



# Side Channel Attacks on Smart Cards

## A cryptovision whitepaper

Version 1.0 (August 2009)

cv cryptovision GmbH  
 Munscheidstr. 14  
 45886 Gelsenkirchen  
 +49-209-167-2477  
 info@cryptovision.com

*Countermeasures against side channel attacks – e.g. power attacks, based on an analysis of the power consumption, or electromagnetic attacks, which are based on the measurement of electromagnetic emanations – play an important role in modern implementations of cryptographic algorithms on Smart Cards or other security tokens. This has led to significant higher efforts in the implementation of cryptographic libraries for these platforms. At least during two stages of the development process, measurements on the hardware platform become necessary. After giving a short introduction, this article outlines the some of the current threats and possible countermeasures. One focus is set on describing how this problems influences the complete development process.*

Today, main reason for using something like a Smart Card is, that such a device can be seen as a personal security environment; processors used for such cards are so-called Security ICs, they are designed to withstand all kind of potential attacks. In the past, focus here was on securing the microprocessor and the memory against invasive attacks like micro-probing; additional shielding or the use of sensors (for noticing such an attack and erasing memory with sensitive data) are effective countermeasures in this case.

### **Attacks during operation**

A new idea – in the meantime known for a couple of years – is, to attack the device by observing the processors behaviour *while it is in operation*; this is a

completely different approach and exposed a lot of potential and surprising weaknesses when it was first introduced. First steps have been to observe the influence of sensitive data on the computation time (so-called *timing attacks*), a mayor breakthrough has been P. Kocher's work in the years 1997-98. The new idea was to collect information by observing the current consumption of a device like a Smart Card (which doesn't have a battery). This is what is called *power analysis* today and the method has been extended and improved a much in the last years.

Most important thing to mention: These are not theoretical attacks. Examples like the PIN or *power break* attack (which can be seen as a successor of side channel attacks) are very easily to mount on cards which contain no adequate countermeasures

even if the attacker possesses almost no special knowledge or equipment. Attacks based on power analyses might higher the border, because an attacker has to invest in special hardware equipment (like an expensive oscilloscope). But in principal, all of these attacks are working in the real work and for a developer of cryptographic modules, one important question is, against what kind of attacker an implementation should be secured. As it can be seen from the following, integration of all possible countermeasures is not an easy task.

### **Actual Threats**

Side channel attacks normally try to take advantage out of the fact, that the behaviour of the computation is at many points directly influenced by the input data; under certain circumstances, this can be exploited if information about sensitive data can be seen from the outside via channels, which existence normally is not realized.

Although it might look like a little bit artificial, depending on the nature of the leaking channel one can distinguish between attacks against the hardware or the software which implements the cryptographic routines. In reality, normally exact distinction lines are not easy to find; practical reason for such an approach is to get a better feeling who (the hardware manufacturer or the developer of the software) is mainly responsible for which kind of problems.

Securing a processor against invasive attacks for ensuring that an attacker is not able to read out critical data like private keys is not topic of this talk. Normally, hardware manufacturers are familiar with the problems in this area. Of course, a software developer should about the remaining ones; only then he is able to consider the respective countermeasures.

Beneath the invasive attacks, actual problems are attacks which interfere the operations of the processor while he is working on critical operations. One classic example is an attack based on DFA (*Differential Fault Analysis*) on RSA. The principal idea of the so-called *Bellcore attack* is the

observation, that an attacker can completely break the system if the implementation is using the CRT (*Chinese Remainder Theorem*) and he is able to manipulate or disturb the computation.

Once known, it is not too complicated to come around this problem (additional checks can prevent the information leakage); the main problem with such an experience is, that the weakness was substantial for many implementations and discovered so late.

### **Possible Countermeasures**

Of course, today normally every microprocessor used for smart card applications contains some hardware countermeasures to prevent these kinds of attacks. This might be something like bus encryption (which can be also seen as a countermeasure against invasive attacks); other examples are random process interrupts which are automatically induced by the processor and randomise the actual steps an operation takes or inducing noise on the current by additional power consuming devices on the chip.

But as already mentioned, focus here are countermeasure which can be integrated in software. In principle, one should distinguish the following two approaches: First, work on the algorithmic level (which should be comparable on different hardware platform), second the use of special coding techniques which should address potential weaknesses of the actual hardware.

Of course, some methods doesn't fit in this kind of classification (e.g. additional checks for proving the correctness of an operation to prevent a DFA-like attack), but for getting an impression what can be done, it makes sense to have a look at these two kind of mechanisms first.

Actually, randomisation is the most easiest and commonly used technique to prevent such kind of attacks. The basic idea is simple: One can try to "scramble" the implemented algorithm. By the use of additional random input it should be ensured, that if one concrete computation is done a couple of times, the performed steps are different from case

to case, especially if secret data (like cryptographic keys) are used during the computation.

A typical example is a private key operation in case of RSA: Basically, this is a modular exponentiation where the private key is used as the exponent (and therefore has to be secured against spoofing). Basic idea is to randomise this operation by computing  $m^{x^R}$  and  $m^R$  for some random number  $R$  instead of computing  $m^x$ ; multiplying these two values gives the correct result, but during the critical operation (the exponentiation) the secret was not directly used.

The method is a classical blinding technique, it can be used in many variations. Especially, if the algorithm used is not consisting only of one basic operation (like it is the case with RSA), but consists of a more complex operation (e.g. creation of digital signatures with ECDSA).

Most important, basic design principal is of course to avoid the use of conditional jumps where the conditions are depending from secret data; one always has to be afraid that the behaviour of the implementation might be visible from the outside. But this countermeasure alone is not enough for getting a secure implementation, other additional techniques must be considered.

To focus only on modifying the algorithm looks very attractive, especially because this is work which can be used for implementations on many different platforms. But there is one important thing to realize: Often information leakages are very different on different hardware; so some method which is useful on one processor might lead to a problem on another one.

So one has to know the underlying hardware in detail, if he wants to ensure, that the problems are addressed correctly. This means, before designing a module an analysing phase where the behaviour of basic operations is studied, becomes important. Problems analysed during this step might be multiplications or copy operations between different memory areas (e.g. EEPROM, RAM, ...) on the chip; in the latter case, either getting information about the

addresses used or about the content of the memory can be subject of an attack.

After this preparing phase, there should be enough knowledge about the potential weaknesses of the processors and the respective countermeasures. The following design and implementation phase can then integrate methods which are mentioned above or additionally tricks like initialisation of registers with random input before using them or parallel operation of host and crypto-coprocessor (if something like this is available on the chip).

But the latter is an typical example, what goal can be reached by such kind of work and what problems still might remain: Of course, the additional noise created by parallel operation of more than one device on the chip is helpful when looking at the power consumption of the complete processor. But sometimes, this is simply not enough. One problem might be, that the attacker is able to distinguish between various current consumers (e.g. by starting a SEMA / DEMA attack where the location of the probing sensor can be changed in three dimensions) and where he gets (because of the nature of the measurement setup) data which are filtered according to the frequency. Therefore, for a real implementation it is necessary to combine various of these techniques.

## **Consequences**

One important thing to mention is, that the integration of effective countermeasures directly affects the total costs of an implementation. Additional tasks (like measurements on real hardware before starting the design phase) are not for free and therefore influence the total development costs.

Another problem is the availability of solutions: Ideally, a Smart Card using a new version of a microprocessor should be available very soon after the availability of the hardware; of course this is not easy to realize if somebody wants to have a closer look on the hardware before starting the development.

In the following, two consequences are described a little bit more detailed: One is, how the end product (typically a library for doing cryptographic operations) is affected in terms of code size and other resource consumption; the other topic is, what changes for the development process are necessary.

The actual talk will give some concrete examples for the resource consumption in terms of code size and running times for implementation on current Smart Card processor (with and without countermeasures) for libraries on two different platforms.

Of course the values given depend on a couple of circumstances (e.g. the actual processor which is used and what countermeasures are integrated), but as a rule of thumb, one can think about additional 20-30% resource consumption in case of the integration of all kinds of necessary countermeasures.

Most important is a complete change in the design phase: knowing the hardware in advance and understanding the whole problem (e.g. the cryptographic aspects) is crucial for getting the highest level of security. So the necessary skill level of the developers is different today: The focus is not on the engineering skills, mathematical understanding of the whole problem is important. One has to understand, how attacks work, to find ways to prevent them.

There is on other important point to mention: How can one get assurance, that integrated countermeasures are really effective? In the past, main approach was doing own measurements or measurements by an examination of the product through a neutral instance (e.g. in form of a CC or ITSEC evaluation); all of this normally takes place when the implementation is finished or at least almost done. But if somebody is interested in the effectiveness of some special method used, this procedure has a couple of drawbacks: One of the problems is, that at the end there are so many countermeasures integrated that it is hard to decide, what exactly the influence of which method is; at this point it is normally not easy to encapsulate the mechanism one likes to analyse.

This is the reason, why the development process of such a module should contain another cycle: After finishing on step of the development, one should not only think about quality management in terms of extensive testing, also doing measurements concerning security issues are necessary. In case of any suspicious behaviour, it is then early enough to correct the problem.

## **Conclusion**

Some people thought, that with the use of public key cryptography the race between cryptographers (who are searching and designing secure algorithms and protocols) and cryptanalysts (who try to break these systems) could come to an end. But problems in the area of side channel attacks are showing, that this is definitely not the case: Theoretical security of an algorithm or a protocol can be something completely different than the security of a real world implementation. This has to be acknowledged.

Of course, something like computational efficiency of operations, code sizes or costs for the development are all important points; but for a product like a Smart Card, the priority should lie on the security of the implementation.

## **References**

For more details about cryptography basics refer to: [www.cryptovision.com](http://www.cryptovision.com).