cbReader, sReaderList)
  cbReader.ListIndex = 0

  Call AddButtons

End Sub

Private Sub bQuuit_Click()

  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If
  retCode = SCardReleaseContext(hContext)
  Unload Me


End Sub

Private Sub bRead_Click()

  Dim indx As Integer
  Dim tmpStr, ChkStr As String
  Dim HiAddr, LoAddr, dataLen As Byte
```

```
 ' 1. Check User File selected by user
If rbAA11.Value = True Then
  HiAddr = &HAA
  LoAddr = &H11
  dataLen = &HA
  ChkStr = "91 00 "
End If

If rbBB22.Value = True Then
  HiAddr = &HBB
  LoAddr = &H22
  dataLen = &H10
  ChkStr = "91 01 "
End If

If rbCC33.Value = True Then
  HiAddr = &HCC
  LoAddr = &H33
  dataLen = &H20
  ChkStr = "91 02 "
End If

' 2. Select User File
retCode = SelectFile(HiAddr, LoAddr)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
tmpStr = ""
For indx = 0 To 1
  tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
Next indx
If tmpStr <> ChkStr Then
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
  Exit Sub
End If

' 3. Read First Record of User File selected
retCode = readRecord(&H0, dataLen)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If

' 4. Display data read from card to textbox
tmpStr = ""
indx = 0
While (RecvBuff(indx) <> &H0)
  If indx < tData.MaxLength Then
```

```
      tmpStr = tmpStr & Chr(RecvBuff(indx))
    End If
    indx = indx + 1
  Wend
  tData.Text = tmpStr
  Call DisplayOut(0, 0, "Data read from card is displayed in Text Box.")

End Sub

Private Sub bReset_Click()

  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If
  retCode = SCardReleaseContext(hContext)
  Call InitMenu

End Sub

Private Sub bWrite_Click()

  Dim indx As Integer
  Dim tmpStr, ChkStr As String
  Dim HiAddr, LoAddr, dataLen As Byte
  Dim tmpArray(0 To 56) As Byte

  ' 1. Validate input template
  If tData.Text = "" Then
    tData.SetFocus
    Exit Sub
  End If

  ' 2. Check User File selected by user
  If rbAA11.Value = True Then
    HiAddr = &HAA
    LoAddr = &H11
    dataLen = &HA
    ChkStr = "91 00 "
  End If

  If rbBB22.Value = True Then
    HiAddr = &HBB
    LoAddr = &H22
    dataLen = &H10
    ChkStr = "91 01 "
  End If
```

```
  If rbCC33.Value = True Then
    HiAddr = &HCC
    LoAddr = &H33
    dataLen = &H20
    ChkStr = "91 02 "
  End If

  ' 3. Select User File
  retCode = SelectFile(HiAddr, LoAddr)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> ChkStr Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    Exit Sub
  End If

  ' 4. Write data from text box to card
  tmpStr = tData.Text
  For indx = 0 To Len(tmpStr) - 1
    tmpArray(indx) = Asc(Mid(tmpStr, indx + 1, 1))
  Next indx
  retCode = writeRecord(1, &H0, dataLen, Len(tmpStr), tmpArray)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If
  Call DisplayOut(0, 0, "Data read from Text Box is written to card.")


End Sub

Private Sub cbReader_Click()

  bFormat.Enabled = False
  tData.Text = ""
  tData.Enabled = False
  rbAA11.Value = False
  rbBB22.Value = False
  rbCC33.Value = False
  fUserFile.Enabled = False
  fFunction.Enabled = False
  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
```

```
    ConnActive = False
  End If

End Sub

Private Sub Form_Load()

  Call InitMenu

End Sub

Private Sub rbAA11_Click()

  tData.Text = ""
  tData.MaxLength = 10

End Sub

Private Sub rbBB22_Click()

  tData.Text = ""
  tData.MaxLength = 16

End Sub

Private Sub rbCC33_Click()

  tData.Text = ""
  tData.MaxLength = 32

End Sub
```

Appendix II – Formatting smart card with DES/3DES and Mutual Authentication to EMV smart card.

Chain3DES (module)

```
'============== ENCRYPTION ALGORITHM Constants ================
Global Const ALGO_DES = 0
Global Const ALGO_3DES = 1
Global Const ALGO_XOR = 3
Global Const DATA_ENCRYPT = 1
Global Const DATA_DECRYPT = 2


'Note : Block is equal to 8 bytes. So to encrypt/decrypt 8 bytes of data  user must use 1
‘block in the parameter.
'       Example:
'        'This code encrypts 8 bytes of data!
'        Dim Data(1 to 8) as byte ' Assume data was entered
'        Dim Key(1 to 8) as byte ' Assume key already exits
'        Chain_DES(Data(1), Key(1), ALGO_3DES , 1 ,DATA_ENCRYPT)
'
'=================================================================
'      CHAIN_DES PROTOTYPE
'=================================================================
Declare Function Chain_DES Lib "chaindes.dll" (ByRef Data As Any, ByRef key As Any,
ByVal TripleDES As Integer, ByVal Blocks As Long, ByVal method As Long) As Long
Declare Function Chain_MAC Lib "chaindes.dll" (ByRef mac As Any, ByRef Data As
Any, ByRef key As Any, ByVal Blocks As Long) As Long
Declare Function Chain_MAC2 Lib "chaindes.dll" (ByRef mac As Any, ByRef Data As
Any, ByRef key As Any, ByVal Blocks As Long) As Long
```

Main Mutual Authentication (DES - 3DES FORMAT, MUTUAL PROCESS, READ, WRITE)

```
Option Explicit

Dim retCode, Protocol, hContext, hCard, ReaderCount As Long
Dim sReaderList As String * 256
Dim sReaderGroup As String
Dim ConnActive As Boolean
Dim ioRequest As SCARD_IO_REQUEST
Dim SendLen, RecvLen As Long
Dim SendBuff(0 To 262) As Byte
Dim RecvBuff(0 To 262) As Byte

Const INVALID_SW1SW2 = -450
```

```
' this routine will encrypt 8-byte data with 8-byte key
' the result is stored in data
Public Sub DES(Data() As Byte, key() As Byte)
   Call Chain_DES(Data(0), key(0), ALGO_DES, 1, DATA_ENCRYPT)
End Sub

' this routine will use 3DES algo to encrypt 8-byte data with 16-byte key
' the result is stored in data
Public Sub TripleDES(Data() As Byte, key() As Byte)
   Call Chain_DES(Data(0), key(0), ALGO_3DES, 1, DATA_ENCRYPT)
End Sub


' MAC as defined in ACOS manual
' receives 8-byte Key and 16-byte Data
' result is stored in Data
Public Sub mac(Data() As Byte, key() As Byte)
Dim i As Integer

   DES Data, key
   For i = 0 To 7
      Data(i) = Data(i) Xor Data(i + 8)
   Next
   DES Data, key
End Sub

' Triple MAC as defined in ACOS manual
' receives 16-byte Key and 16-byte Data
' result is stored in Data
Public Sub TripleMAC(Data() As Byte, key() As Byte)
Dim i As Integer

   TripleDES Data, key
   For i = 0 To 7
      Data(i) = Data(i) Xor Data(i + 8)
   Next
   TripleDES Data, key
End Sub

Private Sub ClearBuffers()

  Dim indx As Long

  For indx = 0 To 262
    RecvBuff(indx) = &H0
    SendBuff(indx) = &H0
```

```
  Next indx

End Sub
Private Sub InitMenu()

  cbReader.Clear
  bInit.Enabled = True
  bConnect.Enabled = False
  bReset.Enabled = False
  Call ClearTextFields
  fSecOption.Enabled = False
  fKey.Enabled = False
  bExecMA.Enabled = False
  mMsg.Text = ""
  rbDES.Value = False
  rb3DES.Value = False
  Call DisplayOut(0, 0, "Program ready")

End Sub

Private Sub DisplayOut(ByVal mType As Integer, ByVal msgCode As Long, ByVal
PrintText As String)

  Select Case mType
    Case 0                    ' Notifications only
      mMsg.SelColor = &H4000
    Case 1                    ' PC/SC Error Messages
      mMsg.SelColor = vbRed
      PrintText = GetScardErrMsg(retCode)
    Case 2
      mMsg.SelColor = vbBlack      ' Input APDU command
      PrintText = "< " & PrintText
    Case 3
      mMsg.SelColor = vbBlack      ' Output data
      PrintText = "> " & PrintText
    Case 4
      mMsg.SelColor = vbRed        ' Notifications on red font
  End Select

  mMsg.SelText = PrintText & vbCrLf
  mMsg.SelStart = Len(mMsg.Text)
  mMsg.SelColor = vbBlack


End Sub

Private Sub AddButtons()
```

```
  bInit.Enabled = False
  bConnect.Enabled = True
  bReset.Enabled = True

End Sub

Private Sub ClearTextFields()

  tCard.Text = ""
  tTerminal.Text = ""

End Sub

Private Function SendAPDUandDisplay(ByVal SendType As Integer, ByVal ApduIn As
String) As Long

  Dim indx As Integer
  Dim tmpStr As String

  ioRequest.dwProtocol = Protocol
  ioRequest.cbPciLength = Len(ioRequest)
  Call DisplayOut(2, 0, ApduIn)
  tmpStr = ""
  RecvLen = 262

  retCode = SCardTransmit(hCard, _
                    ioRequest, _
                    SendBuff(0), _
                    SendLen, _
                    ioRequest, _
                    RecvBuff(0), _
                    RecvLen)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    SendAPDUandDisplay = retCode
    Exit Function
  Else
    Select Case SendType
      Case 0              ' Read all data received
        For indx = 0 To RecvLen - 1
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
      Case 1              ' Read ATR after checking SW1/SW2
        For indx = RecvLen - 2 To RecvLen - 1
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
```

```
      If tmpStr <> "90 00 " Then
        Call DisplayOut(1, 0, "Return bytes are not acceptable.")
      Else
        tmpStr = "ATR: "
        For indx = 0 To RecvLen - 3
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
      End If
    Case 2                ' Read data after checking SW1/SW2
      For indx = RecvLen - 2 To RecvLen - 1
        tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
      Next indx
      If tmpStr <> "90 00 " Then
        Call DisplayOut(1, 0, "Return bytes are not acceptable.")
      Else
        tmpStr = ""
        For indx = 0 To RecvLen - 3
          tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
        Next indx
      End If
  End Select
  Call DisplayOut(3, 0, tmpStr)
 End If
 SendAPDUandDisplay = retCode

End Function

Private Function SubmitIC() As Long

 Dim indx As Integer
 Dim tmpStr As String

 Call ClearBuffers
 SendBuff(0) = &H80       ' CLA
 SendBuff(1) = &H20       ' INS
 SendBuff(2) = &H7        ' P1
 SendBuff(3) = &H0        ' P2
 SendBuff(4) = &H8        ' P3
 SendBuff(5) = &H41       ' A
 SendBuff(6) = &H43       ' C
 SendBuff(7) = &H4F       ' O
 SendBuff(8) = &H53       ' S
 SendBuff(9) = &H54       ' T
 SendBuff(10) = &H45      ' E
 SendBuff(11) = &H53      ' S
 SendBuff(12) = &H54      ' T
```

```
  SendLen = &HD
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    SubmitIC = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    SubmitIC = INVALID_SW1SW2
    Exit Function
  End If

  SubmitIC = retCode

End Function

Private Function StartSession() As Long

  Dim indx As Integer
  Dim tmpStr As String

  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &H84       ' INS
  SendBuff(2) = &H0       ' P1
  SendBuff(3) = &H0        ' P2
  SendBuff(4) = &H8        ' P3

  SendLen = &H5
  RecvLen = &HA
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    StartSession = retCode
    Exit Function
```

```
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx + SendBuff(4))), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    StartSession = INVALID_SW1SW2
    Exit Function
  End If

  StartSession = retCode

End Function

Private Function SelectFile(ByVal HiAddr As Byte, ByVal LoAddr As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &HA4       ' INS
  SendBuff(2) = &H0        ' P1
  SendBuff(3) = &H0        ' P2
  SendBuff(4) = &H2        ' P3
  SendBuff(5) = HiAddr     ' Value of High Byte
  SendBuff(6) = LoAddr     ' Value of Low Byte

  SendLen = &O7
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    SelectFile = retCode
    Exit Function
  End If

  SelectFile = retCode

End Function

Private Function readRecord(ByVal RecNo As Byte, ByVal dataLen As Byte) As Long
```

```
  Dim indx As Integer
  Dim tmpStr As String

  ' 1. Read data from card
  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &HB2       ' INS
  SendBuff(2) = RecNo      ' Record No
  SendBuff(3) = &H0        ' P2
  SendBuff(4) = dataLen    ' Length of Data
  SendLen = 5
  RecvLen = SendBuff(4) + 2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    readRecord = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx + SendBuff(4))), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    readRecord = INVALID_SW1SW2
    Exit Function
  End If

  readRecord = retCode

End Function

Private Function writeRecord(ByVal caseType As Integer, ByVal RecNo As Byte, ByVal
maxLen As Byte, _
                    ByVal dataLen As Byte, ByRef ApduIn() As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  If caseType = 1 Then   ' If card data is to be erased before writing new data
    ' 1. Re-initialize card values to $00
    Call ClearBuffers
    SendBuff(0) = &H80       ' CLA
    SendBuff(1) = &HD2       ' INS
```

```
  SendBuff(2) = RecNo       ' Record No
  SendBuff(3) = &H0         ' P2
  SendBuff(4) = maxLen      ' Length of Data
  For indx = 0 To maxLen - 1
    SendBuff(indx + 5) = &H0
  Next indx
  SendLen = SendBuff(4) + 5
  RecvLen = &H2
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    writeRecord = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    writeRecord = INVALID_SW1SW2
    Exit Function
  End If
End If

' 2. Write data to card
Call ClearBuffers
SendBuff(0) = &H80        ' CLA
SendBuff(1) = &HD2        ' INS
SendBuff(2) = RecNo       ' Record No
SendBuff(3) = &H0         ' P2
SendBuff(4) = dataLen     ' Length of Data
For indx = 0 To dataLen - 1
  SendBuff(indx + 5) = ApduIn(indx)
Next indx
SendLen = SendBuff(4) + 5
RecvLen = &H2
tmpStr = ""
For indx = 0 To SendLen - 1
  tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
Next indx
retCode = SendAPDUandDisplay(0, tmpStr)
If retCode <> SCARD_S_SUCCESS Then
  writeRecord = retCode
```

```
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    writeRecord = INVALID_SW1SW2
    Exit Function
  End If

  writeRecord = retCode

End Function

Private Function ValidTemplate() As Boolean

  If Len(tCard.Text) < tCard.MaxLength Then
    tCard.SetFocus
    ValidTemplate = False
    Exit Function
  End If

  If Len(tTerminal.Text) < tTerminal.MaxLength Then
    tTerminal.SetFocus
    ValidTemplate = False
    Exit Function
  End If

  ValidTemplate = True

End Function

Private Function CheckACOS() As Boolean

  Dim indx As Integer
  Dim tmpStr As String

 ' 1. Reconnect reader to accommodate change of cards
  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If
  retCode = SCardConnect(hContext, _
                    cbReader.Text, _
                    SCARD_SHARE_EXCLUSIVE, _
```

```
                SCARD_PROTOCOL_T0 Or SCARD_PROTOCOL_T1, _
                hCard, _
                Protocol)
If retCode <> SCARD_S_SUCCESS Then
  Call DisplayOut(1, retCode, "")
  ConnActive = False
  CheckACOS = False
  Exit Function
End If
ConnActive = True

' 2. Check for File FF 00
retCode = SelectFile(&HFF, &H0)
If retCode <> SCARD_S_SUCCESS Then
  CheckACOS = False
  Exit Function
End If
tmpStr = ""
For indx = 0 To 1
  tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
Next indx
If tmpStr <> "90 00 " Then
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
  CheckACOS = False
  Exit Function
End If

' 3. Check for File FF 01
retCode = SelectFile(&HFF, &H1)
If retCode <> SCARD_S_SUCCESS Then
  CheckACOS = False
  Exit Function
End If
tmpStr = ""
For indx = 0 To 1
  tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
Next indx
If tmpStr <> "90 00 " Then
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
  CheckACOS = False
  Exit Function
End If

' 4. Check for File FF 02
retCode = SelectFile(&HFF, &H2)
If retCode <> SCARD_S_SUCCESS Then
  CheckACOS = False
```

```
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    CheckACOS = False
    Exit Function
  End If

  CheckACOS = True

End Function

Private Function ACOSError(ByVal Sw1 As Byte, ByVal Sw2 As Byte) As Boolean

  ' Check for error returned by ACOS card
  ACOSError = True
  If ((Sw1 = &H62) And (Sw2 = &H81)) Then
    Call DisplayOut(4, 0, "Account data may be corrupted.")
      Exit Function
  End If
  If (Sw1 = &H63) Then
    Call DisplayOut(4, 0, "MAC cryptographic checksum is wrong.")
      Exit Function
  End If
  If ((Sw1 = &H69) And (Sw2 = &H66)) Then
    Call DisplayOut(4, 0, "Command not available or option bit not set.")
      Exit Function
  End If
  If ((Sw1 = &H69) And (Sw2 = &H82)) Then
    Call DisplayOut(4, 0, "Security status not satisfied. Secret code, IC or PIN not
submitted.")
      Exit Function
  End If
  If ((Sw1 = &H69) And (Sw2 = &H83)) Then
    Call DisplayOut(4, 0, "The specified code is locked.")
      Exit Function
  End If
  If ((Sw1 = &H69) And (Sw2 = &H85)) Then
    Call DisplayOut(4, 0, "Preceding transaction was not DEBIT or mutual authentication
has not been completed.")
      Exit Function
  End If
  If ((Sw1 = &H69) And (Sw2 = &HF0)) Then
```

```
    Call DisplayOut(4, 0, "Data in account is inconsistent. No access unless in Issuer
mode.")
      Exit Function
  End If
  If ((Sw1 = &H6A) And (Sw2 = &H82)) Then
    Call DisplayOut(4, 0, "Account does not exist.")
      Exit Function
  End If
  If ((Sw1 = &H6A) And (Sw2 = &H83)) Then
    Call DisplayOut(4, 0, "Record not found or file too short.")
      Exit Function
  End If
  If ((Sw1 = &H6A) And (Sw2 = &H86)) Then
    Call DisplayOut(4, 0, "P1 or P2 is incorrect.")
      Exit Function
  End If
  If ((Sw1 = &H6B) And (Sw2 = &H20)) Then
    Call DisplayOut(4, 0, "Invalid amount in DEBIT/CREDIT command.")
      Exit Function
  End If
  If (Sw1 = &H6C) Then
    Call DisplayOut(4, 0, "Issue GET RESPONSE with P3 = " & Hex(Sw2) & " to get
response data.")
      Exit Function
  End If
  If (Sw1 = &H6D) Then
    Call DisplayOut(4, 0, "Unknown INS.")
      Exit Function
  End If
  If (Sw1 = &H6E) Then
    Call DisplayOut(4, 0, "Unknown CLA.")
      Exit Function
  End If
  If ((Sw1 = &H6F) And (Sw2 = &H10)) Then
    Call DisplayOut(4, 0, "Account Transaction Counter at maximum. No more transaction
possible.")
      Exit Function
  End If

  ACOSError = False

End Function

Private Function GetResponse() As Long

  Dim indx As Integer
  Dim tmpStr As String
```

```
  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &HC0       ' INS
  SendBuff(2) = &H0        ' P1
  SendBuff(3) = &H0        ' P2
  SendBuff(4) = &H8        ' Length of Data
  SendLen = 5
  RecvLen = &HA
  tmpStr = ""
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    GetResponse = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx + SendBuff(4))), "00") & " "
  Next indx
  If ACOSError(RecvBuff(SendBuff(4)), RecvBuff(SendBuff(4) + 1)) Then
    GetResponse = INVALID_SW1SW2
    Exit Function
  End If
  If tmpStr <> "90 00 " Then
    Call DisplayOut(4, 0, "GET RESPONSE command failed.")
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    GetResponse = INVALID_SW1SW2
    Exit Function
  End If

  GetResponse = retCode

End Function

Private Function Authenticate(ByRef DataIn() As Byte) As Long

  Dim indx As Integer
  Dim tmpStr As String

  Call ClearBuffers
  SendBuff(0) = &H80       ' CLA
  SendBuff(1) = &H82       ' INS
  SendBuff(2) = &H0        ' P1
  SendBuff(3) = &H0        ' P2
```

```
  SendBuff(4) = &H10        ' P3
  For indx = 0 To 15
    SendBuff(indx + 5) = DataIn(indx)
  Next indx
  SendLen = SendBuff(4) + 5
  RecvLen = &HA
  For indx = 0 To SendLen - 1
    tmpStr = tmpStr & Format(Hex(SendBuff(indx)), "00") & " "
  Next indx
  retCode = SendAPDUandDisplay(0, tmpStr)
  If retCode <> SCARD_S_SUCCESS Then
    Authenticate = retCode
    Exit Function
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If ACOSError(RecvBuff(0), RecvBuff(1)) Then
    Authenticate = INVALID_SW1SW2
    Exit Function
  End If
  If tmpStr <> "61 08 " Then
    Call DisplayOut(4, 0, "AUTHENTICATE command failed.")
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    Authenticate = INVALID_SW1SW2
    Exit Function
  End If

  Authenticate = retCode

End Function

Private Sub bConnect_Click()

  If ConnActive Then
    Call DisplayOut(0, 0, "Connection is already active.")
    Exit Sub
  End If

  Call DisplayOut(2, 0, "Invoke SCardConnect")
  ' 1. Connect to selected reader using hContext handle
  '    and obtain valid hCard handle
  retCode = SCardConnect(hContext, _
                    cbReader.Text, _
                    SCARD_SHARE_EXCLUSIVE, _
                    SCARD_PROTOCOL_T0 Or SCARD_PROTOCOL_T1, _
```

```
                hCard, _
                Protocol)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    ConnActive = False
    Exit Sub
  Else
    Call DisplayOut(0, 0, "Successful connection to " & cbReader.Text)
  End If

  ConnActive = True
  fSecOption.Enabled = True
  fKey.Enabled = True
  bExecMA.Enabled = True
  Call ClearTextFields
  rbDES.Value = True
  tCard.MaxLength = 8
  tTerminal.MaxLength = 8

End Sub

Private Sub bExecMA_Click()

  Dim indx As Integer
  Dim tmpStr As String
  Dim CRnd(0 To 7) As Byte          ' Card random number
  Dim TRnd(0 To 7) As Byte          ' Terminal random number
  Dim cKey(0 To 15) As Byte         ' Card Key
  Dim tKey(0 To 15) As Byte         ' Terminal Key
  Dim tmpArray(0 To 31) As Byte
  Dim tmpResult(0 To 7) As Byte     ' Card-side authentication result
  Dim SessionKey(0 To 15) As Byte
  Dim ReverseKey(0 To 15) As Byte   ' Reverse of Terminal Key

  ' 1. Validate data template
  If Not ValidTemplate Then
    Exit Sub
  End If

  ' 2. Check if card inserted is an ACOS card
  If Not CheckACOS Then
    Call DisplayOut(0, 0, "Please insert an ACOS card.")
    Exit Sub
  End If
  Call DisplayOut(0, 0, "ACOS card is detected.")

  ' 3. Card-side authentication process
```

```
' 3.1. Generate random number from card
retCode = StartSession()
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If

' 3.2. Store the random number generated by the card to Crnd
For indx = 0 To 7
  CRnd(indx) = RecvBuff(indx)
Next indx

' 3.3. Retrieve Terminal Key from Input Template
tmpStr = tTerminal.Text
For indx = 0 To tTerminal.MaxLength - 1
  tKey(indx) = Asc(Mid(tmpStr, indx + 1, 1))
Next indx

' 3.4. Encrypt Random No (CRnd) with Terminal Key (tKey)
'      tmpArray will hold the 8-byte Enrypted number
For indx = 0 To 7
  tmpArray(indx) = CRnd(indx)
Next indx
If rbDES.Value = True Then
  Call DES(tmpArray, tKey)
Else
  Call TripleDES(tmpArray, tKey)
End If

' 3.5. Issue Authenticate command using 8-byte Encrypted No (tmpArray)
'      and Random Terminal number (TRnd)
For indx = 0 To 7
  tmpArray(indx + 8) = TRnd(indx)
Next indx
retCode = Authenticate(tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If

' 3.6. Get 8-byte result of card-side authentication
'      and save to tmpResult
retCode = GetResponse()
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
For indx = 0 To 7
  tmpResult(indx) = RecvBuff(indx)
Next indx
```

```
' 4. Terminal-side authentication process
' 4.1. Retrieve Card Key from Input Template
tmpStr = tCard.Text
For indx = 0 To tCard.MaxLength - 1
  cKey(indx) = Asc(Mid(tmpStr, indx + 1, 1))
Next indx

' 4.2. Compute for Session Key
If rbDES.Value = True Then

  ' 4.2a. for single DES
  ' prepare SessionKey
  ' SessionKey = DES (DES(RNDc, KC) XOR RNDt, KT)

  ' calculate DES(cRnd,cKey)
  For indx = 0 To 7
    tmpArray(indx) = CRnd(indx)
  Next indx
  Call DES(tmpArray, cKey)

  ' XOR the result with tRnd
  For indx = 0 To 7
    tmpArray(indx) = tmpArray(indx) Xor TRnd(indx)
  Next indx

  ' DES the result with tKey
  Call DES(tmpArray, tKey)

  ' temp now holds the SessionKey
  For indx = 0 To 7
    SessionKey(indx) = tmpArray(indx)
  Next indx
Else

  ' 4.2b. for triple DES
  ' prepare SessionKey
  ' Left half SessionKey =  3DES (3DES (CRnd, cKey), tKey)
  ' Right half SessionKey = 3DES (TRnd, REV (tKey))

  ' tmpArray = 3DES (CRnd, cKey)
  For indx = 0 To 7
    tmpArray(indx) = CRnd(indx)
  Next indx
  Call TripleDES(tmpArray, cKey)

  ' tmpArray = 3DES (tmpArray, tKey)
```

```
  Call TripleDES(tmpArray, tKey)

  ' tmpArray holds the left half of SessionKey
  For indx = 0 To 7
    SessionKey(indx) = tmpArray(indx)
  Next indx

  ' compute ReverseKey of tKey
  ' just swap its left side with right side
  ' ReverseKey = right half of tKey + left half of tKey
  For indx = 0 To 7
    ReverseKey(indx) = tKey(8 + indx)
  Next indx
  For indx = 0 To 7
    ReverseKey(8 + indx) = tKey(indx)
  Next indx

  ' compute tmpArray = 3DES (TRnd, ReverseKey)
  For indx = 0 To 7
    tmpArray(indx) = TRnd(indx)
  Next indx
  Call TripleDES(tmpArray, ReverseKey)

  ' tmpArray holds the right half of SessionKey
  For indx = 0 To 7
    SessionKey(indx + 8) = tmpArray(indx)
  Next indx
End If

' 4.3. compute DES (TRnd, SessionKey)
For indx = 0 To 7
  tmpArray(indx) = TRnd(indx)
Next indx
If rbDES.Value = True Then
  Call DES(tmpArray, SessionKey)
Else
  Call TripleDES(tmpArray, SessionKey)
End If

' 5. Compare Card-side and Terminal-side authentication results
For indx = 0 To 7
  If tmpResult(indx) <> tmpArray(indx) Then
    Call DisplayOut(4, 0, "Mutual Authentication failed.")
    Exit Sub
  End If
Next indx
```

```
  Call DisplayOut(0, 0, "Mutual Authentication is successful.")

End Sub

Private Sub bFormat_Click()

  Dim indx As Integer
  Dim tmpStr As String
  Dim tmpArray(0 To 31) As Byte

  ' 1. Validate data template
  If Not ValidTemplate Then
    Exit Sub
  End If

  ' 2. Check if card inserted is an ACOS card
  If Not CheckACOS Then
    Call DisplayOut(0, 0, "Please insert an ACOS card.")
    Exit Sub
  End If
  Call DisplayOut(0, 0, "ACOS card is detected.")

  ' 3. Submit Issuer Code
  retCode = SubmitIC()
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If

  ' 4. Select FF 02
  retCode = SelectFile(&HFF, &H2)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If
  tmpStr = ""
  For indx = 0 To 1
    tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
  Next indx
  If tmpStr <> "90 00 " Then
    Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
    Exit Sub
  End If

  ' 5. Write to FF 02
  '    This step will define the Option registers,
  '    Security Option registers and Personalization bit
  '    are not set
  If rbDES.Value = True Then    ' DES option only
```

```
  tmpArray(0) = &H0          ' 00h  3-DES is not set
Else
  tmpArray(0) = &H2          ' 02h  3-DES is enabled
End If
tmpArray(1) = &H0            ' 00   Security option register
tmpArray(2) = &H3            ' 00   No of user files
tmpArray(3) = &H0            ' 00   Personalization bit
retCode = writeRecord(0, &H0, &H4, &H4, tmpArray)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
Call DisplayOut(0, 0, "FF 02 is updated")

' 6. Perform a reset for changes in the ACOS to take effect
retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
ConnActive = False
retCode = SCardConnect(hContext, _
                cbReader.Text, _
                SCARD_SHARE_EXCLUSIVE, _
                SCARD_PROTOCOL_T0 Or SCARD_PROTOCOL_T1, _
                hCard, _
                Protocol)
If retCode <> SCARD_S_SUCCESS Then
  Call DisplayOut(1, retCode, "")
  ConnActive = False
  Exit Sub
End If
Call DisplayOut(0, 0, "Account files are enabled.")
ConnActive = True

' 7. Submit Issuer Code to write into FF 03
retCode = SubmitIC()
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If

' 8. Select FF 03
retCode = SelectFile(&HFF, &H3)
If retCode <> SCARD_S_SUCCESS Then
  Exit Sub
End If
tmpStr = ""
For indx = 0 To 1
  tmpStr = tmpStr & Format(Hex(RecvBuff(indx)), "00") & " "
Next indx
If tmpStr <> "90 00 " Then
  Call DisplayOut(0, 0, "Return string is invalid. Value: " & tmpStr)
```

```
  Exit Sub
End If

' 9. Write to FF 03
If rbDES.Value = True Then    ' DES option uses 8-byte key

'  9a.1. Record 02 for Card key
  tmpStr = tCard.Text
  For indx = 0 To 7
    tmpArray(indx) = Asc(Mid(tmpStr, indx + 1, 1))
  Next indx
  retCode = writeRecord(0, &H2, &H8, &H8, tmpArray)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If

'  9a.2. Record 03 for Terminal key
  tmpStr = tTerminal.Text
  For indx = 0 To 7
    tmpArray(indx) = Asc(Mid(tmpStr, indx + 1, 1))
  Next indx
  retCode = writeRecord(0, &H3, &H8, &H8, tmpArray)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If
Else                        ' 3-DES option uses 16-byte key

'  9b.1. Write Record 02 for Left half of Card key
  tmpStr = tCard.Text
  For indx = 0 To 7          ' Left half of Card key
    tmpArray(indx) = Asc(Mid(tmpStr, indx + 1, 1))
  Next indx
  retCode = writeRecord(0, &H2, &H8, &H8, tmpArray)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If

'  9b.2. Record 12 for Right half of Card key
  For indx = 8 To 15         ' Right half of Card key
    tmpArray(indx - 8) = Asc(Mid(tmpStr, indx + 1, 1))
  Next indx
  retCode = writeRecord(0, &HC, &H8, &H8, tmpArray)
  If retCode <> SCARD_S_SUCCESS Then
    Exit Sub
  End If

'  9b.3. Write Record 03 for Left half of Terminal key
```

```
    tmpStr = tTerminal.Text
    For indx = 0 To 7           ' Left half of Terminal key
      tmpArray(indx) = Asc(Mid(tmpStr, indx + 1, 1))
    Next indx
    retCode = writeRecord(0, &H3, &H8, &H8, tmpArray)
    If retCode <> SCARD_S_SUCCESS Then
      Exit Sub
    End If

  '  9b.4. Record 13 for Right half of Terminal key
    For indx = 8 To 15          ' Right half of Terminal key
      tmpArray(indx - 8) = Asc(Mid(tmpStr, indx + 1, 1))
    Next indx
    retCode = writeRecord(0, &HD, &H8, &H8, tmpArray)
    If retCode <> SCARD_S_SUCCESS Then
      Exit Sub
    End If
  End If

  Call ClearTextFields
  Call DisplayOut(0, 0, "FF 03 is updated")

End Sub

Private Sub bInit_Click()

  sReaderList = String(255, vbNullChar)
  ReaderCount = 255

  ' 1. Establish context and obtain hContext handle
  retCode = SCardEstablishContext(SCARD_SCOPE_USER, 0, 0, hContext)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    Exit Sub
  End If

  ' 2. List PC/SC card readers installed in the system
  retCode = SCardListReaders(hContext, sReaderGroup, sReaderList, ReaderCount)
  If retCode <> SCARD_S_SUCCESS Then
    Call DisplayOut(1, retCode, "")
    Exit Sub
  End If
  Call LoadListToControl(cbReader, sReaderList)
  cbReader.ListIndex = 0

  Call AddButtons
```

```
End Sub

Private Sub bQuit_Click()

  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If
  retCode = SCardReleaseContext(hContext)
  Unload Me

End Sub

Private Sub bReset_Click()

  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If
  retCode = SCardReleaseContext(hContext)
  Call InitMenu

End Sub

Private Sub cbReader_Click()

  fSecOption.Enabled = False
  fKey.Enabled = False
  bExecMA.Enabled = False
  Call ClearTextFields
  rbDES.Value = False
  rb3DES.Value = False

  If ConnActive Then
    retCode = SCardDisconnect(hCard, SCARD_UNPOWER_CARD)
    ConnActive = False
  End If

End Sub

Private Sub Form_Load()

  Call InitMenu

End Sub

Private Sub rb3DES_Click()
```

```
  Call ClearTextFields
  tCard.MaxLength = 16
  tTerminal.MaxLength = 16

End Sub

Private Sub rbDES_Click()

  Call ClearTextFields
  tCard.MaxLength = 8
  tTerminal.MaxLength = 8

End Sub

Private Sub tCard_KeyUp(KeyCode As Integer, Shift As Integer)

  If (Len(tCard.Text) >= tCard.MaxLength) Then
    tTerminal.SetFocus
  End If
End Sub
```

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Tandoğan, Mehmet Murat
Nationality: Turkish (TC)
Date and Place of Birth: 07 February 1980, Bandırma/BALIKESİR
Marital Status: Single
Email: murat_tandogan@hotmail.com

EDUCATION

Degree	Institution	Year of Graduation
MS	Bahçeşehir University Computer Engineering	2010
BS	Bahçeşehir University Computer Engineering	2007
AS	Süleyman Demirel University Computer Programming	2001
High School	Şehit Mehmet Gönenç Lisesi, Bandırma	1997

WORK EXPERIENCE

Year	Place	Enrollment
2009	Liba Laboratuarları A.Ş.	IT Manager
2007-2009	Çözüm Holding A.Ş.	Senior Software Developer
2006	Mako A.Ş.	Intern Engineering Student
2005	Koç Sistem A.Ş.	Intern Engineering Student
2001	Starcom	Intern Engineering Student

FOREIGN LANGUAGES

Advanced English.

HOBBIES

Swimming, Diving, Playing Guitar, Camping, Trophical Aquariums, etc.