6

Is the Performance of Smart Card Crytographic Functions the Real Bottleneck?

Konstantinos Markantonakis[1]

Visa International EU
Access Channels & Platforms
Virtual Visa, PO Box 253
London W8 5TE
United Kingdom
markantk@visa.com

Key words: Multi-Application, Smart cards, Java Cards, Terminal APIs, Performance Measurements, Cryptographic Algorithms.

Abstract: It is generally believed that among the major delaying factors of smart card performance is the speed of the cryptographic algorithms. This is only partially true, as a number of other factors that add substantial delays to the overall performance of a smart card application should also be taken into account. In this paper we analyse the significance of these delaying factors. Furthermore, we also present some performance measurements of the two most widely used terminal application programming interfaces (APIs) and Java Cards. The aim of this work is to emphasise, both to smart card application developers and smart card technology researchers, the importance of these delaying factors and also to provide a reference point as to the performance of each API.

1. INTRODUCTION

Among the major tasks of a smart card application developer is the identification of any delaying factors that slow down the execution of a smart card application. Delays can be encountered either in the application

[1] The views expressed are personal to the author and do not necessarily represent the views of any other person or organisation for whom the author works or has worked.

running in the smart card or in the application residing in the smart card terminal, i.e. the client or terminal application.

In the past, with multi function smart cards [1,2,3] the situation was simplified. For example, the performance of a smart card application could be measured with relevant precision, since both the smart card and terminal Application Programming Interfaces (APIs) were architecturally simple. Therefore, the only way to achieve better execution times was application code optimisation.

In the recent years with the introduction of multi-application smart cards [4,5,7] the situation changed. Smart cards became capable of securely hosting multiple applications along with dynamically, securely downloading and deleting applications. As a result, the complexity of the smart card operating system (SCOS) increased exponentially [6,7]. Similarly, the complexity of the terminal applications increased significantly as new architectures [8,9,26] emerged. These technologies aim to offer interoperability between smart cards and card acceptance devices. Moreover, they also hide the details of the underlying terminal operating system. Even at this stage, in order to improve smart card application execution, a lot of effort was still placed in code optimisation, improved smart card virtual machines and providing faster smart card microprocessors.

On the other hand, it is generally believed throughout the crypto communities that smart cards are "anaemic" devices that should do as little cryptographic computation as possible. This view resulted in a race to improve the performance of smart card cryptographic algorithms. Obviously, this approach is by no means wrong but if we look at the problem from a different angle there are also other factors, which if improved will significantly reduce the overall execution time of a smart card application.

We believe that smart card application delays mainly come from sending and receiving data packets to/from the smart card. Although this observation is generally recognised as valid it has not yet received the necessary attention. Furthermore, in various smart card related newsgroups, discussion forums and research papers, questions such as how long it takes to communicate with the smart card or which communication API performs better in terms of speed, are always a favoured topic.

In this paper we attempt to provide some meaningful answers to the above questions. In order to achieve our aim we present some performance comparisons between the two most widely used terminal APIs, namely Personal Computer/Smart Card Specification (PC/SC) [8] and OpenCard Framework (OCF) [9]. Therefore, this paper serves two purposes: First it provides some reference points towards which of the two smart card terminal APIs performs better in the available smart card testing platforms. The results of this paper could also be considered as a reference point when

designing smart card terminal applications. Second it highlights the fact that in order to achieve better smart card application execution times it is important to look into other factors apart from cryptography.

The remainder of this paper is organised as follow. First, we outline the characteristics of how communication is achieved with a smart card, both at the physical layer and higher at the application level. Subsequently, we present the implementation environment and a short but detailed analysis of the design characteristics. Moving to the core idea of this paper we then analyse the results from the test implementations. Finally, we discuss several practical issues that imposed certain design decisions and introduce new concepts to act as directions for further research.

2. BACKGROUND

In this section we provide an introduction on how low and high level communication is achieved between the terminal and the smart card application. We also provide some typical smart card cryptographic algorithm performance measurements.

2.1 Physical Data Transmission to the Smart card

Currently there is only a single channel for communication between a smart card and a terminal. This implies that the card and terminal can only transmit in turn and the other party should be in receiving mode. This operation is known as half-duplex operation. Most smart card microprocessors have a single I/O port but since the ISO standards [10,11] reserved two of the eight smart card contacts for future use, full duplex could become technically feasible.

Communication between the smart card and the terminal takes place serially. This implies that each byte to be transmitted in the communication channel should be converted into eight individual bits that are sent one after the other. Since the data transmission proceeds asynchronously, each byte must also be provided with additional synchronisation bits i.e. a start bit, a parity bit and two synchronisation bits.

The data transmission rate is directly proportional to the applied clock of the microprocessor. This implies that the duration of a data bit cannot be given in absolute terms. However the existence of awkward divider values along with the most common clock frequencies aim to provide a transmission speed of exactly 9600bits/s.

Two of the most common data transmission protocols [11,12] are T=0 and T=1. T=0 is asynchronous, half-duplex, byte oriented, was used in

France during the initial phase of the Smart Card development. It is also used in the GSM smart cards and is more commonly used in most current smart cards.

T=1 is asynchronous, half-duplex, block oriented and was introduced in 1992 as an ISO/IEC 7816-3, Amendment 1 standard. The block is the smallest data unit that can be transmitted. This protocol allows chaining of blocks of data i.e. an arbitrary large block of data may be transferred as the result of a single command by the transmission of the appropriate number of frames chained in sequence.

2.2 Communicating Through a Terminal Application

As previously mentioned, smart card application programming interfaces form one part of smart card technology. Another important aspect is the APIs that allow terminal applications to communicate with smart card applications.

Until recently there were no card reader independent application programming interfaces. Two specific reasons for this are: Firstly, the smart cards and the card reader devices were very closely coupled; there was no need for a card to be used with a different card reader and vice-versa. Secondly, card reader programming interfaces were not standardised, whereas smart card interfaces were standardised.

Thus, the most common method employed when smart card programmers wanted to communicate with a smart card application via a smart card reader was the following: obtain the specific drivers for the smart card reader, install them in the system and subsequently integrate them within the terminal application.

PC/SC [8] was developed by Microsoft, Hewlett-Packard, Siemens-Nixdorf, and smart card manufacturers. PC/SC is tied to the Windows platform and terminal applications can be developed in Visual Basic and various C++ compilers. Currently, most smart card manufactures provide PC/SC drivers for their smart card readers.

Another more recent initiative is the OCF [9], which enables Java applications to communicate with the smart card in a transparent and portable fashion. OCF is written in Java and was primarily developed by IBM and other computer technology providers. OCF permits the client applications to access the smart card irrespective of the host operating system and CAD (Card Acceptance Device or Card Terminal).

2.3 Typical Performance Figures of Cryptographic Algorithms in Smart cards

The following table provides some typical figures [23] for the performance of certain cryptographic algorithms in some typical smart card microprocessors.

Table 1. Smart card cryptographic algorithm timings.

Micro-Processors		**ST16-CF54B**	**ST19-KF16**	**P83W8516/8532**	**SLE44C-**R80S
Clock Frequency		5 MHz	10 MHz	5 MHz	5 MHz
Algorithm	***Length***				
DES	64 bits	10ms	N/A	10 ms /	3.7 ms
SHA	512 bits	15.2 ms	8.2 ms	5 ms	5.6 ms
RSA 512	Sign with CRT	142 ms	20 ms	37 ms	60 ms
RSA 512	Sign without CRT	389 ms	55 ms	93 ms	220 ms
RSA 1024	Sign with CRT	800 ms	110 ms	160 ms	450 ms
RSA 1024	Sign without CRT	N/A	380 ms	400 ms	N/A
DSA 1024	Sign	N/A	100 ms	150 ms	N/A
DSA 1024	Verify	N/A	160 ms	225 ms	N/A

Please note that these are indicative figures. The implementation of the cryptographic algorithms is based on a specific Gemplus implementation [23]. These figures will be used later on when comparing the performance of the terminal APIs with the performance of certain smart card cryptographic functions.

3. IMPLEMENTATION ENVIRONMENT AND ANALYSIS

Software solutions that use smart cards are separated into the smart card application and the terminal application. In our case, in order to perform the tests, these two distinct entities had to be developed. In this section, we outline the characteristics of these two entities.

3.1 The Smart card Application Development Tools

For the smart card applications we used two of the most popular Java Card API Ver. 2.0 [13, 14, 15] compliant implementations, the GemXpresso

Java Card [16] from Gemplus and the Sm@rtCafé Professional Java Card [17] from Giesecke & Devrient. Each of the development kits came with its own smart card reader and development tools. The GemXpresso card came with the GCR410-X reader and the Sm@rtCafé with the Towitoko PCT-200 reader.

3.2 The Smart card Application

The smart card application receives certain commands from the terminal application and responds accordingly. Initially it checks whether the Application Protocol Data Unit (APDU) [18] contains any data and whether it requests any data to be sent back by the card. This is actually achieved by checking the "Lc" and "Le" parameters of the APDU respectively. Therefore, there are four basic functions implemented by the smart card application:

- The first receives no data from the terminal and sends an ISO exception (2 bytes) back, i.e. "*Sent_0_Get_0*".
- The second function receives no data from the terminal but at the same time the terminal requests some data, (X) bytes from the card, i.e. "*Sent_0_Gett_X*". In order for the data to be sent back to the terminal a "for loop" statement is implemented within the smart card application. Please note that there is some processing overhead at the card side in order to execute the "for loop" statement
- The third function receives (X) bytes from the terminal and sends an ISO exception (2 bytes) back to the terminal. This function will be referred as "*Sent_X_Get_0*".
- Finally, the card receives (X) bytes from the terminal and also sends (X) bytes back to the terminal, i.e. "*Sent_X_Get_X*". Please note that this function runs through each byte provided by the terminal and adds the value of one. The end result is an array of bytes of the same size as the original one but its values are increased by one. Therefore, this function also contains some smart card application overhead (e.g. for loop, addition).

In general, the smart card application contains an APDU dispatcher that will verify the APDU sent by the terminal. The Java source code for the smart card applications is around 5-6Kbytes. The actual smart card application files downloaded in the cards are 1.2-1.9Kbytes. The above functionality is implemented as smart card applications both in the GemXpresso and Sm@rtCafé smart cards.

3.3 The Testing Environment

For the implementation and the testing of the terminal and smart card application we used the following configuration: an Intel Pentium II 400Mhz PC with 128 Mbytes of RAM under Windows NT. We also used the Microsoft Visual J++ Compiler Version 1.02.7318 and Microsoft Java Virtual Machine Ver 5.00.3182.

In order to obtain a meaningful set of results we performed a number of tests. It is important to note that we are running each test in two different smart cards (Sm@rtCafé, GemXpresso) and each card is tested in two different smart card readers (GCR410, PCT-200). We have also developed two sets of terminal applications one for PC/SC and one for OCF as described in the next section

3.4 PC/SC Application Design

PC/SC is enabled when installing the PC/SC base components from Microsoft. Subsequently, the PC/SC drivers for the corresponding smart card readers have to be installed.

For the GCR-410 reader we used the GrSerial Ver. 1.2.11.0 driver downloaded from the Gemplus web page [19], and for the PCT-200 reader we used the Ver 2.14.11 driver downloaded from the Towitoka Web site [20]. The Towitoko (PCT) PC/SC driver does not occupy a COM port from boot time on, and thus it is possible to use any other device after disconnecting the reader. Strangely, the GrSerial (GCR) constantly occupies the COM and as a result if the port is to be used by another application, e.g. OCF, then the device driver should be stopped from the "Devices" menu under "Control Panel" in Windows. In any case it is suggested that the whole "Smart Card Resource Manager" service should also be stopped under the "Services" menu in the "Control Panel" of Windows.

In order to provide a common testing platform between PC/SC and OCF the PC/SC terminal application had to be developed in Java. Up to recently it was impossible to find any PC/SC Java source code samples, even from the Microsoft MSDN libraries. This was the main reason that forced us to create some Java source code wrappers, by using Microsoft J++ Ver. 6.0, for the PC/SC COM service provider's [21]. Eventually, that enabled us to gain access to the PC/SC COM components through Java code.

An interesting observation is the following: initially the GrSerial driver could not work with the Sm@rtCafé smart card. After reporting our findings to Gemplus we were told that, the Gcr410 reader does not relay the TCK byte of the ATR to the driver if the card supports the T=0 protocol. They also stated that this was due to a change in the standards. This was also the

case for the drivers of GemPC240 (Gcr240), and GemPC400 (Gpr400), except under W2K. The OCF driver for the same reader does not check the TCK byte, which explains why the OCF driver worked even with the Gcr410 v1.00. Finally, Gemplus's response was efficient as we were provided with a more recent version of the GrSerial driver that corrected the problem and enabled us to continue our tests.

3.5 OCF Application Design

For the GCR-410 reader we used the Gemplus Card Terminal Ver. 3.0 downloaded from the Gemplus Web page [19], and for the PCT-200 reader we used the Giesecke and Devrient Card Terminal Ver. 1.1 driver obtained through the mailing list of the OCF newsgroup. In order to maintain compatibility between the two testing platforms we used the generic PassThruCardServiceFactory service [9] of OCF.

One of the great advantages of OCF is that it does not constantly occupy the serial port of the smart card reader. This means that it is easy to monitor, by running a serial port-monitoring tool [22], the communications on the serial port. For PC/SC on the other hand the serial port-monitoring program has to be started before starting the PC/SC Resource Manager and subsequently executing the client application and obtaining any results.

4. PERFORMANCE EVALUATION

In this section we compare the performance of OCF and PC/SC for each smart card reader and smart card.

4.1 PC/SC and OCF Results and Performance Evaluation

Different results, i.e. the time in milliseconds to complete the specified task, were generated depending on the actual functions described in §3.2. In addition to the above functions we also provide the performance measurements for connecting to the smart card reader, selecting the smart card application and disconnecting from the reader. We provide the Standard Deviation and Average figures, for each function, based on a total of ten measurements. Please note that all the results are based on the specific configurations. This implies that when referring to comparisons between cards, readers and architectures any general comments are based on the specific versions of the reader drivers, and the specific design of the smart card applications.

Table 2. The performance of PC/SC and OCF on the GCR410 reader.

API/Reader	PC/SC on GCR410				OCF on GCR410			
Smart Card	Sm@rtCafé		GemXpresso		Sm@rtCafé		GemXpresso	
	ST.DEV	AVER	ST.DEV	AVER	ST.DEV	AVER	ST.DEV	AVER
Connect	0.0	0.0	4.8	3.0	8.6	3618.2	10.5	3623.2
Select	0.0	40.0	4.2	52.0	3.2	71.0	5.4	85.1
Send_0_Get_0	4.6	122.4	4.8	147.0	3.3	139.2	3.4	109.5
Send_10_Get_0	5.7	141.1	5.2	166.3	3.2	149.0	5.0	177.2
Send_20_Get_0	4.8	157.2	5.1	185.4	3.1	151.2	0.5	190.3
Send_30_Get_0	5.2	165.3	4.9	203.3	3.1	171.3	0.0	210.0
Send_40_Get_0	4.6	183.4	4.7	220.2	0.5	190.4	0.5	290.7
Send_10_Get_10	6.2	198.1	5.0	174.2	0.0	210.0	4.1	182.3
Send_20_Get_20	3.0	309.5	3.0	271.6	0.5	320.5	2.9	249.3
Send_30_Get_30	7.3	421.5	5.1	324.3	5.0	444.5	0.5	280.4
Send_40_Get_40	5.7	529.9	5.6	366.7	0.0	561.0	4.1	392.3
Send_0_Get_10	8.2	130.1	3.2	129.0	5.0	133.4	0.5	130.4
Send_0_Get_20	7.0	195.3	3.1	201.3	4.3	208.2	5.8	219.3
Send_0_Get_30	3.4	259.5	3.1	231.4	0.4	270.2	8.1	237.3
Send_0_Get_40	5.3	326.2	4.9	253.3	0.4	330.8	5.2	264.5
Disconnect	0.0	0.0	0.0	0.0	0.0	30.0	0.0	30.0
Overall	70.8	3179.5	66.7	2929.0	40.6	6998.9	56.4	6671.8

When comparing the performance of PC/SC and OCF for the Sm@rtCafé implementation and the GCR reader it appears that overall PC/SC is 18.6% faster than OCF. Please note that this figure takes into account the average timings from all functions. When the dependency of the comparatively slow "Connect" and “Disconnect” figures of OCF are completely eliminated, as a potential improvement, then for the Sm@rtCafé implementation PC/SC will maintain, on average, a 7% lead over OCF.

For the GemXpresso implementation on the GCR reader it appears that on average PC/SC is 15,2% faster than OCF. Similarly, if the dependency of the "Connect" and “Disconnect” figures are not taken into consideration, PC/SC maintains on average a 3.1% lead.

As we observe from table 2 there is an obvious lead in the performance of PC/SC over OCF for each individual function in the Sm@rtCafé implementation. For a few functions in the GemXpresso implementation, OCF performs significantly better than PC/SC.

Table 3. The performance of PC/SC and OCF on the PCT200 reader.

API/Reader	PC/SC on PCT200				OCF on PCT200			
Smart Card	Sm@rtCafé		GemXpresso		Sm@rtCafé		GemXpresso	
	ST.DEV	AVER	ST.DEV	AVER	ST.DEV	AVER	ST.DEV	AVER
Connect	0.0	0.0	0.0	0.0	35.0	5080.3	45.9	4096.0
Select	4.7	70.0	4.9	77.1	5.2	114.0	3.1	121.2
Send_0_Get_0	4.0	128.1	4.9	163.2	3.5	131.2	0.5	100.3
Send_10_Get_0	4.9	153.3	5.3	186.1	3.3	159.4	0.0	190.0
Send_20_Get_0	5.0	167.2	5.1	336.7	3.1	161.2	7.8	262.4
Send_30_Get_0	5.3	174.1	0.5	350.3	4.3	178.2	4.7	273.6
Send_40_Get_0	5.4	194.3	0.5	360.7	4.2	192.1	8.4	287.3
Send_10_Get_10	5.3	214.6	4.1	182.2	3.1	221.4	4.7	197.5
Send_20_Get_20	4.6	433.3	3.4	291.4	5.1	326.4	5.5	295.2
Send_30_Get_30	6.5	460.7	4.2	338.5	4.1	438.8	5.8	341.5
Send_40_Get_40	0.3	580.9	5.2	375.5	4.3	552.8	0.5	380.5
Send_0_Get_10	4.9	137.1	4.3	132.4	5.2	1728.3	0.5	10054.3
Send_0_Get_20	0.5	190.3	0.4	220.2	0.5	1772.7	18.3	10063.7
Send_0_Get_30	4.5	300.6	0.5	240.4	0.5	1832.6	4.6	10066.3
Send_0_Get_40	4.1	322.3	4.5	268.4	0.5	1872.5	3.0	10075.6
Disconnect	0.0	0.0	0.0	0.0	8.5	65.0	8.2	63.0
Overall	60.0	3526.8	47.9	3523.1	90.5	14826.9	121.5	46868.4

When comparing the performance of PC/SC and OCF on the PCT reader, i.e. Table 3, the situation becomes more complicated. A closer observation will reveal that the performance of both implementations (Sm@rtCafé and GemXpresso) under OCF is influenced by the extremely slow performance of the "Connect" and "Send_0_Get_X" functions. Specifically for the "Send_0_Get_X" functions, the results are unreasonable and indicate that probably these operations are not handled properly from within the OCF driver of the PCT reader. Therefore, for the sake of completeness and clarity we decided to include the performance of the "Send_0_Get_X" functions in Table 3, but do not take them into account when reaching into certain conclusions.

The performance of PC/SC on the PCT reader for the Sm@rtCafé implementation is on average 16.9% faster compared with the one in OCF. But, this is heavily influenced by the large “Connect” and “Disconnect” figures of the OCF implementation. If the influence of these two functions is removed then PC/SC maintains a marginal lead of 0.2%.

Similarly, the performance of PC/SC for the GemXpresso implementation on the PCT reader is on average 8.8% faster than the

corresponding of OCF. An interesting observation is that when the influence of the "Connect" and "Disconnect" figures of the OCF implementation are eliminated then OCF gains a 9.5% lead over PC/SC.

Another interesting observation, by looking in table 2, is that OCF on the GCR reader demonstrates an overall lower standard deviation, for both smart card implementations, when compared with the corresponding one of PC/SC. This implies that OCF appears to be more stable and produces fewer variations in the measurements. For OCF on the PCT reader, i.e. table 3, the situation is exactly the opposite as the standard deviations are significantly larger when compared with the corresponding ones from PC/SC. The latter observation should be considered of minor importance when taking into account the unreasonable performance of OCF on the PCT reader.

From the figures in both tables we can see that the "Connect" and "Disconnect" figures for OCF are relatively large compared with the corresponding of PC/SC. These delays can be possibly explained on the design of OCF. An interesting observation is that when OCF attempts to establish connection with the reader there is increased hard disk activity as OCF searches for certain Java classes. Therefore, carefully setting the classpath of the testing platform will potentially result in small performance improvements. The slow connect and disconnect figures can be possibly explained by the fact that OCF does not constantly occupy the serial port as is the case with PC/SC.

At this stage we have to be very careful with the above observations as they are really based on the aforementioned specific implementations. In order to obtain a clearer picture on what are the actual issues involved around the performance of each technology, it is recommended that the reader carefully examines the timings in each table and for each individual function. In that way any potential influence to the overall result by each individual function is removed.

4.2 Further Discussion of the Results

When comparing the performance of the Sm@rtCafé implementation under PC/SC in the two different readers we realise that the GCR implementation appears to perform on average better.

When comparing the GemXpresso implementations in the two available readers and under PC/SC we observe that on average the GCR implementation is faster than the PCT. On the other hand the corresponding standard deviation of the GemXpresso and Sm@rtCafé implementation on the PCT reader is lower. For both smart card implementations under PC/SC it appears that application selection takes place faster in the GCR reader.

It is worth mentioning that both the "Connect" and "Disconnect" figures are extremely small, this can be possible explained by the fact that in PC/SC there is constant traffic in the serial port. Therefore, connecting and disconnecting to/from that card happens almost immediately. Overall, the GemXpresso implementation under PC/SC on the GCR reader maintains a marginal lead. The fact that both smart card implementations demonstrate slower measurements in the PCT reader can be explained, at this stage, due to inefficient APDU handling either at the corresponding card reader or at the PC/SC driver level or even due to differences in the actual smart card microprocessors.

When comparing the performance of the Sm@rtCafé implementation under OCF we realise that the GCR implementation is significantly faster. When comparing the performance of GemXpresso implementation under OCF we observe that the GCR implementation is once more relatively faster compared with the PCT implementation

Both smart card implementations under OCF on the PCT reader demonstrate notably large figures for the "Connect and" "Select" functions along with unreasonably large standard deviations when compared with the corresponding ones on the GCR reader. This indicates that probably the OCF driver for the PCT reader is not properly implemented.

5. CONCLUSIONS AND DIRECTIONS FOR FURTHER RESEARCH

When checking the typical smart card cryptographic algorithm figures from Table 1 we can see that, a typical cryptographic operation ranges from 20-450ms and on average it takes around 160ms. This figure is equivalent with sending 10 bytes to the card or sending 10 bytes and also getting 10 bytes as a response. Analogous conclusions can be drawn when taking into account the fact that the performance of different smart card cryptographic algorithms is comparable with sending or receiving a number of bytes to/from the smart card.

Up to recently, a lot of the discussion about smart cards concentrated on improving the performance of the smart card cryptographic functions. The end-result was tiny improvements in order of a couple of tens of milliseconds for a cryptographic function that could be used once or twice within a smart card application. It is clear that more effort should be placed on improving the smart card communication API, as it appears to be more extensively used during the execution of a smart card application, than just concentrating on improving the performance of the smart card cryptographic algorithms.

We have to bear in mind that the PC/SC terminal application was developed in Java. If it was developed in C++ or Visual Basic, non-interpreted languages, then it could be the case that the overall execution time is improved. On the other hand the portability issue will be eliminated, as the terminal application will be closely tied in with the underling development platform.

Further work is actually required in order to obtain more results with the latest versions of the smart card reader drivers and APIs (particularly the new version of OCF 1.2). It would also be helpful to obtain more results when testing the proposed functionality with the native smart card readers in order to explore the actual benefits from sacrificing speed against interoperability.

Another important factor which significantly reduces the overall smart card application performance, and has not yet received the necessary attention, is the size of the communication buffer, i.e. the APDU buffer. More effort should be placed in order to increase the size of the buffer especially in the light of the multi applications smart cards and the high probability of large packets of information travelling towards the card, e.g. applications to be downloaded. For example with an APDU buffer of 512 bytes an application will be downloaded in significantly less time and with less APDU exchanges compared to a 256 byte buffer.

A final remark is that it is not easy to talk about absolute timings and performance measurements when smart card communication is involved. Improving the performance of smart card cryptographic functions [24,25] used to be the area that received the most attention. As demonstrated by this paper increasing emphasis should also be placed in additional areas.

ACKNOWLEDGEMENTS

The author would like to thank Dieter Gollmann for his helpful comments. Also, Konstantinos Skourtis for his help regarding the development of the PC/SC terminal applications. Finally, his current employer, Visa International EU, for facilitating the authors' participation in the conference although the views of the paper do not necessarily represent the company's view.

REFERENCES

[1] Gemplus, "Multi_Application Chip Operating System (MCOS) Reference Manual Ver 2.2", 1990.

[2] Gemplus, "Multi_Application Payment Chip Operating System (MPCOS) Reference Manual Ver 2.2",1994.

[3] General Information Systems Ltd., "OSCAR, Specification of a smart card filling system incorporating data security and message authentication", Available from http://www.gis.co.uk/oscman1.htm

[4] Schlumberger, "Cyberflex Smart card Series Developers manual", Available from http://www.cyberflex.austin.et.slb.com/cyberflex/cyberhome

[5] Gemplus, "GemXpresso Reference Manual", Available from http://www.gemplus.fr/gemxpresso/index.htm

[6] Constantinos Markantonakis, "The Case for a Secure Multi-Application Smart Card Operating System", Information Security Workshop 97 (ISW'97), September 17-19, 1997, Ishikawa, Japan, In Lecture Notes in Computer Science (LNCS), volume 1396, pp.188-197

[7] MAOSCO, "MULTOS Reference Manual Ver 1.2", Available from http://www.multos.com/

[8] PC/SC Workgroup, "Specifications for PC-ICC Interoperability", Available from www.smartcardsys.com

[9] Open Card Consortium, "OpenCard Framework Specification OCF", Available from www.opencard.org

[10] International Standard Organisation. ISO/IEC 7816-2, "Identification cards --- Integrated circuit(s) cards with contacts, Part 2, Dimensions and location of the contacts", International Organization for Standardisation, 1996.

[11] International Standard Organisation. ISO/IEC 7816-3, "Identification cards --- Integrated circuit(s) cards with contacts, Part 3, Electronic signals and transmission protocols", International Organization for Standardisation, 1997.

[12] W. Rankl, W. Effing, "Smart Card Handbook", John Willey and Sons, 1997.

[13] Sun Microsystems, Java Card 2.0 Language Subset and Virtual Machine Specification, http://www.javasoft.com/products/javacard, 1998.

[14] Sun Microsystems, Java Card 2.0 Programming Concepts, http://www.javasoft.com/products/javacard, 1998.

[15] Sun Microsystems, Java Card 2.0 The Java Card API Ver 2.1 Specification, http://www.javasoft.com/products/javacard, 1998.

[16] Gemplus, GemXpresso Reference Manual, Gemplus, 1998.

[17] Giesecke & Devrient, Sm@rtCafe Reference Manual, Sm@rtCafé Professional Toolkit, 1999.

[18] International Standard Organisatian. ISO/IEC 7816-4, "Information technology --- Identification cards --- Integrated circuit(s) cards with contacts --- Interindustry Commands for Interchange", International Organization for Standardisation, 1995.

[19] Gemplus Web Site, www.gemplus.com/developers, 1999.

[20] Towitoko Web Site, http://www.towitoko.de/deutsch/eng/WD1034 s.htm, 1999.

[21] Mary Kirtland, "The COM Programming Model Makes it Easy to Write Components in Any Language", Technical Report, Microsoft Systems Journal, December 1997, http://www.microsoft.com/msj/1297/complus2/complus2.htm.

[22] Mark Russinovich, "Portmon", http://www.sysinternals.com, 1999.

[23] Helena Handschuch, Pascal Paillier, "Smart Card Cryptoprocessors for Public Key Cryptography", In Springer Verlag. Third Smart Card Research and Advanced Application Conference - CARDIS'98, September 1998 to be published.

[24] R. Ferreira, R. Malzahn, P. Marissen, J.-J. Quisquater and T. Wille: "*FAME: A 3rd generation coprocessor for optimising public key cryptosystems in smart card applications*" In P. H. Hartel et al., eds, *Smart card Research and Advanced Applications* -- *Cardis '96*, Amsterdam (The Nederlands),18-20th September 1996, Publ. Stichting Mathematisch Centrum, pp. 59-72, 1996.

[25] T. Boogaerts, "*Implementation of Elliptic curves cryptosystems for smart cards*", In proc. of *CARDIS 1998*, 14-16th September 1998.

[26] Constantinos Markantonakis, "Interfacing with Smart card Applications (The PC/SC and Open Card Framework)", Elsevier Information Security Technical Report, 3(2):82-89, 1999.