Securing a SmartCard

Sampath Thodupunuri
School of Electrical Engineering & Computer Science,
Oregon State University, Corvallis, Oregon 97331 -USA.
E-mail: thodupsa@cs.orst.edu

Abstract—
This paper describes techniques for improving smartcard security by randomizing clock signal, using randomized multithreading, using a robust low-frequency sensor, by the destruction of test circuitry, by the use of a restricted program counter and top layer meshes, by eliminating data dependent power consumption and ways to defend against fault induction.

I. Introduction

Smart cards are increasingly prevalent, particularly in Europe, for authentication and payment mechanisms (credit cards, pay-TV access control, public transport payment, medical records, personal identity, mobile phone SIMs, etc.). They present a harder target for the criminal underworld than their magnetic strip counterparts. Nonetheless, there is sufficient economic gain in cracking smart cards. Pay-TV is particularly vulnerable since communication with the smart card is typically unidirectional, from the broadcasting source to the set-top box hosting the smart card. Since there is no back channel, it is not possible to identify duplicate smart cards via interactive protocols. Consequently, it is economically attractive to reverse engineer a pay-TV smart card in order to make a large number of duplicates. As smart cards are used in more and more applications, many new opportunities for theft and fraud open up to criminals capable of reverse engineering cards or extracting key material.

The next section introduces attack technologies which determine the environment in which smart cards must survive. A number of hardware level security issues are addressed and the possible counter measures against these attacks to build more secure smart cards are described.

II. Attack Technologies

Hardware level attacks fall into two main categories: invasive and non-invasive attacks.

Invasive Attacks: Reverse engineering is the most extreme form of an invasive attack where the smart card is depackaged and completely analyzed. Monitoring of bus signals is often sufficient to extract data, and can be undertaken by dropping picoprobes on bus lines. If bus signals are hidden (e.g. by a top level metal defense grid), a focused ion beam (FIB) workstation may be used to extract signals. There is also the 'litigation attack'; the attacker first obtains a patent that might possibly have been infringed by a smart card designer, then abuse the legal discovery process to obtain design details. Thus, inline with Kerckhoffs' principle (The security of a cryptosystem must not depend on keeping secret the crypto-algorithm. The security depends only on keeping secret the key.), one has to assume that the design details of a smart card are in the public domain. Another attack technique, used in the context of an invasive microprobing attack, is to use a laser to shoot away alarm circuitry, or protective circuitry such as access control matrices which allow certain areas of memory to be accessed only after the presentation of certain passwords [1].

Non-Invasive Attacks: More recently non-invasive attacks (sometimes called side channel attacks) have been investigated. An early approach was to measure data dependent timing in order to extract key information [2]. Another approach is differential power analysis [3]where keys are extracted from the differences between runs of data dependent power emissions. A variation of this attack extracts keys via electromagnetic analysis (EMA). The use of 40mm antennas can easily pin-point individual power distribution nets and can be used to build up a profile of on chip power consumption.

Non-invasive attacks are of particular concern because they leave no evidence of tampering can be undertaken relatively quickly. Software techniques are used to introduce randomness to the execution process to make side channel attacks more difficult. However, to reduce data dependent power leakage requires careful circuit design.

Another threat to smart card systems is fault induction. Faults are induced in a number of ways, such as by introducing transients ('glitches') on the power and clock lines and illuminating one or more transistors with a laser. These can cause the processor to malfunction in a predictable and instructive way. For example, when a processor executes a branch instruction, the write to the program counter is often conditionally set late in the clock cycle. An abnormally high clock frequency may result in branches not being taken. Skipping a branch instruction can result in a jump to some failure case being nulled, thereby bypassing security checks.

This paper describes techniques for improving smart card security by the use of following counter measures:

- Randomized clock signal
- Randomized multithreading
- Robust low-frequency sensor
- Destruction of test circuitry
- Restricted program counter
- Top-layer sensor meshes
- Eliminating data dependent power consumption

- Defending against fault induction

III. Counter Measures

A. Randomized Clock Signal:

Many non-invasive techniques require the attacker to predict the time at which a certain instruction is executed. A strictly deterministic processor that executes the same instruction c clock cycles after each reset – if provided with the same input at every cycle – makes this easy. Predictable processor behavior also simplifies the use of protocol reaction times as a covert channel.

The obvious countermeasure is to insert random time delays between any observable reaction and critical operations that might be subject to an attack. If the serial port were the only observable channel, then a few random delay routine calls controlled by a hardware noise source would seem sufficient. However, since attackers can use cross correlation techniques to determine in real-time from the current fluctuations the currently executed instruction sequence, almost every instruction becomes an observable reaction, and a few localized delays will not suffice.

It is therefore strongly recommended introducing timing randomness at the clock-cycle level. A random bit-sequence generator that is operated with the external clock signal should be used to generate an internal clock signal. This will effectively reduce the clock frequency by a factor of four, but most smartcards anyway reduce internally the 3.5 MHz provided for contact cards and the 13 MHz provided for contact-less cards.

Hardware random bit generators (usually the amplified thermal noise of transistors) are not always good at producing uniform output statistics at high bit rates, therefore their output should be smoothed with an additional simple pseudo-random bit generator.

The probability that n clock cycles have been executed by a card with a randomized clock signal after c clock cycles have been applied can be described as a binomial distribution.

$$p(n,c) = 2^{-c} \left(\begin{array}{c} c \\ 2n \end{array} \right) \left(\begin{array}{c} c \\ 2n+1 \end{array} \right) \quad (1)$$

$$\approx \sqrt{8/\pi c} e^{-8/c.(n-c/4)^2} asc \rightarrow \infty \quad (2)$$

So for instance after we have sent 1000 clock cycles to the smartcard, we can be fairly sure (probability $\doteq 1$) that between 200 and 300 of them have been executed. This distribution can be used to verify that safety margins for timing critical algorithms – such as the timely delivery of a pay-TV control word – are met with sufficiently high probability.

Only the clock signals of circuitry such as the serial port and timer need to be supplied directly with the external clock signal, all other processor parts can be driven from the randomized clock.

A lack of switching transients during the inactive periods of the random clock could allow the attacker to reconstruct the internal clock signal from the consumed current. It is therefore essential that the processor shows a characteristic current activity even during the delay phases of the random clock. This can be accomplished by driving the bus with random values or by causing the microcode to perform a write access to an unused RAM location while the processor is inactive.

B. Randomized multithreading:

To introduce even more non-determinism into the execution of algorithms, it is conceivable to design a multithreaded processor architecture that schedules the processor by hardware between two or more threads of execution randomly at a per instruction level. Such a processor would have multiple copies of all registers (accumulator, program counter, instruction register, etc.), and the combinatorial logic would be used in a randomly alternating way to progress the execution state of the threads represented by these respective register sets.

The simple 8-bit microcontrollers of smartcards do not feature pipelines and caches and the entire state is defined only by a very small number of registers that can relatively easily be duplicated. The only other necessary addition would be new machine instructions to fork off the [4]other thread(s) and to synchronize and terminate them. Multithreaded applications could interleave some of the many independent cryptographic operations needed in security protocols. For the remaining time, the auxiliary threads could just perform random encryptions in order to generate a realistic current pattern during the delay periods of the main application.

C. Robust low-frequency sensor:

Bus-observation by e-beam testing becomes much easier when the processor can be clocked with only a few kilohertz, and therefore a low-frequency alarm is commonly found on smartcard processors. However, simple high-pass or low-pass RC elements are not sufficient, because by carefully varying the duty cycle of the clock signal, we can often prevent the activation of such detectors. A good low frequency sensor must trigger if no clock edge has been seen for longer than some specified time limit (e.g., 0.5 ms). In this case, the processor must not only be reset immediately, but all bus lines and registers also have to be grounded quickly, as otherwise the values on them would remain visible sufficiently long for a voltage contrast scan.

Even such carefully designed low-frequency detectors can quite easily be disabled by laser cutting or FIB editing the RC element. To prevent such simple tampering, an intrinsic self-test must be built into the detector. Any attempt to tamper with the sensor should result in the malfunction of the entire processor. Such a circuit can be designed that tests the sensor during a required step in the normal reset sequence. External resets are not directly forwarded to the internal reset lines, but only cause an additional frequency divider to reduce the clock signal. This then activates the low-frequency detector, which then activates the internal

reset lines, which finally deactivate the divider. The processor has now passed the sensor test and can start normal operation. The processor is designed such that it will not run after a power up without a proper internal reset. A large number of FIB edits would be necessary to make the processor operational without the frequency sensor being active.

Other sensor defenses against invasive attacks should equally be embedded into the normal operation of the processor, or they will easily be circumvented by merely destroying their signal or power supply connections.

D. Destruction of test circuitry*:*

Microcontroller production has a yield of typically around 95%, so each chip has to be thoroughly tested after production. Test engineers – like microprobing attackers – have to get full access to a complex circuit with a small number of probing needles. They add special test circuitry to each chip, which is usually a parallel/serial converter for direct access to many bus and control lines. This test logic is accessible via small probing pads or multiplexed via the normal I/O pads. On normal microcontrollers, the test circuitry remains fully intact after the test. In smartcard processors, it is common practice to blow polysilicon fuses that disable access to these test circuits. However, attackers have been able to reconnect these with microprobes or FIB editing, and then simply used the test logic to dump the entire memory content.

Therefore, it is essential that any test circuitry is not only slightly disabled but structurally destroyed by the manufacturer. One approach is to place the test interface for chip n onto the area of chip n + 1 on the wafer, such that cutting the wafer into dies severs all its parallel connections. A wafer saw usually removes a 80-200 m wide area that often only contains a few process control transistors. Locating essential parts of the test logic in these cut areas would eliminate any possibility that even substantial FIB edits could reactivate it.

E. Restricted program counter*:*

Abusing the program counter as an address pattern generator significantly simplifies reading out the entire memory via microprobing or e-beam testing. Separate watchdog counters that reset the processor if no jump, call, or return instruction is executed for a number of cycles would either require many transistors or are too easily disabled. Instead, it is recommended simply not providing a program counter that can run over the entire address space. A 16-bit program counter can easily be replaced with the combination of a say 7-bit offset counter O and a 16-bit segment register S, such that the accessed address is S + O. Instead of overflowing, the offset counter resets the processor after reaching its maximum value. Every jump, call, or return instruction writes the destination address into S and resets O to zero. The processor will now be completely unable to execute more than 127 bytes of machine code without a jump, and no simple FIB edit will change this. A simple machine-code postprocessor must be used by the programmer to insert jumps to the next address wherever unconditional branches are more than 127 bytes apart. With the program counter now being unavailable, attackers will next try to increase the number of iterations in software loops that read data arrays from memory to get access to all bytes. This can for instance be achieved with a microprobe that performs a glitch attack directly on a bus-line. Programmers who want to use 16-bit counters in loops should keep this in mind.

F. Top-layer sensor meshes*:*

Additional metallization layers that form a sensor mesh above the actual circuit and that do not carry any critical signals remain one of the more effective annoyances to microprobing attackers. They are found in a few smartcard CPUs such as the ST16SF48A or in some battery-buffered SRAM security processors such as the DS5002FPM and DS1954.

A sensor mesh in which all paths are continuously monitored for interruptions and short-circuits while power is available prevents laser cutter or selective etching access to the bus lines. Mesh alarms should immediately trigger a countermeasure such as zeroizing the non-volatile memory. In addition, such meshes make the preparation of lower layers more difficult, because since the etch progresses unevenly through them, their pattern remains visible in the layers below and therefore they complicate automatic layout reconstruction. Finally, a mesh on top of a polished oxide layer hides lower layers, which makes navigation on the chip surface for probing and FIB editing more tedious.

The implementations of sensor meshes in fielded products however show a number of quite surprising design flaws that significantly reduce the protection. The most significant flaw is that a mesh breach will only set a flag in a status register and that zeroization of the memory is left completely to the application software. Note that a common read-out technique involves severely disabling the instruction decoder, therefore software checks for invasive attacks are of little use.

A well-designed mesh can make attacks by manual microprobing alone rather difficult and more sophisticated FIB editing procedures will be required to bypass it. Several techniques can be applied here. The resolution of FIB drilling is much smaller than the mesh line spacings, therefore it is no problem to establish contact through three or more metal layers and make deeply buried signals accessible for microprobing via a platinum or tungsten pad on top of the passivation layer. Alternatively, it is also possible to etch a larger window into the mesh and then reconnect the loose ends with FIB metal deposits around it.

G. Eliminating data dependent power consumption*:*

Simple binary encoding of data, [5]where one wire is used to propagate one bit, results in power consumption proportional to the number of state changes. Data transmission along a bus consumes considerable power due to wire capacitance. Bus activity is observed as the Hamming weight

of the state changes.

One approach to reducing data dependent power consumption is to use an alternative data encoding scheme. For example, one-hot data encoding (1-of-n codes) consume constant power to transmit data since just one wire transitions for every symbol. 1-of-2 (dual-rail) and 1-of-4 data encodings are commonly used in self-timed circuit design [6].

1-of-n encoding is not sufficient to guarantee a data independent power signature. Firstly, the path taken by each wire is likely to vary, which can result in a difference in wire load. This problem may be addressed by careful layout of long buses and careful floor planning to keep random wire lengths under control.

A second problem is the logic complexity variation between each wire. Care has to be taken when choosing standard cells to minimize this effect. Often this constraint means that silicon area has to be sacrificed for greater security. However, most smart card chips are dominated by memory requirements so additional area for logic adds little to the final size of the chip.

Data dependent control presents a third problem. For example, a hardware multiplier implemented using a shift-and-conditional-add algorithm uses data-dependent power if the add operation is only undertaken when required. A similar problem occurs if early completion detection is used. Ensuring that the same operations occur regardless of the data seems to be the best way to deal with this problem. An alternative approach is to add random noise to the power signature, but randomness can often be removed by signal averaging over repeated runs.

H. Defending against fault induction*:*

Self-timed designs are immune to clock glitch attacks. Where the clock is required, for example in the clocked serial smart card interface, it is easy to arrange that data corruption should not result in sensitive data leakage.

A speed independent (SI) asynchronous circuit naturally adapts to power supply voltage, thereby making power supply glitch attacks less successful. However, self-timed circuits cannot protect components like EEPROM which is often used to store keys and error counters. Dual-rail encoding is often used to construct SI circuits. Two wires are used to encode three states: clear, logic-0 and logic-1 (see Table 1). The *unused* fourth state (often 11) is typically not used by dual-rail circuits. However, from the stand point of managing faults, the unused state must be explicitly handled as an error condition which we shall call alarm.

A1	**A0**	**meaning**
0	0	clear
0	1	logical 0
1	0	logical 1
1	1	alarm

Table 1: dual-rail encoding with alarm signal defined

Single point fault induction results in one of three behaviors:

1. Data can be suppressed which results in the circuit deadlocking in the clear state.

2. A clear state is forced into a logic-0 or logic-1 state which typically results in deadlock in the control path because an extra data item has miraculously appeared. Deadlock under these circumstances can be guaranteed.

3. A logic state is forced into the alarm state resulting in data corruption. The alarm signal can be propagated rapidly resulting in data being deleted and a global alarm raised.

The fact that SI circuits deadlock when a fault occurs is very useful because it paralyses the chip until a hard reset is performed. This is in contrast to clocked circuits where induced faults are in the same league as pulses due to static and dynamic hazards, which are considered normal operating behavior in many synchronous designs.

IV. Conclusion

A basis for understanding the mechanisms that make microcontrollers particularly easy to penetrate is presented. With the restricted program counter, the randomized clock signal, and the tamper-resistant low-frequency sensor, some low-cost countermeasures that are considered to be quite effective against a range of attacks have been shown. There are of course numerous other more obvious countermeasures against some of the commonly used attack techniques which cannot be covered in detail in this overview. Examples are current regulators and noisy loads against current analysis attacks and loosely coupled PLLs and edge barriers against clock glitch attacks. A combination of these together with field sensors and randomized clocks or perhaps even multithreading hardware in new processor designs will hopefully make high-speed non-invasive attacks considerably less likely to succeed. Other countermeasures in fielded processors such as light and depassivation sensors have turned out to be of little use as they can be easily bypassed.

There is really no effective short-term protection against carefully planned invasive tampering involving focused ion-beam tools. Zeroization mechanisms for erasing secrets when tampering is detected require a continuous power supply that the credit-card form factor does not allow. The attacker can thus safely disable the zeroization mechanism before powering up the processor. Zeroization remains a highly effective tampering protection for larger security modules that can afford to store secrets in battery-backed SRAM, but this is not yet feasible for the smartcard package.

References

[1] O. Kommerling and M. G. Kuhn: "Design principles for tamper-resistant smartcard processors," *First USENIX Workshop on Smartcard Technology,*, TUGboat Volume 9, Issue 1 (1988)in (Chicago, Il), pp. 9-20, USENIX, May 1999.

[2] P. C. Kocher: "Timing attacks on implementations of diffe-hellman, RSA, DSS, and other systems," in *Proc. 16th International Advances in Cryptology Conference CRYPTO '96*,pp. 104-113, 1996.

[3] J. Jaffe,P. Kocher,and B. Jun, ,: "Differential power analysis," in *Proc. 19th International Advances in Cryptology Conference CRYPTO '99*,pp. 388-397, 1999.

[4] P. Pallier H. Handschuh and J.Stern: "Probing attacks on tamper resistant devices," in *Cryptographic Hardware and Embedded Systems - CHES 1999, Ç. K. Koç and C.Paar, Eds.,1999*,Lecture Notes in Computer Science No. 1717, pp. 187-204.

[5] S. W.Moore, R. J. Anderson and M. G. Kuhn,: "Improving Smartcard Security using Self-Timed Circuit Technology," in *Fourth AciD-WG Workshop,* ,Grenoble, ISBN 2-913329-44-6, 2000.

[6] P. Adi Shamir: "Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies," in *Cryptographic Hardware and Embedded Systems - CHES 2000, Ç. K. Koç and C.Paar, Eds.,1999*,Lecture Notes in Computer Science No. 1965, pp. 71-77.