

Present and Future Smart Cards

Jean-François Dhem and Nathalie Feyt

Gemplus - Card Security Group
Parc d'activités de Gémenos - B.P.100
F-13881 Gémenos
France

{jean-francois.dhem,nathalie.feyt}@gemplus.com

1 Introduction

In an attempt to secure a rapidly growing information network, industries and researchers have been giving an increasing attention to smart card technology.

During the past decade, smart cards have evolved from the basic memory cards to today's complex cards whose chips incorporate powerful processing units with dedicated peripherals which offer an incredibly wide range of applications. Smart cards are used to secure applications ranging from financial transactions and e-commerce to physical access control, through access to mobile communication networks like the GSM or upcoming UMTS.

To carry out such heavy tasks, smart cards are equipped with the ability to perform on-board cryptographic digital signature, encryption and authentication. For all this, smart cards can be seen as security tokens.

However, all this wouldn't have been possible without an implicit cooperation between both chip manufacturers and smart cards providers to propose speed-optimized products, featuring secure and fast hardware features with interoperable applications like open smart cards.

In this paper, we propose a general overview of the smart card technology with an emphasis on its evolution during the past years. Both the hardware and software aspects are covered. We show that an optimum performance (in terms of security and speed) can be reached by considering both features simultaneously. This is illustrated with detailed examples on how a compromise between hardware and software evolution can help to increase the performance of on-board public and secret key cryptography. Finally, we give a general overview of the attacks against smart cards and of the counter-measures that are implemented to guarantee the tamper-resistance of those, less than 25 mm² "cryptographic processing units".

2 Smart Cards as security tokens

Smart cards offer the ability to store secrets and personal data in a secure way, and to interact within a system us-

ing special communication interfaces and applications dedicated to specific protocols. This leads to a variety of services proposed by the card, services ranging from data management to hardware drivers, passing through cryptographic services. Applications for smart cards are numerous: financial services, mobile communications, health and transport services, e-business. Each day, millions of cards are produced worldwide: most come from Europe. The United States is getting more and more interested in this technology, while Asia has already adopted it.

An explanation for this success story lies in the symbiosis between hardware and software that is inherent to smart cards.

We take three examples to show how hardware and software matches together: the first one is about how securely data is kept secret in a smart card device. The second is about the cryptography services' implementation, and the third is about flexibility of smart cards.

Sensitive data, such as personal data, secrets keys, and some application information have to be stored in memories. This write/store operation is far more protected than in any other devices. First the hardware will have special on-board security sensors, known Vcc/Clock glitches detectors, to prevent the memory to be altered during data storage or reading. Then the software demonstrates backup mechanism in case of card power-down during storage. Above all, access mechanisms can be a combination of usual Operating Systems (OS) access management added to systematic cryptographic verification (authentication, confidentiality), enhanced with hardware features that ensured hardwired "firewall" between memory areas. Eventually hardware/software protections exist against illegal tape-out reading!

Enabling authentication and ensuring confidentiality and integrity imply that smart cards have the additional required capacity for arithmetic computations.

Smart cards are at least proposing secret keys algorithms such as the well-known DES, FEAL, AES [MvOV97], or proprietary ones namely for telecom operators. These algorithms only lead to data substitutions, permutations, compressions and table lookups, boolean to arithmetic conver-

sions, which are simple, and which lead to fast implementations even when performed in high level languages. The associated key lengths are short (from 56 to 128 bit length) and quite easy to manage.

For high-end products, smart cards offer much more powerful cryptographic algorithms, known as public key algorithms (RSA, DH). But in this case, you need to have an arithmetic unit to compute modular multiplication and reduction on large numbers, since the keys are at least 512-bit long, at most 2048-bit long! For both type of algorithms, for efficiency issues, chips may have dedicated peripherals such as DES or RSA crypto-coprocessors. Such peripherals are used by the OS during an authentication scheme for example, either directly, either by adding software to enhance hardware security.

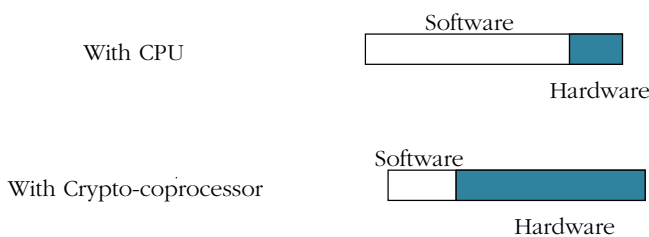


Figure 1: Comparison of the software and hardware parts for an RSA implementation.

The chip and the OS were proprietary since there was no standard on smart card OS. Furthermore services are always driven by constraining standards like ISO 7816-4 or GSM 11.11 or some dedicated customers applications.

There was little room to easily add features, or modify existing ones. The revolution came with open cards, namely Java Cards (see at <http://www.javacardforum.org>, <http://www.opencard.org> and <http://java.sun.com/products/javacard/>). Those cards allow a Virtual Machine (VM) to go beyond hardware and software limitations. This VM interprets a standard language and translates it into machine language. Then the same software can be run no matter what the chip and who the card issuer is. This nevertheless requires a big amount of processing time. More powerful hardware and fast internal clock make it possible.

These three examples have shown how complementary hardware and software can contribute to this technical success story. There is still room for progress: some chips are now even proposing services that were previously done by software, and some OS will perform services that were usually done by hardware.

3 Smart card evolution within the past five years

It is interesting to see how smart cards have been following, with some delay, the same evolution as computers, during the past 10 years. Knowing the technology constraints of

size (maximum silicon chip size is 25mm^2), of power supply (3V to 5V given by dedicated card readers), or terminals (external clock limited to 5 Mhz) and of tamper-resistance, we would even say that smart cards has introduced something new in the information technology: keep doing the maximum with the minimum resources. Even smart card limitation has been pushed further, always enhanced by security features, to offer the maximum, going from new hardware characteristics to new OS architectures.

Compared to 5 years ago, a wide range of smart card chips are now on the market. New chip manufacturers, such as Samsung, Toppan, Toshiba, Emosyn, Datang and Goldkey, propose low-end products which are cheaper, but less secure.

The main chip manufacturers, such as STMicroelectronics, Infineon, Philips, Hitachi, NEC, and even some newcomers like ATMEL, propose high-end chips with different combinations for memory sizes & technology and cryptographic & security features.

Today's cards contain at least 128k bytes of ROM, associated with 64 to 128k bytes of EEPROM or FLASH memory, and between 4k bytes to 8k bytes of RAM, as compared to the 16k bytes of ROM, 4k bytes of EEPROM, and 256 bytes of RAM proposed 5 years ago. The explanation of this evolution comes mainly from silicon technology, which changed its transistors' scale from about $1\mu\text{m}$ in 1995, to at minimum $0,18\mu\text{m}$ nowadays.

Alternatives to the traditional contact cards are useful to bypass all the power supply/communication limitations. First to appear were the contactless chips that are driven by induction circuitry inside the card's body. Those chips lead to several applications, namely in transport or e-ticketing.

Last year, new contact cards having two special contacts for USB port were demonstrated at CARTES2000, opening high speed communication rates with computers (See on <http://www.gemplus.com/usb>), even allowing smart cards (which have added tamper resistance) to replace, some actual USB tokens.

Moreover, to overcome slow external clock limitations, chips are running with their own internal clock, independent or not of the external clock. Five years ago, chips were using this slow (at maximum 5 Mhz) external clock, even for the heavy internal computations such as public-key cryptography. The existing smart card terminals were not able to provide higher frequencies and it was inconceivable to modify all of them. This led to a bottle neck. Chips with asynchronous communications brought the solution: CPU peripheral running their own clock. The CPU's with their peripherals are now able to run at about 30 MHz internal clock. Chips gained a speed factor of 6-10, like computers went from 100 MHz to 1 GHz! Unfortunately, as for computers, applications are not running 10 times faster due to software evolutions and new services.

The latest chips may be very different from one chip manufacturer to another. Furthermore, due to specific hardware mechanisms, namely with dedicated software drivers, the OS architecture may become more and more complex if

we want to take advantage of all the new software features described before. On top of that, time-to-market pressures software to be portable and more flexible. Here is a non-exhaustive list of new hardware features: some are quite simple (for usual microprocessors) such as UART (communication driver), memory banking which enhances addressing capacity, timers, interruption handler, error correction and scrambling for high reliable memories and new security sensors. Others are far more complex because they replace old software services. For example, Memory Management Unit (MMU), or hardwired memory access manager, replaces part of software memory allocation with an added value regarding data access security. Some chips even bear hardwired Java™ VM!

OS evolved from a complete assembly to a higher level language: this is for both flexibility and portability reasons. Ten years were enough to reach the first open cards, changing smart cards from an esoteric programmers' community to a hopefully wider one. We can even imagine smart cards furnished directly with computers, with the opportunities for end-user to easily program or configure it as a real secure personal portable object.

High level languages such as Java™, sustained by Sun, even have a dedicated language instruction subset for smart cards called Java Card™ (by extension, any smart card supporting Sun's Java Card™ is called Java card). Java cards allow any Java™ application (applet) to run on any chip or OS. Interoperability is made possible by means of a standardized set of APIs and bytecode (executable code on a Java VM) subsets. These constitute VM specifications.

Today, smart card OS's infrastructure has completely evolved from a chip dedicated one to a chip independent one, by means of low level drivers layered by system then application services, organized like the OSI layers (see figure 2). The chip drivers are in charge of low level or hardware mechanisms: cryptographic peripherals (e.g. RSA, DES, random generator), system's peripherals for memory writes/reads/accesses and communication, and the security peripheral (normal working conditions sensors, data security sensors. . .). This part is usually chip specific, but can be used either with open cards or with native cards.

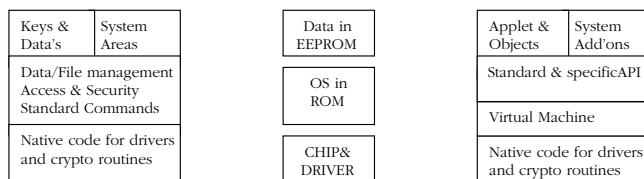


Figure 2: Native card compared to Java Card™ architecture.

Other mechanisms like memory allocation/de-allocation including garbage collector, memory fragmentation or ciphering, backup mechanisms for power loss, cache memory to speed up either code execution or data storage, have been set up to enhance data storage security and execution efficiency. Those mechanisms might depend on the OS, as they manipulate data. However they are closely linked to

the chip, especially if the chip proposes hardwired MMU or memory banking.

The data management remains OS specific (regarding system structure for data containers) whatever the data are objects or files. Access management is far more secure than in any other computer OS: objects/files/keys can be protected in read/write/execution by secrets codes, or authentication-granted-access-rights set up by keys involved in symmetric cryptography such as DES or asymmetric cryptography such as RSA algorithms. Similar mechanisms exist for added value services, like application one's.

The gap between native cards and open ones comes from the ability, with open cards, to download new services and commands that are associated to a dedicated application. An illustration is the so-called applets. Applets give the opportunity to download data and services, dedicated to Java Cards, only one version being needed for all cards or chips. This changes mainly the high level layers of the OS, and the way services are managed during the card life.

Open cards propose services like any OS but have very few commands, that is to say ways to command the card. Eventually, OS services are reachable by means of API's, standardized or not, to satisfy interoperability issues. On the contrary, native cards only propose a fixed set of commands, without any possibility to access a service in another way.

Smart cards are thus the smallest but one of the finest technology jewels. Today's higher demand for and greater awareness of security problems lead to study more secure systems. We are now going, to show alternatives for the next generation of smart cards. We will demonstrate that technical evolution are still possible and required, smoothing the boarder between software and hardware services.

4 Future hardware architectures

Several reasons exist to improve fast software implementations:

- improve smart card flexibility (multi-applicative cards are more and more asked because their implementation often requires secure applet downloads with VM able to secure monitor running applets).
- The need for cryptographic algorithms often require a very high computing power. Furthermore, it is not always possible nor recommended to implement cryptographic algorithms in hardware. Indeed, applications like for the GSM require on-board implementation of proprietary (and not public) cryptographic algorithms which vary from one Telecom Operator to another. It is not possible to provide hardware implementation for each algorithm. Furthermore, it is more flexible to adapt software countermeasures for some potential attacks than redesigning a new hardware block.

The above reasons will certainly lead to improved embedded CPU following with more than a 10 year delay

what, happened on the PC computer market. Nevertheless, as we have already seen, the constraints are far more important since we are in an embedded world, where all the memories and peripherals have to be included in a very small die that should also be well protected against active and passive attacks.

To obtain, in software, speed comparable to the best existing DES (Data Encryption Standard, secret key algorithm [MvOV97]) or modular multiplication (for public key algorithms like RSA [MvOV97]) co-processors, there is still some work to be done when looking to implementations on the state-of-the-art embedded processors like ARM7M, MIPS32 or V850. Some work is actually undertaken (like on STMicroelectronics SmartJ™) to modify such processors by introducing new specialised instructions that will speedup the software implementations in the framework of smart cards (as for cryptographic algorithms as well as for new multi-applicative OS or for communications drivers).

4.1 An historical approach of 32-bit RISC in smart cards

The first project to introduce a 32-bit (RISC) in a smart card was the European Project EP8670 funded by the European Commission through the Esprit Program. The project was launched in December 1993 with several Academic and Industrial european partners (see <http://www.dice.ucl.ac.be/crypto/cascade/>). The designed smart card was based on the ARM7M RISC chip from ARM Ltd (which was a partner of the project). The critical point at that time (1997) was the global die size of the chip, the CPU was quite large for the 25mm² maximum die for smart card (defined in ISO standards 7816) such that the RAM (512 bytes of data) an FLASH (32K bytes for code and data) was quite limited. These constraints are far less important in actual 0.18µm technologies. Today's main constraints are the overall power consumption for markets like the GSM industry (SIM cards must not consume more than 6 mA @ 3V and 4 mA @ 1.8 V for GSM phase 2+ specifications)

4.2 Present Smart Cards using 32-bit CPU

Most of the main manufacturers now have 32-bit RISC CPU for the state-of-the-art smart-cards. We have namely:

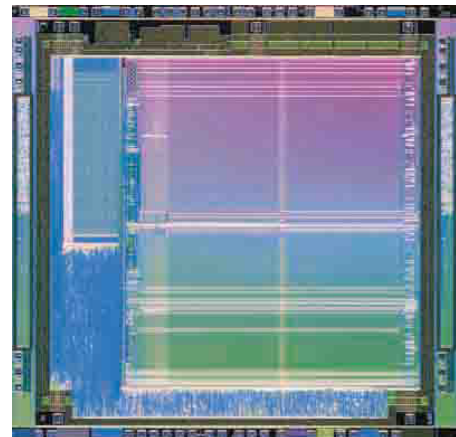
- STMicroelectronics SmartJ™ ST22 using their own 32-bit RISC, see <http://eu.st.com/stonline/prodpres/smarcard/insc9901.htm>
- NEC V-WAY family using a V850 32-bit RISC and SuperMAP crypto-processor <http://www.ee.nec.de/Centers/SmartCard/MCUS/Index.html>
- Infineon SLE88 which will be launched next year. See http://www.infineon.com/cgi/ecrm.dll/ecrm/scripts/prod_cat.jsp?oid=-8235

Other manufacturers like Philips do not migrate directly from 8-bit based smart cards to 32-bit

ones. They prefer using their knowledge on 8-bit CISC to create improved 16-bit CPU like the SmartXA: <http://www.semiconductors.philips.com/identification/products/smartxa/>

Samsung, one of the youngest on the market, is also introducing RISC architecture based on the CalmRISC core which exists in 16 and 32-bit versions: http://www.samsungelectronics.com/semiconductors/System_LSI/System_LSI.htm

Atmel uses its own 8-bit RISC CPU and also provides some components with the ARM 32-bit Thumb core inside.



Photograph: Cadrage
By courtesy of Atmel Rousset, France

Figure 3: Atmel smart card chip 3232 in 0.5µm technology

Compared to 5 years ago, the smart card technology is now very close to the one used for standard CPU's (today 0.18µm) The big challenge for smart card silicon manufacturers is to quickly adapt the analog (charge pumps, voltage and current regulators, embedded clock generators, security sensors, . . .) and EEPROM/flash technologies to smaller and smaller digital technologies.

We are going to go into more details about some possible improvements for cryptography in the next generation smart cards.

The main cryptographic algorithms already implemented on smart cards or that are on the way to be implemented include:

- secret key encryption algorithms like the famous DES [MvOV97], AES (Advanced Encryption Standard, see <http://www.nist.gov/aes/>), which is the new US encryption standard that should progressively replace the DES;
- Standard public key cryptography based on modular multiplication like RSA, DSS (Digital Signature Standard, see <http://www.nist.gov/dss/>) or DH (Diffie-Hellman) [MvOV97];
- public-key algorithms based on elliptic curves[BSS99]. This is quite novel and not yet extensively used. There

exist two main types of commonly used curves which will determine the needs in terms of computing power.

- Curves over $\text{GF}(p)$ which require resources similar to the standard public key cryptography.
- Curves over $\text{GF}(2^n)$ for which implementation is faster than the previous one on standard, CPU architectures: the computations are done without carries (addition/subtraction is a XOR, ...);
- Hash functions [MvOV97]: mainly SHA-1/-2 (Secure Hash Standard, see <http://www.nist.gov/sha>) and RIPEMD [MvOV97].
- There are also several proprietary (for which specifications are not public) secret key algorithms required by many Telecom Operators for their GSM (and UMTS) digital cellular phones.

It is also important to recall that the found improvements for cryptography must be taken into account (must be compatible) with the fast software counter-measures described in section 6.

The following two sections show examples of what can be done to improve the speed of secret and public key cryptographic algorithms on 32-bit CPU's for smart cards.

4.3 Secret-key algorithms

In general, secret-key algorithms like DES, AES and the hash functions have very good performances on 32-bit microprocessors (100 μ s...2ms depending on the algorithm, the CPU and this whether counter-measure are implemented or not). Nevertheless, these implementations can be 10 times slower than hardware co-processors. Furthermore, when countermeasures are implemented, the speed is very close to the millisecond. This is why research is done to improve software implementation.

Example of interesting instructions for secret-key cryptography are:

- rotation instructions on the content of a register;
- bits permutations, expansions and substitutions instructions. This will be done by the means of a configuration register determining the ways in which the bits of one initial register will be placed in another register.
- very fast memory access (e.g. for S-Boxes randomization).

4.4 Public-key algorithms

The public-key battlefield is to keep high security level by gaining performances in execution time and to reduce the product costs by suppressing the dedicated crypto-coprocessor and using only CPU instructions. Another issue is the growing RSA key lengths which was 512-bit in 1995 to 1024-bit and even, today, 2048-bit key lengths.

Big improvements in execution time for usual key lengths were done in recent years. Today, a 1024-bit RSA signature (with CRT) takes at maximum 200 ms for high secure implementations. On-board key generation for 1024-bit RSA keys is less than 10 s.

To gain in software flexibility and in global costs we can get rid of crypto-coprocessors by adapting, efficient CPU core for smart cards, with dedicated instructions for cryptography.

For standard public-key cryptography (e.g RSA), fast software implementation on 32-bit embedded RISC CPU is more problematic than for secret key algorithms.

The basic primitive of cryptography algorithms like RSA, DSS and DH is the modular multiplication of large integers (typically 512...2048 bits). Mathematically the equation is (this consists of the computation of the remainder of an integer division):

$$(AB) \bmod N = (AB) - \left\lfloor \frac{AB}{N} \right\rfloor N \quad (1)$$

$$(2)$$

Where A , B and N are large integers. To work on 32-bit architectures these integers have to be decomposed in 32-bit blocks (see figure 4) and the full size computations should be done by emulation using 32-bit numbers. Decomposing the modular multiplication 1 by block we

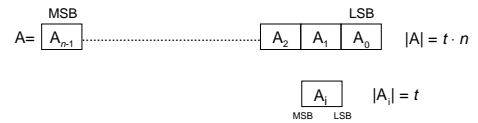


Figure 4: Representation of numbers in multi-precision.

have the following equation:

$$\begin{aligned} AB \bmod N &= \left(\sum_{i=0}^{n-1} 2^{it} A_i B \right) \bmod N \\ &= \left(\dots (A_{n-1} B 2^t + A_{n-2} B) 2^t + \dots \right. \\ &\quad \left. + A_1 B) 2^t + A_0 B \right) \bmod N \quad (3) \\ &= \boxed{\left(\dots \left((A_{n-1} B \bmod N) 2^t + \dots \right. \right. \\ &\quad \left. \left. A_{n-2} B \bmod N) 2^t + \dots + \right. \right. \\ &\quad \left. \left. A_1 B \bmod N) 2^t + A_0 B \right) \bmod N \right) \quad (4) \end{aligned}$$

We could argue that the decomposition of equation (4) is the only one possible to implement the computation. We could, for example, first multiply A by B and then reduce by N . In fact, the RAM memory on smart cards is often limited in term of size and access time, we look therefore for minimizing the usage of the memory (without slowing down the computations). In that sense equation 4 minimize the RAM memory needed for intermediary computations. We can further see that memory access can be minimized

by interleaving (at each step) the multiplication and the reduction phase.

The evaluation of the intermediary quotients $q = \lfloor A_i B / N \rfloor$ can be done very efficiently using various dedicated methods improving the classical quotient evaluation by division (Montgomery [Mon85], Barrett [DQ00], Quisquater [DQ00], Sedlak [Sed88],...). We will not describe these methods. But the method which is the best suited for fast software implementations on standard CPU architecture is the Montgomery one (it has nevertheless some other disadvantage). In this algorithm, the evaluation of a quotient corresponding to q of each intermediate computation in equation (4) is on n bits for n -bit architectures. With other methods, this quotient would be larger than the size of the architecture.

From what we just saw, we may immediately conclude that, to be efficient, we require at least a hardware multiplier. Normal hardware multipliers on embedded (32-bit RISC) processors were initially designed for digital signal processing applications. This means, the existence of a multiplier with a 64-bit accumulator. In terms of input bits we can express this as a 32-bit \times 32-bit+64-bit multiplier with a result on 64 bits. One of the main disadvantages of that adder, when emulating computations on larger integers, is that the carry bit of the accumulator is not accessible.

One step of the computation shown in equation (4) can be represented graphically as in figure 5 where **Temp** represents the result of the previous step to which the next $A_i B$ is added, the result being reduced by N (as shown in equation(4)). There is no subtraction, the computations being done using 2's complement of de product of N .

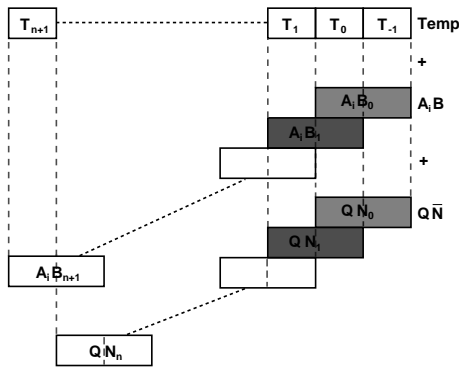


Figure 5: One step of a modular multiplication.

Looking at figure 5, there exists two different ways to implement the computations. To reduce memory access, the multiplication $A_i B$ will be interleaved with the computation of $Q \bar{N}$ as shown in the figure 6 where an elementary step of the computation is represented. The first representation (left part of the figure 6) uses a “standard” DSP-like multiplier 32-bit \times 32-bit+64-bit and the second one uses a better suited multiplier using a 32-bit \times 32-bit+32-bit+32-bit one. The method at the right of the figure seems to be the best one since it does not need any carry bit for the

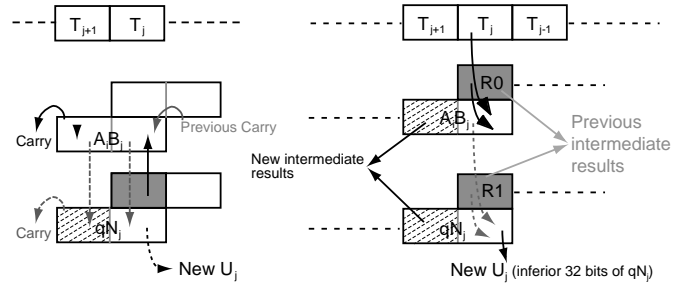


Figure 6: One elementary step.

multiplier and consequently increase the “fluidity” of the algorithm by avoiding the necessity to handle these carries. Indeed, if we look at the way the recombination is done at the right side of figure 6, we realise that the carry of the previous computation has to be added to the next one. The reason why there is no carry on the right method is due to the theorem 1 (easy to demonstrate).

Theorem 1 *If $A, C < 2^\alpha$ and $B, D < 2^\beta$ then*

$$A \times B + C + D < 2^{\alpha+\beta}$$

It is not difficult to modify a standard DSP multiplier to match such modifications. Depending on the implementation, it does not even require an additional large accumulator with larger delays as in a DSP version. The cascade of the two additions is even free in the case of an array multiplier (C and D can be directly injected into the multiplier’s first row of full adders).

In addition to the computations (multiplication) themselves, memory accesses take also place to read and store back the intermediary results in **Temp** and to read A, B and N . A typical implementation, in pseudo-code, of the resulting inner loop of a modular multiplication is as in figure 7.

```

for j=1 to n-1
  (R0,R1)=Q.Rn + Rt + R0
  T[j]=Rk
  Rb=B[j]
  (R1,Rk)=Ra.Rb + R1 + R1
  Rt=T[j]
  Rn=N[j-1]
endfor

```

Figure 7: Main loop optimized.

We can approach dedicated hardware performances by improving the handling of the loop (see figure 7) having specialised instruction handling simultaneously a branch, a counter and a test on its value.

Mixing the multiplication while memory accesses are carried-out makes it also possible to optimise the use of

the multiplier which can then be implemented in a multi-cycle version in order to reduce its die size.

Other improvements include unrolling the loop to gain on loop branches and on some memory accesses.

The implementation described in this section was already successfully implemented in dedicated hardwired coprocessors, it is nevertheless far more interesting to be able to do it by software for the various already discussed questions.

5 Attacks on smart cards

As well as other security devices, smart cards are subject to various kind of attacks by malicious people. We will sketch here the most recent and most used attacks on such embedded device.

5.1 Side Channel Attacks

Side channel attacks consist in retrieving any secret information from the smart card using any leakage of it. A now well known class of attack in this group is based on smart card power consumption analysis (Differential Power Analysis and Simple Power Analysis [KJJ99]) and timing attacks [DKL⁺00]. The first publication in 1996 by Kocher created a shock in the smart card community, even if some doubt was present in the past. After a short period of turmoil, the problem has now been brought under control.

The concept of **Simple Power Analysis** (SPA) is very simple. It consists of observing the variations in the global power consumption of the chip and retrieving from it some information which can help to identify any secret. A very easy SPA on straightforward implementations of RSA cryptographic algorithm can be done by looking to the increase in power consumption each time a modular exponentiation is done (this is even evident when using a hardware coprocessor) which allows to deduce, bit by bit, the secret exponent. Indeed, the RSA signature, in fact consists in a modular exponentiation (the exponent being the secret key). In a straightforward square and multiply implementation of the exponentiation, each bit of the key determines if a modular multiplication must be done or not. In general, an SPA will also give better results if the hardware architecture is known.

The Differential Power Analysis (DPA) is more sophisticated than the SPA: it consists in performing a statistical analysis (on power consumption curves) of several executions of the same algorithm with different inputs to retrieve the information. The DPA is not necessarily more powerful than the SPA. They are in some way complementary attacks and the countermeasures against them are often complementary and different.

Timing attacks were a main issue (certainly for software implementations) in the past because several optimisations implied algorithms with varying timings depending on the data and/or the cryptographic keys. All the present implementations have to be designed with constant timings (at

least not depending of data and secret keys).

Another newer, very promising attack is the **Electromagnetic Analysis** (EMA) [GMO01]. Each part of a silicon chip is emitting electromagnetic fields depending on the activity on it. EMA attacks can be implemented just as SPA/DPA. Only the measured physical quantities are different. RF attacks permits “two dimensional” resolution (permits local leakage measurement, reducing the effect of other parts of the chip) whereas power consumption is global to the smart card.

5.2 Fault Attacks

Fault attacks consist mainly in applying a combination of environmental conditions that causes the smart card chip to produce a wrong computation that can leak secret information concerning any information in the card (secret stored or computed, personal data or even code from the card).

Such attacks can be carried out with weird values of power supply, clock frequency and duty cycle, working temperature, effect of UV lights, laser, microwaves, ion beam, . . . The necessity of abnormal working conditions sensors is however essential to avoid very huge software and hardware countermeasures (better anticipating than correcting errors).

5.3 Invasive Attacks

Invasive attacks happen when the chip is physically and irreversibly modified. Various kinds of attacks are possible on smart cards, all that can be done with “standard” reverse-engineering with a particular interest for the places where keys, PIN’s and any other secure information can be stored in RAM, EEPROM, ROM and Flash memories. Invasive attacks can help to capture any (secret) information flowing through the buses, registers, ALU and any other part of the hardware.

Invasive attacks on smart cards can be used to disconnect or avoid activation of smart card sensors.

Some memories are write protected by burned fuses. Some attacks try to defeat blown fuse links.

It is important to say that all the attacks we described are not only applicable to smart cards. It can also be applied to many hardware tamper-resistant devices that are not necessary “on-chip” protected (these are often “multi-chip” devices protected by some resin and “external” detectors/-sensors) against attacks. Smart card are probably already far more protected against the described attacks.

6 Software Counter-measures

Software counter-measures are today able to counter most known side channel analysis and fault attacks. Nevertheless, the cost in term of performances is very high. Full protection against fault attacks can require multiple execu-

tions of identical computations implying, however, speed performances that are twice worst.

Against DPA/SPA, we should differentiate the protections for public-key and for secret key algorithms. Secret key algorithms like DES and AES can be protected very efficiently using boolean (arithmetic for some other algorithms) masking. The message to encrypt and the key are masked using a random value (the mask) that permits to avoid having message and key in clear, not only when they are stored in memory but also during processing with the cryptographic algorithm itself. It is furthermore necessary, during non linear phases of the algorithm to use modified S-Boxes (if applicable) taking into account the used random mask [AG01]. Since this mask must be changed regularly, new S-boxes have to be re-computed at that moment, from initial value in ROM. This operation slowdown the execution time. The loss in speed can be very high (factor of 3... 5).

For standard public key cryptography, the problem can be more complex. Indeed, randomizing message and private key may not be enough to protect against SPA. A very careful implementation taking into account the hardware particularities (to reduce evident power consumption variations) has sometimes to be carried-out.

We will now sketch out the main countermeasures that are (or could be) implemented to avoid the described attacks.

7 Hardware Counter-measures

This can be subdivided in sensors/detectors and "active" components.

7.1 Detectors

These are working continuously in the background when the smart card is supplied. They often perform hardware reset and flags activation on detection. Most of (nearly always) implemented detectors include:

- Light/UV detectors,
- High/Low temperature sensors,
- High/low Frequency sensor on external clock,
- High/Low voltage detector.

7.2 Active components

Various active protections can be implemented. Namely:

- Memories encryption (data are never stored in clear text),
- Active (e.g. detect if cut) metal (grid) layer on top of the chip (reduce probing, visual analysis),
- Internal voltage/current regulator,

- Internal clock generator including clock (random) jittering (unstable internal oscillator) to reduce possibility to synchronise power consumption curves (even carried with identical processor state and identical data)
- Adding processor random "dummy" instructions, cycles, interruptions. This will, namely, make DPA attacks more difficult since synchronization between power curves will be less evident.

Furthermore, "scrambled" placing and routing is used to increase the difficulty of reverse engineering, to avoid simple line (bus) probing, and to avoid trivial localisation et deactivation of sensors. This means that, except for the memories, all the buses and overall logic (including the CPU) is widespread everywhere through the chip. Placing and routing is very difficult to be carried-out and space optimisation can not be done.

To avoid fault attacks, checksums on memories can be added as well as redundancy of some critical parts of the chip. Depending on the attack it could also be possible to build some "detectors" able to react when some part of the chip is submitted to the attack.

Further to memory encryption, encryption (scrambling) of transmitted data on buses is sometimes considered to reduce information leakage.

Probably one of the most promising methods to reduce side channel attacks consists in implementing the whole or some critical parts of the smart card chip in dual rail/logic avoiding overall variations of power consumption. This can also, in some implementation, reduce commutations effects.

Asynchronous design technology is being investigated by some smart card manufacturers as an alternative to build smart card chip having good performances against power analysis. It is then impossible the synchronise DPA curves on the rising/falling clock edges, because all the notion of clock has disappeared.

8 Conclusions

The smart card is following the same evolution as PC but with a couple of years of delay (e.g. first commercial smart cards with 32-bit core are available only since very few years).

The smart card is now moving from a close manufacturer dependant OS to an open card multi-applicative system like Java Card™. To increase OS performances as well as cryptographic applications ones, smart card CPU are more and more designed in a way to improve capabilities directly linked to smart cards: new cryptographic instructions are introduced to improve cryptographic flexibility and reduce the importance of crypto-co-processors. The same work is done for Java dedicated instructions.

The smart card is more and more secure, all the manufacturers develop and implement state-of-the-art countermeasures on all known attacks both in software and in hardware.

The smart card is therefore the smartest, low cost, security token.

9 Acknowledgements

We would like particularly to thank Jacques Fournier for his relevant text improvements and his important help to improve our English.

References

- [AG01] M.-L. Akkar and C. Giraud. An implementation of DES and AES, secure against some attacks. In *Pre-Proc. of CHES2001 - Workshop on Cryptographic Hardware and Embedded Systems, Paris, France* (edited by C.K. Koç, D. Naccache and C. Paar), To be Published in Lecture Notes in Computer Science, pp. 315–325, May 14–16, 2001.
- [BRL01] D. Boneh, DeMillo R.A. and R.J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of CRYPTOLOGY*, 14:101–119, 2001.
- [BSS99] I.F. Blake, G. Seroussi and N.P. Smart. Elliptic curves in cryptography. In *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1999.
- [Che00] Z. Chen. *Java Card™ Technology for smart Cards: Architecture and Programmer's Guide*. Addison Wesley, 2000.
- [DKL⁺00] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater and J.-L. Willems. A practical implementation of the timing attack. In *Proc. CARDIS'98, Smart Card Research and Applications, Louvain-la-Neuve, Belgium* (edited by J.-J. Quisquater and B. Schneier), vol. 1820 of *Lecture Notes in Computer Science*, pp. 167–182. Springer, 2000.
- [DQ00] J.-F. Dhem and J.-J. Quisquater. Recent results on modular multiplications for smart cards. In *Proc. CARDIS'98, Smart Card Research and Applications, Louvain-la-Neuve, Belgium* (edited by J.-J. Quisquater and B. Schneier), vol. 1820 of *Lecture Notes in Computer Science*, pp. 336–352. Springer, 2000.
- [GMO01] K. Gandolfi, C. Mourtel and F. Olivier. Electromagnetic analysis: concrete results. In *Pre-Proc. of CHES2001 - Workshop on Cryptographic Hardware and Embedded Systems, Paris, France* (edited by C.K. Koç, D. Naccache and C. Paar), To be Published in Lecture Notes in Computer Science, pp. 255–265, May 14–16, 2001.
- [HP00] H. Handschuh and P. Paillier. Smart card coprocessors for public-key cryptography. In *Proc. CARDIS'98, Smart Card Research and Applications, Louvain-la-Neuve, Belgium* (edited by J.-J. Quisquater and B. Schneier), vol. 1820 of *Lecture Notes in Computer Science*, pp. 386–394. Springer, 2000.
- [KJJ99] P. Kocher, J. Jaffe and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, Santa Barbara, California* (edited by M. Wiener), vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer, 1999.
- [Mon85] P.L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
- [ND96] D. Naccache and M'Raihi D. Cryptographic smart cards. *IEEE Micro*, pp. 14–24, Jun 1996.
- [MvOV97] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [RE97] W. Rankl and W. Effing. *Smart Card Handbook*, 2nd Ed. John Wiley & Sons, 2000.
- [RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In *Communications of the ACM*, volume 21, pp. 120–126, February 1978.
- [Sed88] H. Sedlak. The RSA cryptography processor. In *Advances in Cryptology - EUROCRYPT '87, Amsterdam, The Netherlands* (edited by D. Chaum and W.L. Price), vol. 304 of *Lecture Notes in Computer Science*, pp. 95–105. Springer, 1988.