Project Number	AC095
Project Title	ASPeCT: Advanced Security for Personal Communications Technologies
Deliverable Type	P (Public)

CEC Deliverable Number	AC095/GD/W24/DS/P/11/1
Title of Deliverable	**Report on limiting smart card constraints on UIMs**
Contractual Date of Delivery to the CEC	February 1997 (Y02 / M12)
Actual Date of Delivery to the CEC	2 April 1997 (Y03 / M02)
Work packages contributing to Deliverable	WP 2.4
Nature of Deliverable	R (Report)
Editor	Eric Johnson

Abstract:

This report considers the use of smart cards as mobile personal security modules. The format of a smart card is described and the impact that this format has on both the way a smart card can be used as a security module and the limitations that the smart card imposes are discussed.

The security threats to smart cards, from physical to technological and environmental, are considered along with the various measures that are or can be adopted to prevent them.

Further sections consider:

- how the performance limitations of smart cards for cryptographic purposes may in certain circumstances, be circumvented
- the use of operating system commands for application implementation and how this may be improved by use of an interpreter concept along the lines of the Java Card API
- the required security functionality of a UIM in the UMTS system.

Keyword List:

ACTS, ASPeCT, card applications, SIM, GSM, security, smart card, UIM, UMTS

Table of Contents

Executive Summary

As a result of experience gained from the first generation analogue systems the designers of the GSM system realised that security needed to play a much larger role. To achieve this they introduced the SIM as a user security module which was based on a smart card. The use of a smart card as a SIM limited the ways in which this could be used – for example, the card could not possibly encipher the speech data at a rate of 16 kbs since that exceeded the card's I/O bandwidth.

Since the introduction of the SIM smart card technology has advanced and although there are still limitations there are now many new opportunities open for the UIM in the next generation of mobile telephony. The possibilities of multiple applications has been discussed in the ASPeCT deliverable D04 and in this document the constraints and security related issues of smart cards are considered.

The security benefits of a smart card are its principal benefit – there are other more efficient and cheaper ways of transporting user specific data than smart cards. The security of the smart card is paramount and it must be considered throughout the life cycle of the card: from its design by the semiconductor manufacturer, through the embedding, secure personalisation right through to its use by the end-user. This document considers this life cycle and the various threats that the card may be subjected to.

In the past year new attacks on electronic hardware and smart cards in particular have been presented. There has been much “hype” in the media about these attacks and that they mean that smart cards are no longer useful as secure devices [32]. One chapter of this report attempts to present a more balanced perspective and show that with the correct precautions smart cards are still acceptable to use as personal security modules. Of course, as with all security implementations it is important to remain vigilant against any possible attack and to ensure that the architecture of the system has been designed so that such an attack has minimal impact on the overall system.

An important feature of UMTS is that that it must be able to support various different authentication schemes which may have a different structure. The only user component which knows about the authentication algorithm is the UIM and the terminal must be completely standardised. This means that the UIM / terminal interface must be standardised and the constraints this imposes and the solution using either standardised commands or an interpreter are considered.

Document Control

Revision	*Date*	*Changes*
A	10 Mar 1997	Version for PMC Approval
B	25 Mar 1997	Second PMC Approval Version
1	01 Apr 1997	Issued Version

Authors

Eric Johnson	Giesecke & Devrient GmbH Prinzregentenstraße 159 D-81607 München Germany	Phone: +49 89 4119 1944 Fax: +49 89 4119 1905 X.400: c=de; a=cwmail; p=g+d; s=johnson; g=eric 100632.353@compuserve.com
Achim Müller	Giesecke & Devrient GmbH Prinzregentenstraße 159 D-81607 München Germany	Phone: +49 89 4119 1547 Fax: +49 89 4119 1540 X.400: c=de; a=cwmail; p=g+d; s=mueller; g=achim; 106340.2310@compuserve.com

References

[1] ACTS AC095, ASPeCT Deliverable D04, Report on the use of UIMs for UMTS

[2] Anderson R., & Kuhn, M. : "Tamper Resistance – a Cautionary Note", *Proceedings of the 2nd Workshop on Electronic Commerce*, available as http://www.cl.cam.ac.uk/users/rja14/ tamper.html.

[3] ANSI X3.92, "American National Standard for Data Encryption Algorithm (DEA)", American National Standards Institute, 1981.

[4] ANSI X9.17, "American National Standard for Financial Institution Key Management (Wholesale)", American National Standards Institute, 1985.

[5] ANSI X9.30, "Public Key Cryptography Using Irreversible Algorithms for the Financial Services Industry", American National Standards Institute, 1995.

[6] Biham E., & Shamir A.: , "Differential Fault Analysis of DES", Research Announcement

[7] Boneh, DeMillo & Lipton: "On the Importance of Checking Computations", available at http://www.bellcore.com

[8] Dobbertin H., Bosselaers A. & Preneel B. : "RIPEMD-160: A Strengthened Version of RIPEMD", *Fast Software Encryption,* LNCS 1039, Springer-Verlag 1996, pp.71-82.

[9] ETSI GSM 11.11, Digital cellular telecommunications system (Phase 2+); Specification for the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface

[10] FIPS Pub 180-1, "Secure Hash Standard", Federal Information Processing Standards Publication, April 1995

[11] Java Card Platforms, Gemplus, http://www.gemplus.com/press/java_2.html

[12] ISO/IEC 7816-1:1987 – Identification cards – Integrated circuit(s) cards with contacts – Part 1: Physical characteristics

[13] ISO/IEC 7816-2:1988 – Identification cards – Integrated circuit(s) cards with contacts – Part 2: Dimensions and location of contacts

[14] ISO/IEC 7816-3:1989 – Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols

[15] ISO/IEC 7816-3:1992 – Identification cards – Integrated circuit(s) cards with contacts – Part 3, Amendment 1: Protocol type T=1, asynchronous half duplex block transmission protocol

[16] ISO/IEC 7816-4:1995 – Identification cards – Integrated circuit(s) cards with contacts – Part 4: Interindustry commands for interchange.

[17] ISO/IEC WD 7816-7:– Identification cards – Integrated circuit(s) cards with contacts – Part 7: Interindustry commands for Structured Query Language (SCQL)

[18] ISO/IEC WD 7816-8:– Identification cards – Integrated circuit(s) cards with contacts – Part 8: Security architecture and related interindustry commands

[19] ISO/IEC CD 9796-2– Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Mechanisms using a hash-function

[20] ISO/IEC CD 10118-3– Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions

[21] ISO/IEC CD 14888-3– Information technology – Security techniques – Digital Signatures with appendix – Part 3: Certificate-based mechanisms

[22] Java Card: API Specification, October 1996, Sun Microsystems Inc.

[23] JavaSoft, Frequently asked Questions about Applet Security, http://www.java.sun.com/ sfaq.html

[24] Knudsen L.: "A Weakness in SAFER K-64", *Advances in Cryptology – CRYPTO'95 Proceedings*, Springer-Verlag, 1995.

[25] Lai X. & Massey J. : "A Proposal for a New Block Encryption Standard", *Advances in Cryptology – EUROCRYPT'92 Proceedings*, Springer-Verlag, 1992.

[26] Lübben B.: "FRAM der ideale Speicher ?", *Design & Elektronik* , 23/ 5 Nov. 1991

[27] Massey J.L., "SAFER K-64: A Byte-Orientated Block Ciphering Algorithm", *Fast Software Encryption*, Lecture Notes in Computer Science, No 809, New York, Springer 1994.

[28] Mitchell C.: "On the danger of separating the implementations of signatures and hash-functions", ASPeCT/REP/RHUL/2.3/067A

[29] Princeton Safe Internet Programming Team, http://www.cs.princeton.edu/sip/java-faq.html

[30] Rankl W., Effing W.: "Handbuch der Chipkarten", Carl Hanser Verlag, 2nd Edition, 1996

[31] Rivest R.L., Shamir A. & Adleman L.M. : "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, v.21, n.2, Feb. 1978, pp.120-126

[32] "Not So Smart Cards", *Scientific American*, January 1997, p19

[33] Rowley T.: "How to Break a Smart Card", 1997 RSA Data Security Conference, Proceedings

[34] Schabhüser G.: "Zur Sicherheit von Signatur-Algorithmen", *7. GMD-SmartCard Workshop Tagungsband*, January 1997.

[35] Watts A:, "EEPROMs mit einer Million Schreib-/Lesezyklen", *Elektronik* 22/1991

[36] EMV '96, Integrated Circuit Card – Specification for Payment Systems, Version 3.0, 30 June 1996

[37] Biham E., & Shamir A.: , "The next stage of Differential Fault Analysis", Research Announcement

[38] Anderson R., & Kuhn, M. : "Improved Differential Fault Analysis", Research Announcement

[39] "Draft liaison statement from ETSI STC SMG 5 to SMG 1/WPC, SMG 9", ETSI IC-Card Group, 30 May 1996, ETSI SMG 5#17, 28-29/05/96 Helsinki.

[40] Blum L., Blum M. & Shub M.: "A simple unpredictable pseudorandomnumber generator", *SIAM J Computer* 15, 1986 pp364-383

[41] *Tagespiegel*, 29 Jan 1997

[42] ACTS AC095, ASPeCT Deliverable D12, Migration Scenarios: First implementation

[43] ACTS AC095, ASPeCT Deliverable D05, Migration Scenarios

Abbreviations

BER-TLV	Basic Encoding Rules – Tag Length Value, for ASN.1 notation
DFA	Differential Fault Analysis
DSS	Digital Signature Standard
EEPROM	Electrically Erasable Programmable Read Only Memory
EPROM	Electrically Programmable Read Only Memory
EMV	Europay, Mastercard and Visa
ETSI	European Telecommunications Standards Institute
FCI	File Control Information
FRAM	Ferrroelectric Random Access Memory
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JDK	Java Developer’s Kit
LIVA	Light Induced Voltage Analysis
MT	Mobile Terminal
RAM	Random Access Memory
RFU	Reserved for Future Use
ROM	Read Only Memory
SIM	Subscriber Identity Module
SVC	Stored Value Card
UART	Universal Asynchronous Receiver Transmitter
UIM	User Identity Module
UMTS	Universal Mobile Telecommunications System
USIM	User Service Identity Module

1 Smart Cards

1.1 Introduction

The smart card was developed in the late 1970’s and made popular in France initially by Bull. Since then it has slowly expanded and following on its success in the GSM world is becoming ubiquitous due to its uptake by the banking organisations.

On one level a smart card can simply be regarded as a standard plastic banking card where the magnetic stripe has been replaced by an integrated circuit. However, this is a gross simplification and it is the advantages that a smart card offers over a magnetic stripe card that explains it success.

A magnetic stripe card can store approximately a total of 200 bytes on its 3 stripes and, although there are some mechanisms (such as magnetic watermarking and MM), there is nothing to prevent anyone changing the data stored on the stripe. If a card is only to be used as a convenient means of data transport then the smart card is not a sensible replacement to a magnetic stripe card. Optical cards, which use a technology similar to that used for compact discs, can store megabits of data at a very low cost per bit and provide security by the nature of the fact that the data cannot be erased.

The advantage offered by smart cards is that they contain a processor which can take an active rôle in providing the security of the system rather than being the passive agents that magnetic stripe cards are. In the banking world this means that the possibility of performing transactions without requiring an on-line connection becomes feasible and by reducing telephone charges allows cost savings without compromising security. In the mobile telecommunications world it allowed a personal security module to be given to every user and to use this to massively reduce fraud.

Since the 1980’s ISO has been standardising different aspects of smart cards to ensure their interoperability and relatively quickly three standards were produced detailing the physical and electrical characteristics of the cards [12,13,14].

1.2 Smart Cards in Mobile Telephony

The adoption of smart cards for the security device in GSM handsets greatly enlarged the market for smart cards and their success in this field led to their adoption in other areas.

The use of a smart card in mobile telecommunications equipment allowed the separation of the user’s security identity from the handset. This provided both enhanced security and enhanced user mobility. The first because the security of a smart card is intrinsically harder to break than that of a handset which was not designed with security in mind. The user mobility was enhanced because the smart card could be moved between different handsets with the user keeping the same identity to the network. Another feature was that it allowed the network operators to introduce their own proprietary algorithms for authentication: the only component on the user side of the network which must support these algorithms is the smart card and this is provided to the user by the network operator or his agents.

All these factors made the introduction of the smart card in the GSM a large success and quite naturally ETSI is intending to keep the concept of a user security module in the UMTS system. ETSI do not seem quite so clear what this device will be called or what form it will take [39]. The smart card in the GSM system is known as a Subscriber Identity Module or SIM and early documentation from ETSI used the expression User Identity Module or UIM but it now appears that it may be called a User Service Identity Module or USIM. The ASPeCT project is using the term UIM and so that is the term used in this document.

Despite the suggestion from ETSI it seems difficult to conceive that the UIM will be implemented on something other than a smart card. It is true that a smart card imposes certain constraints on what the UIM would be able to do but the following points should be considered:

- the manufacturing and production of smart cards in a secure environment is well established. The introduction of a new device would necessarily be more expensive than a conventional smart card.
- smart cards are now being used in many other applications, especially in the banking world. The use of a format other than a smart card would mean that the UIM could not serve as a multi-functional card and would thereby lose many market opportunities.
- it is not clear that smart cards will not be able to support most of the features required by ETSI by the time UMTS is a reality.

This document is an attempt to capture the important features that define a smart card and how these relate to their use as personal security modules. By considering these details, it should become clearer what the smart card of the future could offer and to assess whether it could be used as the UIM.

2 Hardware Features

2.1 Current Devices

The following table lists some of the smart card devices that are available from semiconductor manufacturers. The list gives details on the amount of memory that is available on such devices and the type of processor which was used as the basis of the design. The presence of an co-processor for use in public key algorithms is indicated.
Note that this table should not be seen as exhaustive but does list the major devices.

Table 2-1 : Features of Current Smart Card Processors

	Processor	ROM	EEPROM	RAM	Co-processor
Atmel					
AT89SC81	8051	8 K (Flash)	1 K	256	
AT89SC164	8051	16 K (Flash)	4 K	256	
AT89SC168	8051	16 K (Flash)	8 K	256	
Hitachi					
H8/3102	H8	16 K	8 K	512	
H8/3111	H8	12 K	8 K	800	
Motorola					
SC21	6805	6 K	3 K	128	
SC27	6805	16 K	3 4K	240	
SC28	6805	12.5 K	8 K	240	
Philips					
P83C852	8051	6 K	2 K	256	✓
P83C855	8051	20 K	2 K	512	✓
P83C858	8051	16 K	8 K	512	✓
P83C864	8051	16 K	4 K	256	
SGS Thomson					
ST16SF42	6805	16 K	2 K	384	
ST16SF44	6805	16 K	4 K	384	
ST16SF48	6805	16 K	8 K	384	
ST16SF54	6805	16 K	4 K	480	✓
Siemens					
SLE44C10S	8051	7 K	1 K	256	
SLE44C42S	8051	15 K	4 K	256	
SLE44C80S	8051	15 K	8 K	256	
SLE44CR80S	80651	15 K	8 K	256	✓

2.2 Physical & Electrical Characteristics

The physical characteristics of smart cards are described in ISO/IEC 7816 Parts 1 and 2 [12,13] and the extension to a micro-SIM is specified by ETSI in GSM 11.11 [9]. The physical dimensions and locations of the contacts are shown in Fig 2-1 and were derived from existing standards defining bank and identification cards. These physical restrictions place limitations on what hardware can be used in a smart card. The flexibility requirements and thickness are perhaps the most serious of these – the former limits the die size of the chip and the second prevents a smart card containing any type of keyboard or display (at least with existing technology).

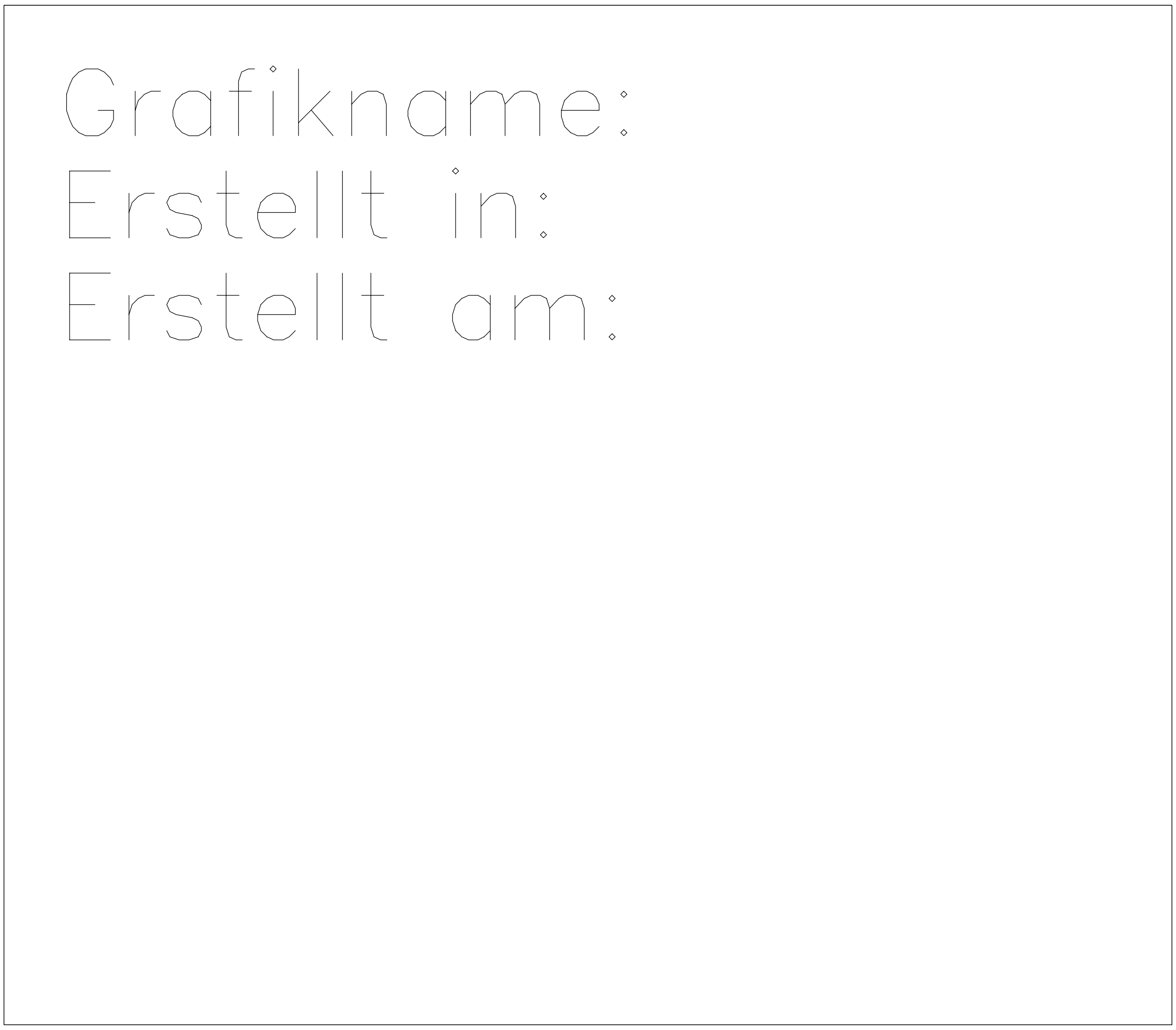


Figure 2-1, Physical Dimensions and Contacts of Smart Cards

As defined in ISO/IEC 7816-3 [14] a standard smart card has 8 contacts which are used to communicate with the external world, of these, 2 are reserved for future use and a third is no longer used on current devices. This leaves 5 contacts and whose locations are shown in Fig 2-1.

The following is a brief description of the contacts.

VCC This is the supply voltage which is defined to be either 4.5V - 5.5V for Class A devices and 2.7V - 3.3V for Class B devices. Current versions of 7816-3 only define the electrical characteristics for smart cards which operate at 5V, however, a new version of the standard is being developed which introduces the Class A and Class B nomenclature for devices which operate at a nominal voltage of 5V and 3V respectively. The initiative for this enhancement to the standard came from the mobile telecommunications industry where the need for equipment to operate at lower voltage levels is important; in the banking industry this is not an issue and the EMV [36] specifications do not currently require 3V operation. The latest CD version of 7816-3 reduces the maximum current that the smart card may use from 200 mA down to 50 mA.

GND This is the reference 0V contact.

VPP The VPP contact is almost not used on modern cards – it was originally used to supply the high voltage needed to program EPROM or EEPROM memory. Due to potential attacks on the hardware, this was replaced internally by a charge pump. The EMV specifications have even taken the step of specifying that this pin is not used and must be isolated ($\geq$10MΩ with a voltage of 5V).

RST This active low contact makes it possible to reset the processor in the smart card.

CLK This contact is the input clock for the processor. The new ISO draft specifies a minimum clock frequency of 1 MHz and a maximum of 20 MHz. Until the protocol type selection is completed the frequency must be 3.57 MHz. The EMV specifications restrict any clock drift to less than 1% during a session.

I/O This contact is used for the input and output of data from the card. All the communication between the card and the terminal must take place over this line – the communication is therefore necessarily half-duplex. The card must not be damaged if both the card and the terminal are in transmit mode and trying to hold the line at different voltage levels.

2.3 Memory

A smart card has three different kinds of memory:

- volatile memory
- non-volatile read only memory
- non-volatile read/write memory

The volatile memory is implemented as static RAM, and because it occupies a large amount of space on the die, is of limited size. It is used as working space for the processor during normal operation. Data that is stored in the RAM is lost after power is removed from the chip, thus it can only be used for ephemeral data and in fact the security of the system relies on this memory being correctly erased and a card operating system will clear the contents of the memory on power-up.

The read only non-volatile memory is the ROM area of the microcontroller which contains the executable code, data for the operating system and, perhaps, also for any applications that are loaded onto the card. This memory is very compact and so, despite the size, it does not occupy that large a portion of the die. The ROM is typically implemented using ion implantation which makes it much harder for an attacker to read. This is particularly important if any keys or confidential data tables are stored in the ROM.

The read/write non-volatile memory is used as the main area of data storage for the card. In this memory user specific data is stored – this data is different for each user and so could not be stored in ROM. It is also often used for the storage of application code if there was either not enough space in the ROM or if the application did not merit the production of a custom ROM mask. It is also possible to install patches to the main operating system code if this feature is supported by the operating system. In outdated smart cards this was implemented using EPROM technology where it provided write once memory. This was replaced by EEPROM technology as that improved. The main benefits being that it allowed the memory content to be subsequently erased and because it obviated the need for the VPP contact. Newer technology advances mean that other technologies are becoming possible such as Flash EEPROM and FRAM. The latter is probably the most exciting because it is a technology which has a write cycle similar to RAM (200 ns) and yet can maintain stored data for the same duration as EEPROM (10 years) [26].

Figure 2-2 illustrates the difference in size of the different memory components (this is only illustrative and clearly is dependent on the fabrication process used for any particular component).

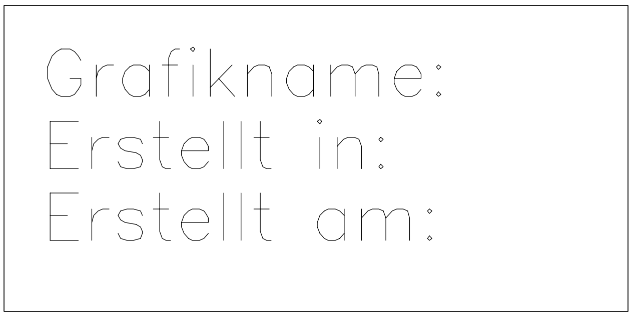


Figure 2-2, Relative Size of Memory Technologies

Other important features of non-volatile read/write memory other than its size are:

- write and erase time
- number of write / erase cycles
- data retention duration
- page size

It is difficult to compare these features for different technologies because the limitations can often be modified by extra techniques. For example, SGS Thomson have developed special EEPROM cells which use redundancy to extend the number of read/write cycles [35] and some chips from Philips used error correcting codes to achieve the same objective. However, the data in the following table gives an indication of the correct order of magnitude.

Table 2-2: Features of Memory Technologies

Technology	Write/Erase Cycles	Write/Erase+Write Time
EEPROM	10^5 -10^6	1.75 ms / 3.5 ms
Flash EEPROM	10^3 - 10^4	1.5 ms / 3.5 ms
FRAM	10^{10}	200 ns

Flash memory provides a good alternative to ROM, especially for rapid production of prototypes. It is possible to obtain components which use Flash EEPROM rather than ROM and this enables the complete application to be developed in EEPROM and used for a field test. Once the design is proved it can be implemented in ROM for lower unit price. For example, Atmel supply smart cards with up to 16 k of Flash EEPROM which can be completely programmed in 1s.

3 Security Threats and Countermeasures

This chapter considers the various measures that have been adopted in the smart card industry to try and protect against various attacks. These countermeasures have been broken down into the following categories: physical, electrical and logical security. The final section considers the new technological attacks and new security features which are being developed to enhance the security.

3.1 Physical Security

The integrated circuits that are used in smart cards are not simply cut-down versions of the standard micro-controllers that they are based on. The security requirements necessitated by the applications these components are intended for imposes many special needs which makes the design, production and handling of these devices very special.

The first significant difference in the design of a smart card arises during the hardware design stage. At this point the semiconductor manufacturer has to consider many factors which are not relevant for normal devices. These typically comprise the inclusion of extra features in the design such as voltage level and clock frequency sensors. Other features are included to make the component much harder to reverse engineer. Although manufacturer's typically use these techniques to protect their own intellectual property, for smart cards the requirement is very different. For example, the manufacturer will typically scramble the address lines and arrange the RAM so that it is not straightforward to determine which RAM cell corresponds to which bit in the logical memory space. Whilst it might seem that this is trivial it can be achieved very easily, at virtually no cost, and makes an attackers work more tedious. Spurious geometric designs, which look like functional transistors are often added to the layout with the intention of deceiving an attacker who examines the device optically. In contrast, ion implantation is often applied to regions of the silicon so that the functionality is changed even though this does not appear to be the case. Other considerations which are related to the use of the device need to be taken into account. For example, if the program code of an application remains secret then it is substantially harder to attack the system. Thus smart cards invariably use ion implantation techniques rather than fuse technology to implement ROM. This prevents the reading of the ROM by purely optical techniques.

A further important factor is the presence and functionality of test pins. A semiconductor manufacturer will typically want to be able to characterise his fabrication process and will certainly want to test the devices on a wafer before he goes to the expense of bonding them. The former is normally achieved by adding a special test die to the mask at different locations throughout the wafer but this is often supplemented by tests on actual production devices. Clearly the actual go/no-go tests must be applied to the actual production component. This typically requires the device to be able to enter a test mode which allows access deep into the internals of the chip. It is clearly essential to the design of the system that the design prevents this test mode from being available after the device tests are completed. The test mode is usually disabled by blowing a fuse or writing to an area of write once memory – whatever the method it must be irreversible, so that, for example, it is not possible to repair the fuse. If special test pads are used then they must also be disabled in a way that prevents them from being used again.

The final physical barrier employed by the semiconductor manufacturer is the addition of a passivation layer to the chip surface which makes access to the chip itself more difficult. This passivation layer can be seen in the cross-section of a smart card module shown in Fig 3-1.

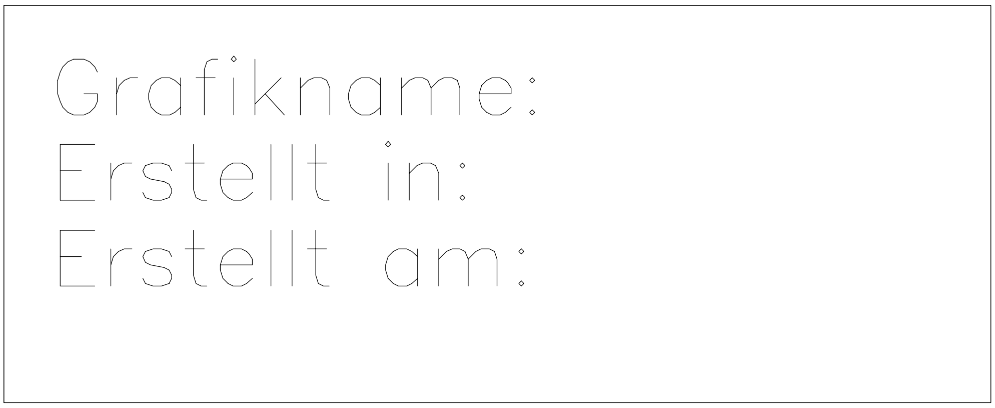


Figure 3-1, Cross-Section of Chip Module

Handling of the produced devices is also essential. If an attacker is able to obtain a supply of chips containing the ROM code of an application then he is already halfway into the process of producing clones. The semiconductor manufacturer should be able to precisely control the number of devices that he has produced and delivered to the module or card manufacturer. Throughout their life the proper handling of the chips is essential and audits can be held where the various organisations that have to handle the cards are required to prove that the numbers of cards tally and, for example, that they really have destroyed the components that they have claimed to. Clearly, this handling becomes more important the further a card progresses in the production process. This handling is assisted by the fact that each card is assigned either a serial number or a unique identity.

Figure 3-2, Production Sequence of a Smart Card

The card manufacturer may not be the same entity that actually personalises the completed smart cards. Indeed, even if they are the same entity the two steps in the process are usually logically separated so that there is less possibility for one person to have access to enough phases in the production that they could perpetrate a fraud. Between the initialisation of the cards and the personalisation cards are typically locked with a key which is derived from the card serial number or its unique identity. Without knowledge of this key it is not possible to personalise the card. This prevents a replay attack in the unlikely case that an attacker managed to obtain a supply of chips and a record of the data that was sent to cards during an earlier personalisation run. The overall production sequence of a smart card is shown in Fig 3-2.

3.2 Electrical Security

Smart cards also include a collection of electrical features to enhance their security. These are typically sensors to detect the following:
High & Low Voltage Sensors.
These sensors detect if the supply voltage to the chip has fallen outside its normal operating range and will cause a reset if this is the case. This is to prevent incorrect operation of the processor which might allow its security to be breached. For example, if the voltage drops below a threshold then the charge pump may not allow the state of EEPROM cells to be changed. In the past there have been attacks on smart card systems which manipulated the VPP voltage level – these are clearly no longer applicable since the VPP pin is no longer used.
High & Low Frequency Sensors.
These sensors detect if the clock frequency has gone outside the normal operating range and will cause a reset if this is the case. The low frequency detection ensures that the processor cannot be single stepped which may reveal internal information and would assist an attacker who is attempting to analyse the processor using contrast electron microscopy, light induced voltage analysis (LIVA) [33], or similar methods.

Rather than using sensors some devices use a phase locked loop which is locked to the external frequency but will only operate in the valid range of the chip. PLLs are also sometimes used when a supplementary arithmetic processor is included in the chip design which requires a higher clock frequency than the kernel processor.
Miscellaneous Sensors & Mechanisms
There are often other sensors on the chip which serve a security rôle. For example, the presence of the passivation layer on the chip can be verified by means of a capacitive sensor. Light sensors may also be present to detect the absence of the passivation layer.

Another electrical feature which is used is the state of EEPROM cells. If an EEPROM cell is controlling the access to some part of the chip memory then it is important that the access is only granted if the cell is in the active state. EEPROM cells are based on the Fowler-Nordheim Tunnel Effect and rely on charge being stored on a floating gate. If this charge were to leak away then the state of the cell would change and it is important that this does not reduce any access conditions to data stored on the card.

3.3 Logical Security

Other security features can only be implemented in the program code of the operating system or of applications, we denote these security features by logical security.

3.3.1 Operating System Protection of Distinct Applications

The most obvious logical security feature that must be provided by the operating system is the logical separation of distinct applications. It must not be possible for one application to access the data of another application. This protection can only be guaranteed by the operating system if all accesses to memory are handled by the operating system. This presents no problem if commands supported by the operating system are used but if it is possible for a user to load his own application code into the processor then this cannot be guaranteed. This is one of the main reasons why only the card manufacturer can load executable code onto the card. The development of operating systems with an interpreter which has received security certification (e.g., ITSEC) could change this situation and is further discussed in Chapter 6.

3.3.2 Secure Write / Integrity of Data

An important design issue in the development of code for a smart card is how to ensure the integrity of the EEPROM. During the course of a smart card command the program will almost certainly want to update the state of the EEPROM. This is not always the case but certainly after an electronic cash transaction or an incorrect PIN verification attempt the state of the card must be changed. The problem arises if the power to the card is removed before it has completed processing the command. This removal of power may be intentional, in an attempt to defraud the system, or it might be due to either an impatient customer removing his card prematurely or a fault in the terminal. In the financial sector, a transaction that is prematurely terminated in this way is called a “torn transaction” by comparison with the idea of tearing a debit or credit card voucher before the all the details have been completely printed.

It is clear that the terminal will not receive the correct status code from the card if power is removed so that the terminal knows that something has gone wrong. Ideally, in such cases, the state of the card could be returned to the state it was in before the command was sent to the card. Unfortunately, this is not always possible and the only thing that can be guaranteed is that the card is either returned to its state before the command was issued or that it is in the state it would have been in if command had been completely processed. It should certainly not be the case that the card can be left in an indeterminate state. Note that the restoration of the card to a valid state can only be carried out after the next power up of the card. It is normally either carried out during the reset phase of the card or during the processing of the first command after the reset.

The following is a procedure that explains how the integrity of the data stored in part of the EEPROM, denoted here by *S*, can be maintained using a classic semaphore approach. A separate area of EEPROM, *S’*, is used to store a shadow of *S*. In addition a flag, *F*, is used to indicate whether the data in *S* is valid (*F*=1) or the data in *S’* is valid (*F*=0).

1	$S' \leftarrow S$ $F \leftarrow 0$	*S valid*
2	Process data changing *S*	*S’ valid*
3	$F \leftarrow 1$	
4	Return from Command	*S valid*

After the next power up of the processor the following is performed:

```
if $F$=0 then
  $S \leftarrow S'$
  $F \leftarrow 1$
```

This ensures that if the processing is interrupted during Step 2 then after the next reset the contents of *S* will be reset to the value they contained in Step 1. The only problem arises if the processing is interrupted during step 4, at this point the contents of *S* has been marked valid but the terminal has not yet received the confirmation: After a reset the card still believes the transaction to be successful and although the terminal knows it to have been incomplete it does not know the current state of the card. It is not possible to solve this problem without introducing a further handshaking command and so this can only done at the application level.

Note also that the application should be structured so that no time consuming tasks need to be performed during Step 4 so that this window is as narrow as possible.

3.3.3 PIN Timing Attack

Another example where the security of the system must be controlled logically rather than through electrical methods is the verification of a PIN.

A card application that is protected by a PIN will typically have a counter to keep track of the number of consecutive incorrect PIN tries, this counter will have to be stored in EEPROM. It is important that the command does not return more quickly when the correct PIN has been entered than when an incorrect value has been given. If this were not the case then an attacker could try all the possible values for the PIN and disconnect the power from the card if it has not responded that the PIN was correct in the appropriate time – if the card had not responded then the attacker would know that the PIN was incorrect.

This example also illustrates how security problems may be solved in more than one manner. An alternative approach to the above would be to always change the PIN counter before the PIN is verified. If the verification is correct the counter can be reset, if the PIN was incorrect then the counter has already been updated. Removing the power early in this case would only have a detrimental effect.

3.4 Technological Issues

In the semiconductor world new advances in technology are made extremely rapidly. Equipment that is state-of-the-art becomes standard laboratory tools in a relatively short time. Chips that are in production necessarily use older technology and this is particularly so in the chipcard industry where the hardware has to undergo security testing before it can be approved for use in applications.
Consequently, there is a real threat that modern equipment can be used to launch attacks against smart cards. As an example the current chips used in the smart cards described in Chapter 1 are implemented using a fabrication processes supporting a relatively large geometry of 0.8-1.2µm.

There is therefore a constant battle between the techniques used by the manufacturer to secure a smart card against attack by the latest technology. It is important that the applications on a smart card are designed so that the system as a whole is not compromised if one element is compromised. Thus, for example, a smart card should not contain any global keys and should contain a signature from the card issuer. These two steps would ensure that it would only be possible to clone cards rather than to generate new cards and the risks of any compromise would be substantially lower. This in turn means that it is less profitable for an adversary to attack the system in this way and may even make the attack unviable.
Increasing the security of a system always comes at some cost and in a real world situation this cost can only be justified when contrasted to the exposed risk.

A common criticism of the processor in smart cards is that they are based on standard components of some considerable vintage. Almost all smart cards in use today are based on either the Intel 8051 or the Motorola 6805 processors as their kernel. These processors are extremely well known and are available as standard micro-controllers. If smart cards used unknown processors then it would make life harder for the attacker both to understand the new processor and to simply use a micro-controller version of the component as an emulation. The counterfeit PayTV cards distributed by pirates in the past were almost exclusively based on the micro-controllers rather than being actual smart cards.
With the increasing arrival of smart cards supporting crypto co-processors this situation is changing. There is no equivalent microcontroller which also contains a crypto unit and so emulation would only be possible with either special hardware or with a much more capable

processor. Another possible, benefit is the use of modified versions of the kernel processor which would make the timing different – this would mean that code developed for a smart card would not run at the same speed on a commercially available micro-controller. An example of this is the Siemens family of devices which use their own version of the 8051 processor with some additional instructions, memory and very different timing.

This process needs to be taken further. In the banknote printing industry the paper used to print the banknotes is very special and contains a lot of the security features of banknotes. In most countries it is a criminal offence for an individual to possess this paper and this makes counterfeiting more troublesome since the raw materials are not readily available. In the case of smart cards if the semiconductor manufacturers produced chips which could only be used in particular smart cards it would also increase the security.

Taking the analogy one step further banknotes have special features added to them such as watermarks. If a smart card also had such features they could be used to enhance the security. Such a feature would be built into the hardware of the chip and would operate at a speed which made emulation by a standard micro-controller impossible. The terminal would be able to check for the presence of this feature and, if for example, data was exchanged during this special hardware check then the correct operation of the application could be made to depend on this data.

4 Hardware Based Attacks

In the autumn of 1996 there was a flurry of information on the Internet concerning the possibility of applying environmental stresses to electronic hardware, and smart cards in particular, to cause them to make a computational error. The significance of this was that in certain circumstances the error could be used to reveal key information.

Although much information was circulated there are really only three distinct methods that were presented: the original paper from Boneh, DeMillo & Lipton at Bellcore, a note introducing Differential Fault Analysis (DFA) from Biham & Shamir, and a paper on Tamper Resistance from Anderson and Kuhn.

This chapter presents a brief introduction to these attacks and considers their relevance to smart card systems and ways in which the viability of the attacks could be prevented.

4.1 Hardware Model

The basic idea of these attacks is that it is possible to subject electronic hardware, a smart card in our case, to environmental stresses which will induce a hardware fault. The faults induced can be as simple as a gate on the device producing an incorrect result or a bit in a register flipping. The fault need not be permanent so that the hardware functions completely normally after the environmental stress is removed and it has been reset. In addition, the location of the fault need not be known. Depending on the attack in question it may be important that the number of such faults is small.

Clearly, many such induced faults will have no obvious or indeed actual effect (such as if the contents of a register changed, and then the processor stored a completely new value on top of this corrupted value) but the assumption of the attack is that we can continue until the fault causes an incorrect output to be returned from the smart card.

Some of the attacks assume that it is possible for the attacker to obtain direct access to the cryptographic algorithm on the card and thereby get the card to exactly repeat the command with and without the presence of the fault inducing stresses.

4.2 Bellcore Attack

This attack makes use of the algebraic structure of a cryptographic algorithm to reveal information when a hardware fault is induced. In its simplest form this can be used to attack an implementation of RSA which uses the Chinese Remainder Theorem.

In this case the smart card is signing a message using the secret key. The calculation can be performed more efficiently (typically by a factor of 4) by splitting it into two smaller sub-problems and then using the Chinese Remainder Theorem to combine the two results. The fault is assumed to affect the card during its calculation of only one of the two sub-problems. The elegance of this is that any fault which causes the wrong answer, during the calculation of one of the problems, is good enough – any number of small faults may occur – all that matters is that the calculation produces the wrong answer for only one of the problems.

Denote by p, q the factors of the RSA modulus and by e the public exponent. If the attacker knows the message, m, that is being signed and that due to environmental stress the resulting signature from the card is Y rather than the correct value y. Then, without loss of generality, assuming that the fault occurred during the mod q calculation, we have:

$$Y = y \bmod p \quad \text{but} \quad Y \neq y \bmod q$$

and so, with a high degree of probability

$$\gcd(m - Y^e, pq) = q$$

If, instead, the attacker does not know the full message, as would be the case if random padding is used, but the attacker can force the card to calculate the signature of the message twice, with the same padding, then there is a similar attack. In this case, it is assumed that the signature is calculated correctly as y and then the environmental stress is applied and the incorrect result Y is calculated. It is again assumed that the error occurs in only one part of the Chinese Remainder Theorem. Then with a high degree of probability the we have:

$$\gcd(y - Y, pq) = q$$

In both cases one of the factors of the public modulus has been detected and the key has been broken.

A further part of the paper describes an attack that succeeds in detecting the secret exponent using only signatures of randomly chosen messages, without requiring the correct signature of any message or the multiple signature of any message.

As pointed out in the paper these attacks can be foiled by the hardware verifying the result before releasing it. In the case of RSA the verification is usually much faster than the signature calculation so this probably could be carried out without much time penalty. An additional approach is to use random padding so that an attacker does not know the exact message that has been signed. In a discussion paper [34] for the proposed German digital signature legislation it was proposed that the smart card did not use the Chinese Remainder Theorem and did use random padding – it can only be assumed that the Bellcore attack is the reason for this recommendation.

4.3 Differential Fault Analysis

The paper by Boneh, DeMillo & Lipton was announced before it became generally available. During this time many researchers hypothesised what the technique actually was and it inspired Biham and Shamir to apply their differential cryptanalysis ideas to fault induced hardware. This result they called Differential Fault Analysis (DFA).

Unfortunately, although there have been two announcements of their results [6,37] there is no full paper so details are inevitably a bit sketchy. What is discussed here is believed to be correct based on the announcements.

In the first announcement, they assume that they can cause a single bit error in the contents of a register during the DES calculation which has affected the resulting ciphertext. A previous encipherment had provided the correct ciphertext. If the bit error occurs in one of the inputs to the S boxes during any of the last three rounds of the DES calculation then their method allows the deduction of the complete key when approximately 200 such ciphertexts are available. It is claimed that the method is still viable under more general assumptions about the fault location but no details are presented.

The second announcement is quite different in nature. Here they assume that the key of an unknown cipher is stored in the EEPROM of a smart card or security module. Further they assume that by applying environmental stress to the card the contents of this memory changes with an unequal probability. Thus for an EEPROM cell in a smart card it is more likely, under the influence of environmental stress, that the charge drains away from the gate and the value stored in the cell becomes 0 rather than it becoming 1. In their model, the effect is not reversible so as it is consistently applied all the key bits ultimately become 0.
A set of ciphertexts are collected as these bits gradually are changed to 0 and then they perform a reverse analysis constructing the key. It is claimed that this approach will determine the key of an unknown cipher system.

Anderson [38] points out that this second attack is not very realistic for smart cards since they typically contain program code and other information in EEPROM and if this gets corrupted, during the environmental stress rather than the cryptographic key, then the most likely result is that the card will crash or fail to return any output. Furthermore, he points out that the key is usually protected by some sort of checksum – a single bit error in the key will cause the checksum to be invalid and the card will indicate a memory failure rather than continuing with the cryptographic algorithm. In both these cases the attacker has merely destroyed the card and learnt nothing.

The first attack can be detected by the card if it repeats the cryptographic algorithm or reverses it and checks the result against the original input but this halves the net processing speed of the card. Maintaining a checksum on certain data blocks might help but not if the claim that the attack works under a more general model. It is also not clear how the attack would cope if more than a single bit was corrupted in a register – which could be quite likely depending on the way in which the fault is induced. Further consideration really needs to wait for the full paper to appear.

4.4 Dallas Attack

In a paper published in November, Anderson and Kuhn report on some general work on Tamper Resistance and in particular on Kuhn's attack on the Dallas DS5002 micro-controller. It should be pointed out that this attack is not appropriate to smart cards, despite some reports in the press to the contrary. This Dallas processor relies on external memory for both data and program storage. This information is protected by means of a proprietary cipher so that any data which is stored is enciphered and using bus encipherment system the address where the information is stored is also unknown to an external attacker. The method devised by Kuhn did not crack the encryption algorithm or key but did enable the attacker to load a small program which could read out all the data from the chip.
This attack is not relevant to smart cards, since they do not use such a bus encryption scheme but it does suggest that complacency is dangerous – the developers of the Dallas system never considered that the encryption could be bypassed and so did not look for such weaknesses.

4.5 Relevance of Attacks

The obvious conclusion to draw from the Bellcore and DFA attacks is that the hardware should always verify its result perhaps borrowing techniques from fault tolerant computing systems. But there are also other ways of defeating the attacks.
As pointed out above the DFA attack requires many fault induced ciphertext variants based on one common plaintext and one variant of the Bellcore CRT attack needs two ciphertexts from the same plaintext. If the system can avoid this possibility the attacks are no longer appropriate. Possible ways are the introduction of random data or sequence counters in the messages to be signed. That is the system can be protected by its internal structure. ISO does not define a command that allows arbitrary data to be signed or enciphered and most applications do not do this either. In a typical application the card will return an authentication token which contains a random generated by the card. This random in the token appears to preclude the use of DFA in this case. Similarly, if the card is enciphering response data for use in the secure messaging protocol then a different key is used for each message so once again the attack is not valid.

Anderson's suggests a modification where, using probing equipment, we intentionally change the content of the EEPROM which we know (by some means) contains the key. He argues that the idea of a checksum on the key will enable us to determine whether we have changed a bit in the key by noting if the card returns a memory error. It is interesting to note that if the key data was protected by an error correcting code which corrected the memory then this method would fail because the attacker would recieive no information on the key.

One other observation is that the DFA is a less powerful attack because it relies on the fault having a specific form. The Bellcore attack can allow a more general type of error because it takes advantage of the algebraic structure in RSA which is not present in DES.

As argued elsewhere in this report the security architecture of a smart card system should be so designed that a card only contains card specific keys. This means that the knowledge gained by an attacker would only allow a clone of that card to be made rather than new apparently valid cards. Once this cloned card is detected it should be possible to put it on black lists and thus to minimise any impact on the system of the compromise.

In conclusion, the best defence against these hardware attacks is to:

1. internally verify the result of cryptographic calculations before returning the result
2. ensure that the structure of the card commands prevent the user from knowing corresponding plaintext/ciphertext pairs. This can be achieved by ensuring the cryptographic protocols make careful use of sequence counters or random numbers.
3. ensure that compromise of an individual card does not put the entire system at risk

5 Cryptographic Algorithm Implementation

5.1 Limits

When using a smart card to implement cryptographic algorithms we are restricted by limits that are imposed due to the nature of the smart card itself. These limitations can broadly be summarised as limited I/O bandwidth, limited processing performance, and limited memory capacity and access speeds. These are considered in the following along with some other issues.

5.1.1 I/O bandwidth

A standard smart card has only one I/O pin through which all communication must be carried out. Most current smart cards contain no UART and so the I/O polling must be handled by the processor itself; leaving it no spare ability to do anything else whilst data is being input or output. This is not such a serious problem because command/response structure of the communication means that in general there is little that the processor could be doing during this time. However, if a repetitive sequence of commands must be sent to the card such as when enciphering a data stream or when calculating a hash over a data stream then this property does restrict performance. Some manufacturers produce components that are able to operate in full duplex mode but this is not standardised and is therefore only applicable in certain situations.

The second problem with the I/O is that it is so slow. The default communication speed for a smart card is 9600 baud. Recent Committee Drafts of the new version of ISO/IEC 7816-3 [15] allow the baud rate to be radically increased – with existing Siemens processors using the standard clock frequency of 3.57 MHz it is possible to transmit data at rates of up to 115 kbaud. The problem being that currently established terminals do not support this higher data rate.

If a command needs to transfer *n* bytes of data to the card then the T=0 or T=1 protocol overhead is 7 bytes on a command and 3 on the response (there is more overhead from the actual transport protocol itself – especially if channel errors cause retransmissions but we shall neglect these during the present calculations). At the standard data rate of 9600 baud each character requires 1 start bit, 8 data bits, 1 parity bit and 2 stop bits making a total of 12 bits and this requires 1.25 ms. Thus transferring *n* bytes to the card will require $10+n$ characters needing at least $12.5 + 5n/4$ ms and if we want to return the *n* processed bytes, if they had been enciphered for example, we must transfer $10+2n$ bytes which needs at least $12.5 + 5n/2$ ms.

The following table shows this for several values of *n* and also the values when a higher data rate is used.

Table 5-1: I/O Bandwidth Limitations

n	9600 baud		115000 baud	
	Tx only	Tx & Rx	Tx only	Tx & Rx
8	22.5 ms	32.5 ms	1.9 ms	2.7 ms
16	32.5 ms	52.5 ms	2.7 ms	4.4 ms
32	52.5 ms	92.5 ms	4.4 ms	7.7 ms
64	92.5 ms	172.5 ms	7.7 ms	14.4 ms

Thus it can be seen that increasing the data rate makes a tremendous difference. The situation is made even more dramatic when one considers that a DES encipherment takes approximately 12 ms and so for encipherment purposes at 9600 baud the transfer of the data takes asymptotically twice as long as the actual encipherment. This slowness means in the GSM system it was infeasible to use the SIM for encipherment purposes. If higher data rates could be achieved in a mobile phone handset then it may indeed be possible for the SIM to perform the encipherment.

5.1.2 Processing bandwidth

As pointed out elsewhere, the processors in smart cards are based on outdated 8 bit designs. These processors were designed to be micro-controllers rather than high performance devices and so it is not surprising that they are not ideally suited to performing cryptographic calculations. The situation is slowly changing, not because the processors are improving, but rather because more and more manufacturers are incorporating cryptographic co-processors into their smart card designs. These co-processors are specially designed to support algorithms based on modular arithmetic, in particular the RSA and DSA digital signature algorithms.

In principal, it is possible to develop special hardware to implement any cryptographic algorithm and incorporate it onto a smart card but this would not be economically viable unless the hardware was either very widely used or general purpose and applicable to many applications. Thus it is not sensible to implement DES in hardware for a smart card.

As pointed out in the previous section the data transfer rate can play a significant part in any real world application, therefore, the actual processing speed may not be that critical. In any application it is only important that all the processing can be performed in the time relevant to that application. For example, if a financial transaction is authorised using the DES algorithm then the 12 ms for the DES calculation will be swamped by other factors, not least of which will be the approval or PIN entry by the card holder. In such a circumstance there is no advantage to be gained by being able to perform the DES calculation in half the time, especially if such an implementation demands larger memory resources. Of course, there will be applications where the processing speed is a fundamental barrier – as was the case before the crypto co-processors were introduced.

5.1.3 Memory capacity and speed

Another limitation which smart cards impose when they are used for the implementation of cryptographic algorithms is in terms of memory capacity and the access speed of that memory.

Current smart cards all have less than 512 bytes of RAM and most less than 256 bytes. Much of this memory is reserved for internal housekeeping operations of the smart card or for buffering – although buffer memory can often be used for scratch-pad during calculations. If some algorithm requires more than the available amount of RAM then it must use EEPROM memory. This has the disadvantages that the time to write to it is very slow and it can only be used for a limited number of erase/write cycles. The most effective strategy is to move all the important elements of RAM into EEPROM, then use the RAM for the calculation and finally restore the original RAM contents from the EEPROM at the end of the calculation. This sort of technique is often important especially when public key techniques are used and the operands are large (with a 1024 bit RSA key the signature is 128 bytes long).

Algorithms can often be speeded up by unrolling the loops they contain or by using large look-up tables. These gain speed at the cost of more program code which must be stored in ROM. This is also a limited resource and when a mask is being developed it is important to decide where the trade-offs in terms of program execution speed and code size should be made. The amount of ROM on smart cards will certainly increase but as more applications are also loaded the ROM requirements will also rise. The implementation of an interpreter or Java Card will also increase the ROM requirements.

5.1.4 Random Numbers

Cryptographic algorithms or protocols often need a source of cryptographically strong random numbers. Some smart cards are equipped with hardware which can generate random numbers and these can be further processed to remove any bias that may be in the original numbers. Other smart cards without such random noise generators can only employ pseudo-random number generators because they have no source of random data. These methods rely on the strength of the random generator to prevent its internal state from becoming discovered. If it becomes possible to predict the outputs of a card's random number generator then it will be possible to compromise the security of the system. In such a system it is also very important that the pseudo-random number generator is initialised with a random seed during the initialisation of the card.

Smart card implementations to date have used algorithms based on DES, one such method is described in Annex C of ANSI X9.17 [4], or on the Blum-Blum and Shub generator when a crypto co-processor is present [40].

Note that programs which work on desktop machines often use the time between unpredictable events such as a users keyboard rate as the input to their random generators. For a chipcard this is not possible because it cannot measure anything independently and so could be defrauded by a malicious terminal if it adopted such a technique.

5.2 Implementations

This section considers some standard cryptographic algorithms and their performance on typical smart card systems.

DES This is the classic cryptographic algorithm [3]. It is extensively used in the banking sector where it is used in many general purpose roles: message authentication and authentication being the most common. It was designed to be efficient to implement in hardware rather than software.

SAFER-SK64
This is an interesting algorithm developed by Massey for the Cylink corporation [27]. It was specially developed to be efficient when run on a smart card platform. It is a relatively recent algorithm and so has also been designed to be resistant to both differential and linear cryptanalysis. Due to a perceived theoretical problem it was enhanced by Knudsen [24]. At first considerable interest in the algorithm was shown by the authorities in Singapore and they designed an extension to use a double length 128 bit key which was subsequently adopted by Massey.

IDEA This is an algorithm developed by Lai and Massey [25] and proposed for adoption as a standard cipher. The algorithm uses a 128 bit key and a combination of operations from different algebraic structures. These operations can be performed efficiently in software so that the algorithm is more efficient than DES. This algorithm is probably most famous due to its adoption in the Pretty Good Privacy program.

RSA The most famous of the public key algorithms[31]. It cannot be efficiently implemented using the processing power of a standard smart card and is the reason why crypto co-processors are becoming increasingly popular in smart cards. It is based on the difficulty of factorisation.

DSA This is the public key signature algorithm [5] developed for use in the US Digital Signature Standard. It is based on the El Gamal signature algorithm and the difficulty of the discrete log problem.

SHA-1 This is the secure hash algorithm developed by the NSA for use in the DSS. The NSA stated that it was based on the MD4 algorithm but other than that no design principles were given. It has so far withstood the attacks which have broken the MD4 and MD5 algorithms.

RIPEMD-128, RIPEMD-160
These are extensions to the RIPEMD hash function. The enhancements were made by Dobbertin, Bosselaers & Preneel [8] in response to the successful attacks on MD4 and MD5.

The following table gives an idea of the amount of code and the speed of the above algorithms when implemented on an 8051 type processor. The figures will differ if different processors are used – especially if a processor like the H8 is used with its more powerful architecture. As indicated above, the implementations aim at a compromise between code size and speed because, for general purpose smart card implementations, both are of comparable importance. Either could be improved but probably at the cost of the other.

Table 5-2: Typical Cryptographic Algorithm Performance

Algorithm	ROM	RAM	Block Size	Time[1] / Block (ms)	Time[1] / Byte (ms)
DES	1000	32	8	12	1.5
SHA-1	400	112	64	50	0.8
SAFER K64	1200	32	8	3	0.4
IDEA[2]	800	–	8	9	1.1
RSA[3] (512)	1000 ?	64	64	220	3.5
RSA[3,4] (1024)	–	–	128	450/1700	3.5 / 13.3
DSA[3,4] (512)	1000	40	64	90/170	N/A

Notes:
1. Timings are based on a clock frequency of 4.91 MHz
2. Some pre-calculation of the key required
3. Based on Siemens crypto-processor and makes use of its internal memory
4. Timings are for signature/verification

5.3 Overcoming limits

Several efforts have been made over the year to try to overcome the limitations of smart cards for use in cryptographic systems. These are only really necessary if the speed of the smart card is not high enough – either due to I/O or computational bandwidth restrictions. The simplest approach, as used in the GSM system, is simply to let another processor perform all the calculations and the smart card is just used to produce the session key. Other approaches have attempted to partition the algorithm in question so that the terminal can perform enough of the calculation to enable the smart card to complete the rest in the required time interval. Of course, it is important that the terminal does not gain any information about the secret key known by the card or could use the process to assist in an impostor attack.

One example of this is the pre-calculation of values for the computation of a DSA signature. A DSA signature calculation on a hash value h requires the calculation of

$$r = (g^k \bmod p) \bmod q$$
$$s = (k^{-1} (h + xr)) \bmod q$$

where p is a prime, q is a prime factor of p-1 with 160 bits, g is an element of order q, x is a number less than q and k is a random number less than q which is different for each signature. The signature is the pair (r, s). The user's private and public keys are x and $y=g^x$ respectively.
Since r does not depend on the message but only on the random k and the public parameters (p, q, g) a collection of (k^{-1}, r) pairs can be pre-calculated and stored on the card. Whenever the card needs to calculate a signature it uses one of these pairs and simply calculates s using 2 modular multiplications and 1 modular addition, note also that these modular calculations are mod q which only has 160 bits. This method is secure providing the card can trust the source of the (k^{-1}, r) pairs because knowing k and a signature enables the secret key to be determined. The only drawback of this approach is that each (k^{-1}, r) pair requires 40 bytes and the card will not be able to sign anything when it has used up all its pairs.

Care must be taken when attempting to partition algorithms in these sort of technique since there are many possible traps. Even calculating the hash function externally can be dangerous [28] – the solution to this is usually to perform all but the last round of the hash function externally and let the card perform the last round and then compute the signature.

6 Operating Systems & Interpreters

As is clear from its title “Interindustry commands for interchange”, ISO introduced the commands described in ISO/IEC 7816-4 [16] with the view to ensuring the interoperability of smart cards made by different manufacturers when used in developed applications.

This was a laudable aim and to a certain extent it has been realised but the standard was both too ambiguous in its specification and too late in its publication to realise its true potential. The consequences of this failure, at least with regard to the various further specification of commands has already been considered in D04 [1]. In this chapter we consider, how this idea may be, and indeed is being further evolved – by the implementation of an interpreter on the smart card.

The presence of an operating system or an interpreter on the card also allows the secure loading of an application once the card is actually in the field. The consequences and manner in which this may be achieved are also considered in this chapter.

6.1 ISO/IEC 7816-4 based Operating Systems

With the aid of the commands that have been specified in ISO/IEC 7816-4 it has been possible to develop smart cards which can be used in a relatively interoperable way. However, this interoperability is severely restricted in many ways.

The most important aspect is that a specific application could be developed and implemented by more than one card manufacturer. This is certainly possible but to ensure that problems are not encountered in normal operation of the cards a service provider must both explicitly define exactly which subset of 7816-4 he wishes to support and also perform strict certification testing on the different cards. The reason for this is that the 7816-4 allows too much flexibility in its interpretation and has only become a final standard in 1995.

The following two examples of how these factors have had affects on real world applications.

SELECT Command

Until the final draft of 7816-4 ISO had provided an option to the SELECT command which permitted the file control information (FCI) to be returned in the response to the command or not. As a result of this several applications were developed which used this feature. The final draft of the 7816-4 removed this option and defined the corresponding parameter value as being Reserved for Future Use.

The consequence of this action is that any operating system developed under the ISO draft will probably support this command option and more seriously any terminals which make use of this feature will receive an “Invalid Parameter” error if they try to use it on a card developed to be compliant to the actual standard.

The Visa Stored Value Card (SVC) is an example of an application which uses this option of the SELECT command. This means that any multi-application smart card which contains the SVC and another ISO compatible (or more likely EMV compatible) application will have to contain special code to support the feature for one application and not for the other. This inevitably leads to a more complicated application requiring more code and hence cost.

Interpretation of Le

As described earlier, for commands that expect a non-empty response the transport protocol contains a byte denoted Le which tells the card how many bytes the terminal expects in response. The exact meaning of this byte is not clear from 7816-4. The problem arises in what to do if the card would like to send back a number of bytes different to Le. There are two possibilities:

1. The Le byte tells the card that it should not send back more than the specified number of bytes.

In this case the card will truncate any response so that it has a length of exactly Le bytes. If there are fewer than Le bytes to return then the card will report an '67 00' error.

2. The Le byte tells the card that it should send back at most Le bytes.
   In this case the card will send back as much data as it likes but will report an '6C XX' error if it would like to send back more than Le bytes, the value 'XX' indicates the number of bytes that the card would like to send back.

These two possibilities arise because due to a difference in perspective. In the second case Le is being used to indicate the maximum buffer size of the terminal and in the first it is being used to indicate the maximum buffer size in the card. Again this difference in interpretation has an impact on real world applications and costs code to support both options on a multi-functional card.
Fortunately, in this case the Le=0 option means the same in both cases – the card sends back as much information as it wishes. It should be noted that the EMV specifications only allow a value of Le=0 to eliminate such problems as the one highlighted here.

As a result of problems such as these the 7816-4 standard provides a useful framework but does not ensure that compliant cards and terminals are necessarily interoperable. Real world applications which are implemented on multiple smart card platforms from different vendors are, in practice, always subject to type approval. Examples, of systems which have been implemented on multiple platforms and undergone specific testing are:

- GSM SIM cards (ETSI GSM 11.11 specification)
- German eurocheque cards (ZKA specification)
- Visa SVC cards (Visa SVC specification)

It is also interesting to note that none of these applications is truly ISO compatible, indeed the German eurocheque cards is closer to the ETSI type of file system than it is to ISO.

Despite the shortcomings of ISO/IEC 7816-4 it is still possible to develop many applications using the command that it provides. The nature of these applications is limited in scope by the command set provided by the standard. Typical applications include access control, user authentication, and secure key or data transport. The flow of a typical application would be as shown below:

1	Select Application
2	Read Record
3	Verify
4	Get Challenge
5	External Authenticate
6	Read / Write Record

The VERIFY command allows the user to enter his PIN which authorises the card to allow further access to the keys and files on the card. In the example, this permits an authentication of the external (to the card) entity, who can then read and write to particular files in the card provided the authentication was successful. Note that although the above type of application can be implemented using standardised commands there is no standardised way to define the access requirements of the different files – these issues have always been card manufacturer specific. This is yet another reason that the application provider must fully specify the functionality he requires from the card.

If an application needs functionality additional to that provided by the commands in the 7816-4 specification then they will need to be defined and implemented on each card that must support them. The executable code for the command need not be in ROM and for applications with a small volume the code is invariably loaded in EEPROM.

ISO/IEC is currently producing a new part to 7816, part 8 [18], which will define interoperable commands for cryptographic use.

6.2 Smart Card Interpreter

The idea of implementing an interpreter on a smart card is not a new one and the various banking organisations have been looking at the idea since they began considering smart cards as the successor to magnetic stripe cards in the late 1980s. On a related topic there are smart cards which implement a simplified version of the standardised SQL database query language which is in itself an interpreted system. These SCQL commands are currently being specified by ISO as a further part to 7816 [17]. Yet another part to 7816 is defining a security architecture and further commands specifically for the support of cryptographic functionality [18].

There are many reasons why it is desirable to require a smart card to support an interpreter, but, of course, there are also some drawbacks. The following two sections highlight those that are the most obvious.
Advantages

- *Platform Neutrality*. Application will run on more than one platform. This in turn means that the application provider can easily use a variety of different card suppliers
- *Fast prototyping and time to market*. New applications can be rapidly developed and run on the interpreter. This prototype can be a much larger, and consequently more expensive, component than necessary. The prototype implementation allows the application's concept to be proved and field trials to be conducted. By the time the application is ready for the mass market a ROM version of the application can be developed for a smaller chip and provide the necessary cost savings.
- *Unambiguous Definition*. The interpreter can have a precise definition and a reference emulation which removes the ambiguity in ISO/IEC 7816.
- *Delayed Loading of Applications*. When combined with a suitable security architecture could allow applications to be loaded after the card has been distributed to the end users. This would allow even the over-air loading of applications in a mobile telecommunications environment.
- *Multiple Applications*. Allows for a security architecture which prevents the different applications on a multi-application card from interfering. That is, the interpreter can ensure that an application is not able to access the memory space of a different application.
- *Security*. If the security of the underlying interpreter has been certified then it should be much easier to obtain certification of an application written in for the interpreter than for the equivalent application when written in assembler. Furthermore, there is only one application implementation written in the interpreted language and this need only be certified once. An assembler based approach requires each different platform implementation to be certified.

6.2.1 Disadvantages

- *Cost*. An interpreter will necessarily be expensive in terms of the processor power and memory that it requires when compared against a hand-crafted assembly program.
- *Speed*. An interpreted program will run more slowly than its assembler counterpart. However, the interpreter will only be controlling the flow of the program rather than implementing the fundamental algorithms which must still be written in assembler and supplied by the operating system.

- *Security*. By allowing applications to be loaded later on the card their is the possible risk that a later application may compromise the security of an application already loaded. The correctness of the security manager becomes critical in protecting the different applications. Currently, an application provider can be certain which applications will co-reside on the card and hence positively vet them.
- *Complexity*. The interpreter concept is a more complicated system than the use of ISO/IEC 7816-4 type commands. This makes it harder to prove the security of the system. Such a proof is vital before the later loading of applications becomes possible.

6.3 Java Card

On the 12 February 1997 Gemplus and Schlumberger announced the introduction of compatible smart cards which support the Java Card standard [11] (JCOS1 and Cyberflex respectively). These cards are intended to support the Java Card API [22] which was developed by Schlumberger and endorsed by JavaSoft in 1996.
The announcement of such a product fills a perceived need by customers but, as always, the strengths have been overstated and the weaknesses understated. This section attempts to describe the system and discuss its applicability.
An important benefit claimed by Java Card is that it inherits the security model of Java which allows applications to be executed even if they are from an untrusted source. This claim is investigated below.

6.3.1 Java

Java is a platform independent language that was developed by Sun, originally intended for use in developing code for embedded consumer electronic applications. After the rapid growth of the Internet it was re-targeted and substantially redesigned to become the language known today. The language design is based on object oriented principles; it was modelled on C++ but was intended to overcome many of the weaknesses of that language. Through intellectual property rights Sun maintain control of the Java specification but this is to control the development of the language preventing the introduction of platform specific features.

Right from the start the designers of Java considered security issues which meant that these features were not simply bolted on as an afterthought but form part of the cohesive whole of the language. This is important and the security of Java has often been stressed. In the Internet domain this may well be true and, in contrast, to the security weaknesses of Microsoft's ActiveX as revealed recently by the "Chaos Computer Club" [41] Java does appear relatively secure. However, to be used in smart cards for authentication and banking applications where a user is legally responsible for transactions carried out by his card we need something stronger.

One of Java's principal strengths is its architectural neutrality – this is its ability to run on any platform which supports a Java interpreter. A Java program is compiled into an intermediate form called *Byte-Code.* This byte-code is eventually executed by a Java Virtual Machine and it is this latter component which must be implemented for a platform to support Java.

When a user visits a web site which contains a Java application the byte-code is downloaded to the local web browser and then processed by the *Byte-Code Verifier*. This is essentially a theorem prover which parses the byte-code and performs a data-flow analysis to ensure that it meets certain criteria. If the byte-code is rejected by the verifier then it will not be interpreted by the virtual machine. The verifier ensures at least the following

- there are no stack overflows or underflows
- all register accesses and stores are valid
- parameters to all byte-code instructions are correct
- there is no illegal data conversion

- there are no forged pointers
- there are no access violations

The byte-code verifier does not assume anything about the source of the byte-code, but any byte-code produced by the compilation of a valid Java program will pass the checks. One aim of the verifier is to ensure that the byte-code executed by the interpreter should not cause any problems to the running of the interpreter. Knowing that the byte-code has been verified allows the virtual machine to execute the code without needing to perform run time checks and thus run faster.

After verification the byte-code data is instantiated as a new class by the *Class Loader*. The class loader is responsible for ensuring that the name space of the code is correctly managed. This prevents new methods which are loaded from overriding methods in built-in classes.

Once loaded the class can be executed by the virtual machine. During execution the various Java libraries consult the *Security Manager* whenever the virtual machine is about to invoke a dangerous method. Examples of dangerous methods are those that request file or network access or those that define new class loaders. The Security Manager makes its decision based on which class loader loaded the class and the requesting class. If the Security Manager does not approve the request then it raises a security exception.
Figure 6-1 shows an overview of the whole process.

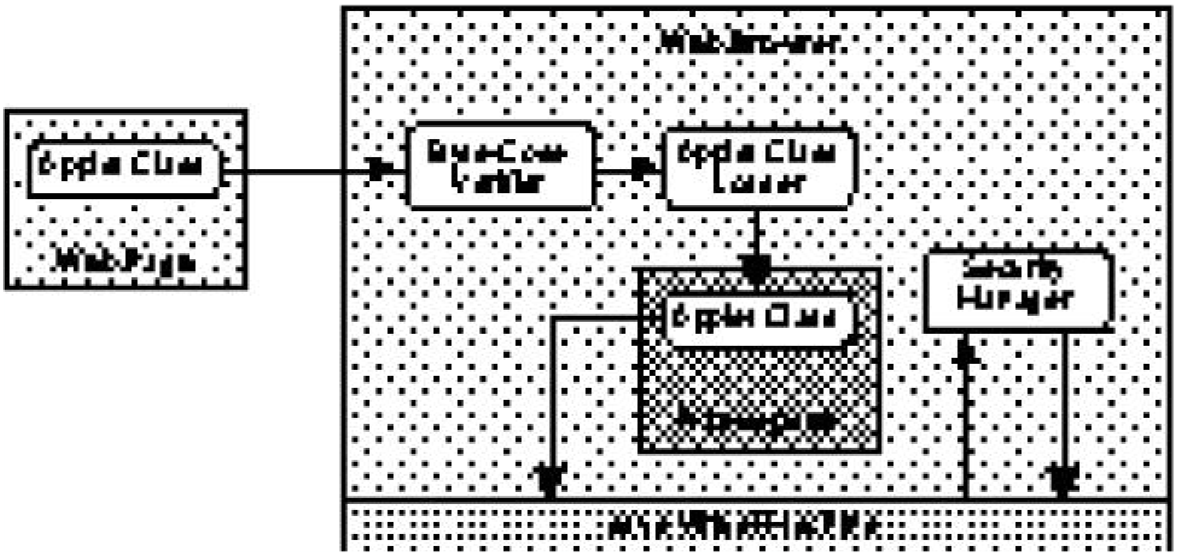


Figure 6-1, Java Security Model

It is common to refer to two types of Java programs: *applets* and *applications*. Applets are Java programs which are run from inside a web browser and are typically loaded over the network. In contrast applications are standalone programs run by directly invoking the Java interpreter and are normally locally stored. Applications are not passed though the byte-code verifier because they are assumed to come from a trusted source; or at least because they are local have been previously verified. Due to their different origin they are loaded by different Class Loaders and thus the Security Manager can apply different criteria when assessing a request. Usually an application will be granted more privileges than an applet because it has a more trustworthy source.

The security model in Java is being further refined –the latest release of the Java Developer's Kit (JDK 1.1, released on February 18, 1997) incorporates a digital signature. This signature must be verified before an applet is allowed certain privileges. This provides at least knowledge of who supplied the applet but does not prove that it is safe to run. More enhancements are planned for later in 1997.

To date there have been several Java security problems uncovered and publicised by the Princeton Safe Internet Programming Team [29], these have been corrected in the subsequent updates to the JDK or in the browser implementation. Some of these problems have been due to bugs in the Byte-Code Verifier or the Class Loader [23].

6.3.2 Java on a Smart Card

What exactly does it mean to implement Java on a smart card? Java was clearly intended to run on a 32 bit processor and most of the applications assume that there is a graphical user interface and a sockets connection. – these are clearly not appropriate for a smart card which only supports a standard file system and a simple I/O channel.
Unfortunately, the Java Card API is very general on details. It states that the design criteria required the following hardware as a minimum:

- 300 KIP CPU
- 12 K ROM
- 4 K EEPROM
- 512 byte RAM

Furthermore, the Java Card API supports

- boolean, byte and short data type
- all object-oriented scope and binding rules
- all flow-of-control statements
- all operators and modifiers
- unidimensional arrays of supported data types

A list of all the supported byte-codes is given.

The API also presents the `java.iso7816` package as a class library which supports, or at least defines, all the standard ISO/IEC 7816-4 commands. Exceptions are not supported and the card simply aborts the command returning an error code as defined in 7816-4.
The document claims that any standard Java compiler can produce Java Card programs. These must be verified externally because there is no byte-code verifier present on the card. This means that the Java Card must rely on a code signature before it accepts code from an external source. It is claimed that the virtual machine performs the same runtime checks as any other Java virtual machine.

Due to the fact that the byte-codes supported by the Java Card form a subset of the normal Java Card it would appear to be necessary to use a non-standard byte-code verifier or else the virtual machine could receive invalid code. A programmer must also use some care when developing an application that he does not use some of the non-supported features (such as 32 bit operands or arrays of user defined types) – if a standard compiler is used the error would only be detected during byte-code verification because these constructs are perfectly valid Java. Unless these observations are incorrect it seems that many of the claimed advantages of Java Card such as the ready availability of Java development tools and Java programmers may not be quite so convincing as they first seem.

Another question is that it is unclear is that if the `java.iso7816` is the only class library available then what exactly has been gained over a card that implements the standard ISO/IEC 7816-4 commands? Presumably the answer to this is the ability to write new commands using the subset of Java that comprises Java Card. For this approach to be profitable there must be other standard classes defined – for example, an object type that is tearing resistant or an object type that is specially protected by a checksum. Whilst this functionality could be implemented in Java it would be more efficient if it is implemented by the underlying operating system and presented to the application developer using the Java interface. Similarly it is probably not sensible to implement cryptographic algorithms in Java rather than assembler on grounds of both code size and performance, thus there needs to be a class interface which enables native methods for code such as DES and hash function calculations. In order to support export regulations the security manager would probably allow only code signed by the manufacturer to have explicit access to the encryption functions, a normal user would only be allowed access by a class which supported, for example, authentication mechanisms.

Since a Java Card does not actually support the Java language but rather the subset of the byte-code generated by a Java compiler which is supported by the card's virtual machine, the obvious question to ask is whether this is the best choice of interpreter for a smart card. For the present this question must remain open. However, the fact that the Java Card virtual machine has now been standardised and is based on the proven technology of Java means that it certainly provides a good starting point for a smart card interpreter concept. The ready availability of Java also suggests that it will be straightforward to implement an emulation of a Java Card on a desktop computer and allow the development and debugging of applications before they are loaded onto the smartcard – this should certainly reduce application development times.

On a practical note the realisation of the Java Card specification is certainly possible. Schlumberger have released the following performance figures for their prototype:

Table 6-1 : Performance of Schlumberger Java Interpreter

	Java Interpreter	Smart Card Primitives
ROM	4K	8K
RAM	200 bytes	90 bytes
CPU	1.5 K codes/s	300 K/s

As described above there is a difference between a Java application and a Java applet. What would this difference mean to a Java Card? One possibility would be that an application is either code in the ROM or is code that has been loaded into the EEPROM after verification of an attached signature. The card could insist that this signature belongs to either the card issuer or perhaps additionally the card supplier. Such code would have high privileges and be able to perform almost anything (except accessing data outside its application space). Applets could be unsigned code that is provided by a terminal and would have low privileges and thus be limited in what they do but could allow enhanced flexibility to applications.

On desktop machines the execution speed of Java has been dramatically improved by the introduction of Just in time (JIT) compilers. These have been claimed to achieve up to 50% of the speed of native code by caching the converted byte-code and thus preventing repeated interpretation of the same code in loops. Such a system would be too complicated for a memory constrained smart card and so are inappropriate to Java Card.

7 Security related card commands

With respect to a transition from today’s telecommunication systems like GSM to new ones already emerging at the horizon, smart cards are faced with new and changing requirements. This holds in particular for the design of security related card commands which are used to allow access to services. In this chapter we will sketch what these changes may look like and analyse their impact from a “What should be” perspective – we shall not concentrate on implementation details which are outside the scope of this chapter.

From a general point of view, this chapter is concerned with the design of card commands. We look at the GSM and UMTS as examples of telecommunication systems and outline what future card commands may look like. It is important to consider the design of card commands together with the constraints that the card imposes because this allows one to close the gap between the available technology and the requirements of applications

The bulk of this chapter comprises two sections: the first considers authentication protocols with regard to the requirements they place on the processing that must take place in the smart card and the constraints that the smart card could place on such protocols. The second considers the functionality that the card must contain to support such authentication mechanisms. The final section considers some issues relevant for the interoperability of the smart cards in UMTS.

7.1 Authentication protocols in mobile communication systems

If a smart card is to be used to support the implementation of an authentication protocol in a mobile communications system then it is clear that the architecture of the system will impose certain requirements on the smart card. This section attempts to analyse these requirements by means of a concrete example, comparing the authentication mechanism in the GSM system with a proposed authentication mechanism for the UMTS system. The intention of this is to bring out the characteristic functionality which is needed by a general authentication framework.
The UMTS authentication mechanism which is described can be regarded as a high level overview of the authentication mechanism being implemented by ASPeCT for its migration path demonstration.

7.1.1 The Global System for Mobile Communication (GSM)

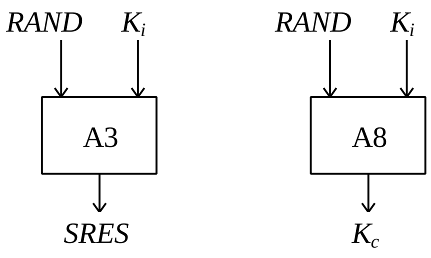
In the GSM system, there is only one command RUN GSM that is used to authenticate a mobile user. The command structure itself is standardised: this allows proprietary features to be hidden inside the smart card and does not require any intelligent action by the mobile terminal – except the fact that the terminal must send the command.

In the T=0 protocol, the situation looks like follows:

COMMAND	CLA	INS	P1	P2	P3	DATA
RUN GSM	`A0`	`88`	`00`	`00`	`10`	*RAND*

For the RUN GSM command, the data part consists of a 128 bit random number, *RAND*, that was sent from the network to the mobile terminal and will be used as input to a pair of algorithms called A3 and A8. These algorithms are implemented in software in the SIM and depend on choices made by the network operator – since the way both sides compute an answer for some input is not standardised.

In case of the SIM, the answer to the *RAND* challenge is a pair (*SRES*, K_c) : both values are computed by the smart card using *RAND* and a 128 bit key K_i as input for the A3 or A8 respectively.



The *SRES* value (32 bits) produced by the SIM is compared by the network with the one expected and used to grant access to the GSM system if they match. Because *RAND* is non-predictable and K_i is unique and secret, only the SIM of the user associated with K_i can have computed *SRES*. The second value K_c (64 bits) is a key subsequently employed by the A5 algorithm to encipher speech over the air interface.

From a mobile terminal's point of view, the above authentication protocol is as easy as it could be: it does not have to care about any details related to authentication. The characteristics are:

- the whole authentication protocol is embedded in one command-response pair
- the coding of the command and the length of parameters are always the same
- the mobile terminal does not know anything about the authentication algorithm

From a card's point of view, the situation is a bit different: the A3 or A8 implemented on a particular SIM depends on a choice between the MoU version or the network operator's version of the algorithms. The key point is that the functional behaviour of the SIM stays the same, even though the numerical values computed by the SIM differ.

The authentication of mobile user's in GSM can therefore be described as follows:

- the mobile terminal acts as a transparent device between network and SIM
- the SIM has protocol dependent knowledge : it contains the A3/8 algorithms
- the mobile terminal does not know anything about the authentication algorithm

The requirements on the smart card posed by GSM are quite stable: the implementations of the A3/8 cost some number of bytes which may vary slightly between different types of microprocessors and of course depend on the choice between MoU or the network operator's version, but these have been proved not to be a major problem compared with today's level of card technology available.

7.1.2 The Universal Mobile Telecommunications System (UMTS)

For UMTS, there are several proposals for authentication of mobile users. It is not clear yet which one will head the race. One fact seems to be stable nevertheless: they are all much more complex than the one used for GSM. To illustrate this, we now describe briefly one of the proposed protocols (the one we choose was developed by Siemens AG). Although a specific protocol is discussed it is only one of a family of protocols which could be used within the UMTS authentication framework.

The authentication begins with a negotiation phase in which the network operator and the user must agree on some protocol to use for authentication. This set-up is common to all protocols which are supported by UMTS and must always be performed before some subsequent action can be started.

Although the following illustrations have been published in deliverable D05 [43], we briefly repeat them for the sake of completeness.

7.1.2.1 The UMTS authentication framework

As the first step, the user sends a message called InitAuthRqt to the network which tells the network operator the identity of the user's service provider and an authentication capability class:

$$\text{user} \xrightarrow{\text{SpId , AuthCapCl}} \text{network}$$

The authentication capability class classifies users according to their authentication capabilities. Based on this information, the network can ask the user's service provider about the detailed capabilities of a user applying for registration – only the service provider has sufficient knowledge to interpret the AuthCapCl value.

When the network operator receives this information from the service provider, it can make a decision which mechanism to use for that particular user. This decision may be influenced by different factors: the capabilities of the network itself, unwanted algorithms or other factors. The choice made by the network is presented to the user who can accept it or decline to use that mechanism:

$$\text{user} \xleftarrow{\text{PresAuthMech}} \text{network}$$

$$\text{user} \xrightarrow{\text{Yes / No}} \text{network}$$

For the following text, we assume that the Siemens' protocol outlined below has been negotiated.

7.1.2.2 Authentication of mobile user's not yet registered – new registrations

To authenticate an unknown user, the network expects that this user sends it a message containing some random challenge and the identity (or a list of identities) of a certification authority (some trusted third party that issues certificates on public keys).

$$\text{User} \xrightarrow{g^{RND\_U}\text{ , id\_CA}} \text{network}$$

The challenge g^{RND_U} is a multiple of some point on an elliptic curve – the particular curve and basepoint g used must be common between network operator and user. The manner in which these parameters is negotiated will be explained later; note, however, that the ASPeCT demonstrator will use fixed values. The network now responds to the user with a second message which is more complex: it includes some temporary identity (encrypted) allocated for that mobile user and a certificate on the network operator's public key agreement key. This key was certified by some certification authority known to the user and will be used to set up a session key shared between network operator and the user for the current session.

$$\text{User} \xleftarrow{RND_N\text{ , }AUTH_N\text{ , Enc}(K_S,\text{data1})\text{ , CertN}} \text{network}$$

The random challenge RND_N and the authentication token $AUTH_N$ will be explained later. Finally, the user sends a third message to the network operator which contains a certificate on a public signature verification key for that user

$$\text{User} \xrightarrow{\text{Enc}(K_S,\text{Sign}_U(h(K_S||\text{data1})))\text{ , Enc}(K_S,\text{CertU})} \text{network}$$

The description of the protocol given so far is based on a network operator's point of view: the user is a logical entity. How it is physically implemented doesn't matter – as long as the physical user behaves like the logical one.

7.1.2.3 Mapping network messages to card commands

In the real world, a user is represented by its UIM (UMTS SIM). The type of interaction between network and user is further limited by the fact that half of the actual communication takes place between the mobile terminal and smart card plugged into the terminal. The above messages must therefore be mapped to sequences of card commands that can be processed by the UIM. The criteria for the design of these commands are as follows:

- there must not be any protocol dependent knowledge in the mobile terminal
- the UIM must tell the mobile terminal exactly how to execute the current protocol

The term current protocol reflects the fact that there may be a different authentication mechanism for every UMTS network operator or even service provider that issues UIMs. In order to be usable with every UMTS network, a mobile terminal must be able to cope with this diversity and therefore not depend on any particular authentication protocol.

At first sight, this may look like squaring the circle. How can the card tell the terminal what to do or even which card command to issue – that seems to be a reversal of roles. The solution lies in the idea of an interpreter implemented in the mobile terminal that reads out a descriptor for the current protocol stored in the UIM and acts accordingly.

We conclude this introduction with a detailed description of the card commands necessary for the UMTS authentication framework and the authentication protocol proposed by Siemens AG.

7.1.2.4 Card commands supporting the negotiation phase of the authentication framework

The first step to be taken by the mobile terminal is to read out the authentication capability class and the user's service provider identity which are stored in files EF_{ACCL} and EF_{SPID} respectively. This can be done with two ordinary Read commands:

Authentication Capability Class = ReadBinary(File_ID[EF_{ACCL}])
Service Provider Identity = ReadBinary(File_ID[EF_{SPID}])

The mobile terminal transparently forwards this information to the network operator. When an identifier for some proposed authentication mechanism comes back from the network, the terminal sends it to the UIM:

Status = SetAuthMechism(AuthMechID)

Possible reactions of the card may be:

- authentication mechanism unknown
- authentication mechanism not supported
- authentication mechanism not accepted
- authentication mechanism accepted

In case of a successful negotiation, the selected authentication protocol can be started.

7.1.2.5 Card commands supporting the Siemens' authentication protocol

For the Siemens' protocol, the mobile terminal must first read out the identity of some certification authority the UIM accepts for signature verification: this is crucial because the network operator does not know anything about the UIM at the beginning of the communication except for the authentication capability class and the service provider identity.

It must therefore find out which certification authorities are accepted by the card and which ones not. And it must look for a certificate on its own public key agreement key issued by one of those accepted CAs.

This can be achieved by reading a card file EF_{CA} that contains a list of identities for certification authorities known to the UIM. The first card command to be issued is once again a simple read:

List of Identities = ReadBinary(File_ID[EF_{CA}])

The mobile terminal sends this list of identities transparently to the network operator.

With this information, the network knows what to look for: a certificate on its own public key agreement key signed by one of the CAs known to the UIM. This certificate is denoted by CertN and must have a well-defined structure: this is important because the card will analyse it and extract the different data fields inside it. Well-defined means that some sort of BER-TLV encoding with defined tag-values is used. This allows to recognise which data fields in the certificate are present and which ones not. This BER-TLV approach includes structured fields and only requires some commonalties on the lowest level. An example of such a certificate is that being used in the ASPeCT demonstrator in D12 [42].

If the network operator can provide such a certificate, it sends it to the mobile terminal which stores it in some file EF_{CertN} in the UIM with an Update command:

UpdateBinary(File_ID[EF_{CertN}], CertN)

At this point the user represented by his UIM and the network operator have a common basis to start the authentication protocol: this basis consists of the particular elliptic curve and some cyclic subgroup of points (given by its generator) to be used for the random challenges by both sides.

It is up to the card to decide whether it wants to accept the certificate or to reject it: this decision is initiated by the mobile terminal issuing a VerifyCertificate command. This command is sent immediately after the certificate has been stored and does not require any further interaction between mobile terminal and network. It is simply the second part of an atomic sequence of card commands that consists of storing, verification and the challenge which is described in a few lines:

Status = VerifyCertificate(File_ID[EF_{CertN}])

The answer produced by the card is a YES or a NO. In case of NO, the UIM will reject any further commands which require it to use the public key in the certificate. The card will only process such commands when it has a successfully verified the networks operator’s certificate. The identifier of the CertN file must be included because different protocols may need to verify different certificates.

What has been achieved in case of YES is that both UIM and network operator know which group to use for the key agreement. Based on this knowledge, the card computes a challenge g^{RND_U} for some random number RND_U and sends it to the network. To facilitate this, the mobile terminal issues a GetChallenge command (without interaction by the network):

Challenge = GetChallenge( )

The challenge produced by the card is transparently forwarded to the network by the mobile terminal. With the challenge and its private key agreement key, the network operator can compute a session key K_S shared with the UIM and an authentication token $AUTH_N$ which is a hash-code of the session key. Together with a random number RND_N and an encrypted temporary identifier allocated for the mobile user applying for registration (embedded in data1), the message sent from the network operator to the mobile terminal looks like:

$$RND_N \,||\, AUTH_N \,||\, Enc(K_S,data1)$$

The mobile terminal does not process this message – it takes it as the data part of a complex card command which is called MutualAuthenticate:

$$\textit{Authentication Token} = \text{MutualAuthenticate}(\, RND_N \,||\, AUTH_N \,||\, Enc(K_S,data1)\,)$$

where

$$\textit{Authentication Token} := Enc(K_S, Sig(K_S, data1))$$

It would be desirable to include the third message sent from the user to the network operator in the response part of the MutualAuthenticate command. This is unfortunately impossible in the T=0 protocol because, in practice, the message length is limited to 255 bytes and certificates are usually longer. For this reason, the data returned only contains the encrypted signature on data1.

The certificate on the user's public signature verification key is read out by the mobile terminal with a second command in encrypted form and transparently forwarded to the network by the MT:

$$\textit{Encrypted Certificate} = \text{ReadCertificate}(\,)$$

All together, it should be clear that the requirements placed on smart cards to support such a complex protocol are much more demanding than the ones posed by GSM. Nonetheless, the newer cards that are becoming available and which include a hardware arithmetic processor are capable of supporting such a mechanism.

7.2 Security relevant functionality of the UIM

In this section we consider the protocol from a different perspective: the functional blocks required by the card are considered together with the way they are combined to form card commands or can be used internally by the card's operating system.

The situation is similar to a puzzle: the parts are easy to survey because their functional behaviour is well defined. A good example for such a component are digital signatures: a message M is signed with a private signature key and the output of the signature process is the signed message (M, Σ). Σ denotes the signature on M. Another example is a block that encrypts or decrypts messages.

7.2.1 Security functions implemented in software

We now explain in detail which functional blocks must be implemented in the UIM to support the UMTS authentication framework and the Siemens' authentication protocol. This list is by no means exhaustive – different protocols may lead to different lists. Which problems arise with respect to protocol independence in the mobile terminal will be discussed at the end of the section.

7.2.1.1 Exponentiation in finite abelian groups

A very fundamental operation is to compute powers of an element g in a finite abelian group G. In a smart card context, point groups of elliptic curves are particularly interesting because the timing and storage requirements are not so tight as with other popular groups. These groups will be used for the AMV signature scheme and key agreement. They are not suitable for RSA.

7.2.1.1.1 Rational points on elliptic curves

The underlying elliptic curve is given in Weierstraß normal form $Y^2=X^3+aX+b$ and rational points on this curve are represented by their affine co-ordinates (x,y). According to the terms used throughout the literature, the group operation is called the sum of two points.

From a functional point of view, we define the corresponding module as follows:

POINT := MULTIPLE(FACTOR, POINT, CURVE);

The factor is some natural number that should not be larger than the order of the point on the given curve. How this function is evaluated depends heavily on the implementation. There are several ways to optimise the computations:

- using projective co-ordinates as internal representations
- point compression to save memory
- fast algorithms for computing multiples of point (addition-subtraction method)

7.2.1.1.2 Modular arithmetic

For the RSA signature scheme, we need modular exponentiation and reduction combined with chinese remaindering. Both operations are usually supported by special on-chip hardware which gives acceptable timing estimates.

7.2.1.2 Digital Signatures

The operating system of the UIM must be able to sign a given message or to verify signatures produced by some other entity. We use two different signature processes which are being standardised:

- RSA according to ISO/IEC CD 9796-2: Digital signature schemes giving message recovery – Part 2: Mechanisms using a hash-function (Annex A) [19]
- AMV according to ISO/IEC CD 14888-3: Digital signatures with appendix - Part 3: Certificate-based mechanisms (Annex A.2.2) [21]

7.2.1.2.1 Signature generation

From a functional point of view, we have:

SIGNATURE := SIGN(MECHANISM, PARAMETERS, MESSAGE);

The particular signature mechanism to be used is selected together with the necessary parameters – an AMV signature for example depends on an elliptic curve together with some point on it and additional information. The message is the data string to be signed. As the result of the function call, a signature on M comes back. For AMV, we have a pair (R,S) of integers but for RSA only one integer.

7.2.1.2.2 Signature verification

The function definition looks like:

TEST := VERIFY(MECHANISM, PARAMETERS, MESSAGE, SIGATURE);

Compared with the generation of signatures, the only difference is that the signature on the message must be presented in order to verify its validity. If the signature cannot be verified, the TEST value is set to WRONG and otherwise to OK.

7.2.1.3 Hash Codes

In the course of the Siemens' protocol, a session key is generated as the hash code of some message. This shows that hash functions are not only employed in connection with digital signatures. The output of such a function is either 128 or 160 bits long. Typical candidates are the RIDPE-MD family or SHA-1 according to ISO/IEC 10118-3 [20].

7.2.1.4 Encryption and Decryption

Another building block is a block cipher either in ECB or CBC mode. For smart cards, DES or Triple-DES is probably the best choice:

MESSAGE’ := ENCRYPT(ALGORITHM, PARAMETERS, MESSAGE);

or

MESSAGE := DECRYPT(ALGORITHM, PARAMETERS, MESSAGE’);

7.2.1.5 Certificates on public keys

Two types of certificates are necessary:

- external certificates: on a public key of some other entity (network operator for example)
- user certificates: on a public key issued for the user

The major difference between the two is that the second is permanently stored in the UIM but the first one only for the current session (it is a temporary one) during the authentication. The data fields included in certificates are described in the X.509 series of standards.

7.2.1.6 Key management

Last but not least, the management of keys is a functional block that defines how much security can really be achieved. From a low level perspective, every key is some piece of data that can be classified as follows:

- stored permanently or temporarily in the UIM
- secret or public

The idea of an access condition is basically the successful presentation of some secret only known to the cardholder. This may be some PIN, a biometric pattern or a key used in a mutual authentication between smart card and card reader. Such an access condition defines a level (state of the card) that can only be reached after a successful presentation of the secret.

For each card command, it is possible to allocate an access condition that protects data processed by that command from disclosure or manipulation. This holds in particular for READ or WRITE commands which could be used to read out a key.

The appropriate selection of access conditions for different card commands is the crucial point in the security of the whole card. If it is possible by whatever way to cheat the card’s operating system, the security of any background system relying on that card will be compromised.

7.2.2 Composing card commands from security functions

In the previous section, we sketched some functions which are the building blocks for security related commands. The function themselves are not accessible from the outside world: this could be a major lack in the card’s security architecture. They are used internally by the card’s operating system to execute a card command.

We illustrate this with the VerifyCertificate command:

There is some file in the UIM that contains a certificate on some public key. In the Siemens’ protocol, this will be a public key agreement key of the network operator. We denote the certificate with CERT and assume that it was signed with AMV. This additional information can be found in CERT.

For the operating system of the card, the VerifyCertificate command reduces to a simple

TEST := VERIFY(AMV, PARAMETERS, CERT, SIGNATURE_ON_CERT);

It is important to understand the difference between a generic Sign or Verify command and this restricted VerifyCertificate one:

- Certificates have well defined structures
- the card can refuse to verify a certificate if something is wrong with it

7.2.3 Protocol Descriptors and Interpreters in Mobile Terminals

We conclude this introduction into security related card commands with some general considerations on interoperability issues. One of the major reasons for the success of the GSM system is the fact, that every GSM mobile terminal (except for the frequency supported) works with every network and every SIM. The handset acts as a transparent device between network and smart card. This is an easy task in this context because there is only one standardised command which hides any proprietary features in the SIM.

For UMTS, the situation is radically different. Neither the number of card commands or even their functionality stays the same if the authentication protocol changes. This prevents a simple migration from GSM to UMTS. And it introduces a serious danger: if a mobile terminal has some knowledge on particular protocols, it can only support a limited number of such protocols and will not be usable with every network or service provider. The mobile terminal must therefore not have any knowledge on the underlying protocol being executed between UMTS network (service provider) and UIM.

This seems impossible at first sight because the UIM cannot tell the mobile terminal what to do and what not. This is prevented by the master-slave relationship between card and card reader. But the mobile terminal can read some file $EF_{Protocol}$ which stores some sort of description how a protocol (for example the Siemens' one) works in detail. Such a descriptor must describe both the communication between the smart card and the card reader as well as the message flow over the air interface seen from a command triggered point of view.

It is therefore a sort of program written in a high-level description language supported by some interpreter which is implemented in the mobile terminal. It is clear that this high-level description language must be standardised so that every terminal supports it.

With such a solution, we introduce one more trade-off: the description must tell the interpreter exactly which card command to issue at which point in time. This is only possible if there is a fixed list of card commands that can be used in whatever combination. Such a list restricts the protocols that can be supported and is therefore unpleasant from a network operator's point of view.

On the other hand, if the static card commands on the list are so small that nearly every protocol can be supported with them, we have the situation that the security functions described before must be made accessible to the outside world. This is not desirable from a card's point of view.

The solution will probably lie somewhere in the middle and is a good place for further research.