Chapter I

Smart Card Security

Kostas Markantonakis [I.1], Keith Mayes [I.2], Michael Tunstall [I.3], Fred Piper [I.4], and Damien Sauveron [I.5] [I.6]

.

I.1 Introduction

I.2 Smart cards and Cryptography

I.3 Smart card Specific Attacks

Sensitive systems that are based on smart cards use protocols and algorithms that have usually been subjected to rigourous analysis by the cryptographic community. Attackers have therefore sought other means to circumvent the security of the protocols and algorithms used in smart card based systems. As smart cards are small, portable devices they can easily be put in a situation where their behaviour can be observed. The simplest form of analysis of this type is intercepting al the

[I.1] Smart Card Centre, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, k.markantonakis@rhul.ac.uk

[I.2] Smart Card Centre, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, keith.mayes@rhul.ac.uk

[I.3] Smart Card Centre, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, m.j.tunstall@rhul.ac.uk

[I.4] Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, f.piper@rhul.ac.uk

[I.5] Smart Card Centre, Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK

[I.6] XLIM – UMR CNRS 6172, University of Limoges, 123 avenue Albert Thomas - 87060 LIMOGES Cedex, FRANCE, damien.sauveron@unilim.fr, http://damien.sauveron.free.fr/

communication between a smart card and its reader. Tools are readily available on the Internet [22] that allow the commands to and from a smart card to be logged and/or modified. This problem is relatively easy to solve with secure sessions, as proposed in [14], but highlights the ease of man-in-the-middle type attacks against smart cards.

More complex attacks can be realised by monitoring information that leaks naturally during a smart cards processing of information, referred to as side channel attacks. Smart cards can also be attacked by inducing a fault during their normal processing to change the chips behaviour. Comparing the two results can then be used to make deductions about secrets held by a smart card. These two classes of attack are described in more detail in the following sections.

I.3.1 Side Channel Attacks

The first example of a side channel attack was proposed in [17]. This involved observing the differences in the amount of time required to calculate a RSA signature to derive the secret key. This attack was conducted against a PC implementation but a similar analysis can be applied to smart card implementations. It would be expected to be more efficient against a smart card as more precise timings can be achieved with an oscilloscope or proprietary readers. An example of equipment capable of acquiring this sort of information is shown in Figure I.1. The I/O race can be seen on the oscilloscope as the yellow trace that will allow the exact amount of clock cycles a command took to be determined (the smart card being analysed is in the reader in the top left corner of the image). For this reason, the amount of time taken for a command is often constant or, if variable, not related to any secret information.

The most common form of side channel attack is the analysis of the power consumption, originally proposed in [18]. This is measure by placing a resistor in series with a smart card, between the card and earth. The potential difference across this resistor is measured with an oscilloscope. This gives a measurement of current in the circuit between the chip and the resistor. The equipment shown in Figure I.1 is capable of taking these sort of measurements. The current at each point in time during a command is shown as the blue trace on the oscilloscope. An enlargement of this trace is shown on the computer screen, where the minimum, average and maximum of the points represented by one pixel can be seen. This is referred to as the power consumption in the literature, and the attacks based on these measurements are power analysis attacks. There are two main types of power attack; these are simple power analysis (SPA) and statistical power analysis.

Simple Power Analysis. This is the analysis of one power consumption trace. An attacker will look for repetitive patterns and distinctive events to reverse engineer the algorithm being used and to try and derive any secrets being manipulated. Figure I.2 shows a power consumption trace taken during the execution of a DES implementation, using equipment similar to that shown in Figure I.1, where the 16 rounds of the DES algorithm can be seen. The initial (IP), final (IP^{-1}) and initial

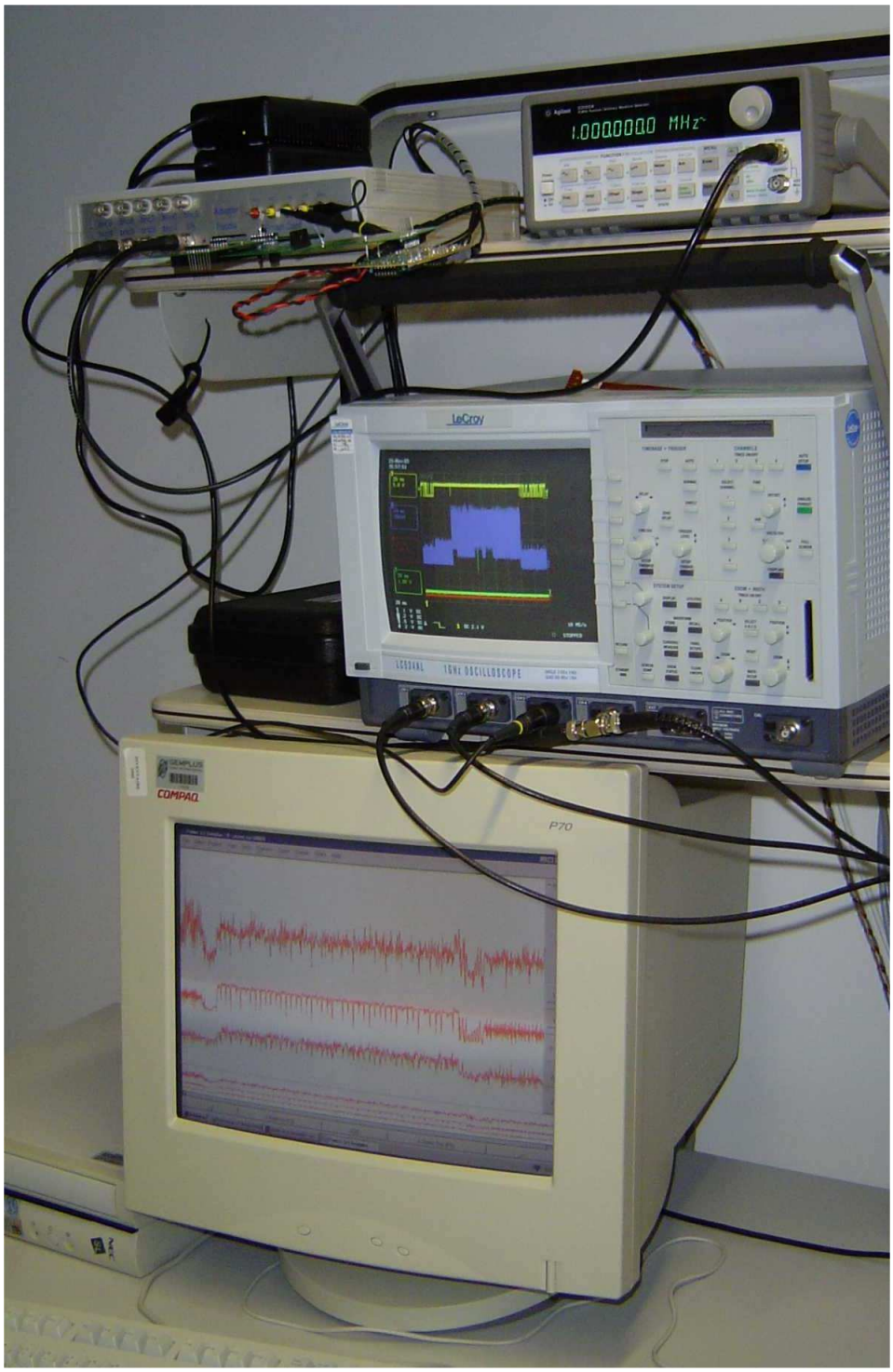

Figure I.1. Data Acquisition Tools.

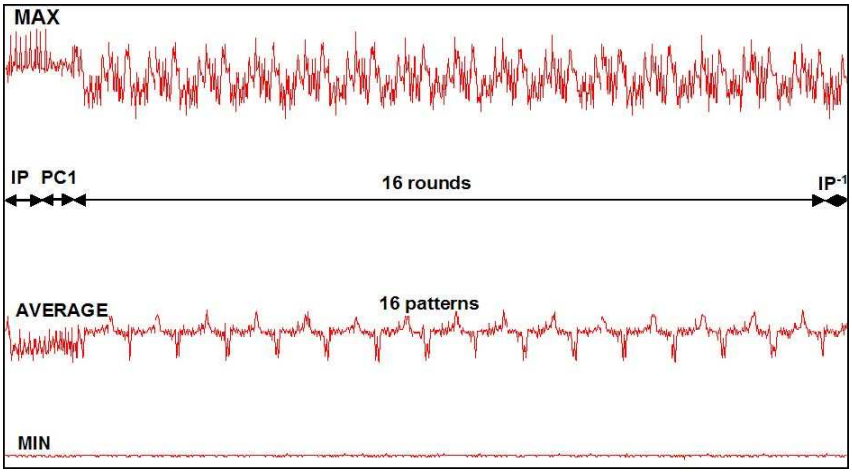

Figure I.2. Power consumption during the execution of a DES implementation.

key (PC1) permutations have also been identified. The initial permutations can be distinguished due to the fact that they manipulate different amounts of data, i.e. the initial permutation will have 8 sets of patterns with 8 features and the initial key permutation will have 8 sets of patterns with 7 features (or 7 sets of 8 depending on the implementation).

A compiler can implement the bitwise permutations used in DES incorrectly. An example of an algorithm that could be produced is given in Algorithm I.1, where each bit in buffer X (a buffer of n bits, where $(x_0, x_1, \ldots, x_n)_2$ is the binary representation) is tested to determine whether a bit in buffer Y should be set.

Algorithm I.1 Unsecure bitwise permutation function

Input: $X = (x_0, x_1, \ldots, x_n)_2$, $P[\cdot]$ containing the indexes of the permutation
Output: $Y = (y_0, y_1, \ldots, y_n)_2$
1. $Y := 0$;
2. for $i = 0$ to $n - 1$
3. if $(x_{P[i]} = 1)$ then $y_i := 1$;
4. return Y;

end.

This is not a secure implementation due to the conditional test present in step 3. The setting on one bit in buffer Y will take a certain amount of time when the tested bit in buffer X is equal to 1. If a power consumption trace for this function is compared to a known key, for example all zeros, the point where the two traces differ will reveal the first manipulated bit that is not equal to the known key. In the case where the known key is all zeros where the power consumption diverges the manipulated bit of the unknown key is equal to one, and all previous bits were equal to 0. The power consumption trace can then be shifted to take into account the time difference due to this bit and the process repeated for the rest of the key.

This can be avoided by implementing the bitwise permutation as shown in Al-

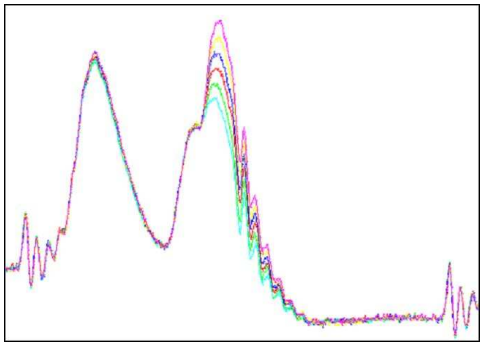

Figure I.3. Superimposed acquisitions of one clock cycle showing the data dependence of the power consumption.

gorithm I.2, where each bit is taken separately for buffer X and assigned to buffer Y. This is usually more time consuming than Algorithm I.1 as each bit needs to taken from the relevant machine word.

Algorithm I.2 Secure bitwise permutation function

Input: $X = (x_0, x_1, \ldots, x_n)_2$, $P[\cdot]$ containing the indexes of the permutation
Output: $Y = (y_0, y_1, \ldots, y_n)_2$
1. for $i = 0$ to $n - 1$
2. $\quad y_i := x_{P[i]}$;
3. return Y;
end.

It is not always possible to change an algorithm in this manner. The modular exponentiation used RSA is an example of an algorithm that it is exceedingly difficult to secure against this type of attack as branches in the algorithm cannot be avoided. An example of a power consumption trace that shows the square-and-multiply algorithm is shown in [15].

Statistical Power Analysis.

Countermeasures.

Constant Execution

Random Delays can be inserted at different points in the algorithm being executed i.e. a dummy function that takes a random amount of time to execute can be called. This does not provide a countermeasure, but creates an extra step for an attacker. In order to conduct any power analysis an attacker needs to synchronise the power consumption acquisitions *a posteriori*. The effect of conducting statistical power analysis attacks in the presense of random delays is detailed in [10].

Randomisation: [8, 1, 15]

Randomised Execution is the manipulation of data in a random order so that an attacker does not know what is being manipulated. In the case of Algorithm I.1 an attacker would not know which bit is being manipulated at any given point in time. Given the $n!$ possible combinations an attack by simple power analysis would be extremely difficult.

This also inhibits any statistical analysis of the power consumption, as this relies on the same unknown variable being treated at the same point in time. As an attacker cannot know the order in which the data has been treated, this provides an extremely efficient countermeasure when combined with randomisation. An example of this is given in [20].

I.3.2 Fault Attacks

The first example of a theoretical fault attack was presented in 1997 [7] as a method of injecting faults in to RSA signature generation when using the Chinese Remainder Theorem. This was followed by several other attacks on public key algorithms [6, 16] and an equivalent for private key algorithms was proposed in [5], usually based on the effect of a 1 bit error during the computation of the algorithm under study. As no implementations were forthcoming interest in this type of attack waned. One of the first publications describing an implementation of this type of attack was presented in 2002 [2], and described an implementation of the attack against the RSA signature scheme and some countermeasures. This revitalised interest in this type of attack and is now an important aspect of smart card security.

Modelling the Effect of a Fault. There are several known mechanisms for injecting faults into microcontrollers. These include variations in the power supply to create a glitch or spike [13], white light [23], laser light [3] and eddy currents [21]. Figure I.4 shows equipment that can be used to inject faults with white light (left image) and laser light (right image). The models for the faults that can be created by these effects can be summarised as follows:

Data randomisation: the adversary could change the data to a random value. However, the adversary does not control the random value and the new value of the data is unknown to the adversary.

Resetting Data: the adversary could force the data to the blank state, i.e., reset a given byte, or bytes, of data back to `0x00` or `0xFF`, depending on the logical representation.

Modifying opcodes: the adversary could change the instructions executed by the chip's CPU. This will often have the same effect as the previous two types of attack. Additional effects could include removal of functions or the breaking of loops. The previous two models are algorithm dependent, whereas the changing of opcodes is implementation dependent.

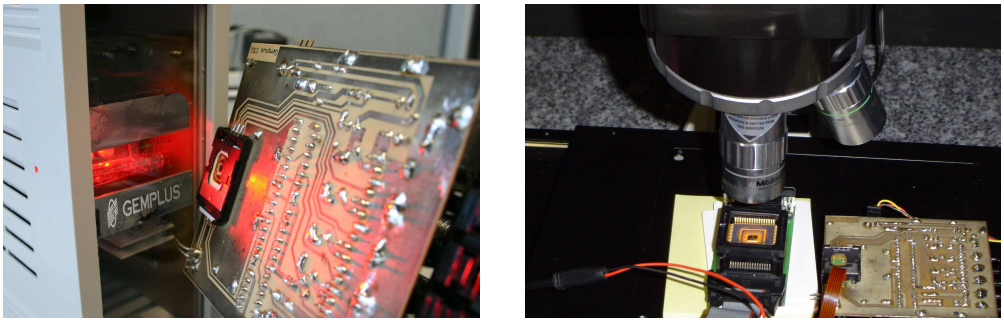

Figure I.4. White and laser light fault injection equipment.

These three types of attack cover everything that an attacker could hope to do to an algorithm. In most cases it is not usually possible for an attacker to create all of these possible faults. Nevertheless, it is important that algorithms are able to tolerate all types of fault, as the fault injection methods that may be realisable on a given platform are unpredictable. While an attacker might only ever have a subset of the above attacks available, if that attack is not taken into account it may have catastrophic consequences the security of a protocol or algorithm.

All of these models generally assume that a fault injected into a chip will change a machine word rather than a single bit. The initial models of a 1 bit fault do not appear to be realistic. This model was based on studies of faults caused by the effects of the radiation present in the upper atmosphere [24], which is important in the design of vehicles designed to travel in the upper atmosphere or space. The effects of radiation are random, whereas an injected fault is intentional and will target a given point in time.

There have not been any publications that exploit the change of 1 bit of information in a chip. It is possible to use a lasers simulate the effect of radiation in a circuit [19], but smart cards will generally use scrambled/randomised layouts so the possibilities are limited. More success has been achieved by targeting larger areas to provoke a large fault.

Injecting Faults in Algorithms. The first fault attack published in [7] and implemented in [2] can be described as follows: If we assume that the calculation of an RSA signature i.e. $s = m^d \pmod{n}$, where $n = p \times q$, the two primes upon which the security of RSA is based. If this is calculated using the Chinese remainder theorem the following values are calculated,

$$\begin{aligned} s_p &= m^{(d \pmod{p-1})} \pmod{p} \\ s_q &= m^{(d \pmod{q-1})} \pmod{q} \end{aligned} \tag{I.1}$$

which can be combined to form the RSA signature s using the formula $s = a \times s_p + b \times s_q \pmod{N}$, where:

$$\begin{cases} a \equiv 1 \pmod{p} \\ a \equiv 0 \pmod{q} \end{cases} \quad \text{and} \quad \begin{cases} b \equiv 0 \pmod{p} \\ b \equiv 1 \pmod{q} \end{cases}$$

This can be calculated using the following formula:

$$s = s_q + \Big((s_p - s_q) \times q^{-1} \pmod{p}\Big) \times q \tag{I.2}$$

If this is calculate correctly, and then recalculated but with a fault injected during the computation of s_p or s_q, information can be derived on one of the primes used to create n.

If s_q is changed to s'_q, then the signature generated will be of the form $s' = a \times s_p + b \times s'_q \pmod{n}$. The difference between s and s' gives:

$$\begin{aligned} \Delta &\equiv s - s' \\ &\equiv (a \times s_p + b \times s_q) - (a \times s_p + b \times s'_q) \\ &\equiv b(s_q - s'_q) \pmod{n} \end{aligned} \tag{I.3}$$

Hence, as $b \equiv 0 \pmod{p}$ and $b \equiv 1 \pmod{q}$ it follows that $\Delta \equiv 0 \pmod{p}$ (but $\Delta \not\equiv 0 \pmod{q}$) meaning that Δ is a multiple of p (but not of q). Hence, a GCD calculation gives the secret factors of n, i.e. $p = \gcd(\Delta \pmod{n}, n)$ and $q = n/p$.

This attack could potentially be achieved with any of the fault models given above. This is because any fault in one of the modular exponentiations given in Equations I.1 will be enough to conduct the attack. This includes modifying the variable entering the equation before the start of the computation of the modular exponentiation.

Another attack that was proposed shortly after the RSA attack was published targeting DES [5]. This assumed a 1 bit fault modified the calculation of the algorithm in the last couple of rounds of DES. This was generalised to allow for a larger fault and became a frequently cited attack within the smart card industry. The most popular form of this attack is described in [3], and is repeated here.

DES is considered DES to be a transformation of two 32 bit variables (L_0, R_0), i.e. the message, though sixteen iterations of the function as shown in Figure I.5 to produce the ciphertext (L_{16}, R_{16}). This is overly complex for the requirements of this paper, and the bitwise permutations will not be considered. This means that the round function can be simplified to:

$$\begin{aligned} R_n &= S(R_{n-1} \oplus K_n) \oplus L_{n-1} \\ L_n &= R_{n-1} \end{aligned} \tag{I.4}$$

where $S(\cdot)$ is the s-box function. The existence of the initial and final permutation is also ignored as they do not contribute to the security of the algorithm.

The last round, as described above, can therefore be expressed in the following manner:

$$\begin{aligned} R_{16} &= S(R_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L_{16} \oplus K_{16}) \oplus L_{15} \end{aligned} \tag{I.5}$$

If a fault occurs during the execution of the fifteenth round, i.e. R_{15} is randomised by a fault to become R'_{15}, then:

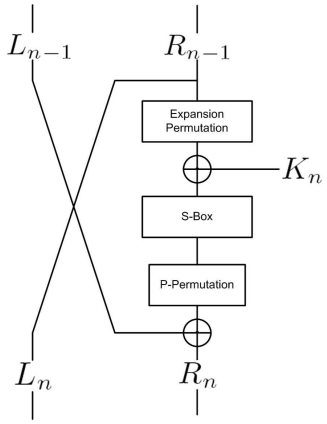

Figure I.5. The DES round function for round n.

$$\begin{aligned} R'_{16} &= S(R'_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(L'_{16} \oplus K_{16}) \oplus L_{15} \end{aligned} \tag{I.6}$$

If we xor R_{16} and R'_{16} we get:

$$\begin{aligned} R_{16} \oplus R'_{16} &= S(R_{15} \oplus K_{16}) \oplus L_{15} \oplus S(R'_{15} \oplus K_{16}) \oplus L_{15} \\ &= S(R_{15} \oplus K_{16}) \oplus S(R'_{15} \oplus K_{16}) \\ &= S(L_{16} \oplus K_{16}) \oplus S(L'_{16} \oplus K_{16}) \end{aligned} \tag{I.7}$$

This provides an equation where only the last subkey, K_{16}, is unknown. All of the other variables are visible in the ciphertext. This equation holds for each s-box in the last round, which means it is possible to search for key hypotheses in sets of six bits. All 64 possible key values corresponding to the xor just before each s-box are exhausted to generate a list of possible key values for these key bits. After this, all the possible combinations of the hypotheses can be searched though with the extra 8 key bits that are not included in the key to find the entire key.

If R'_{15} becomes random then the expected number of hypotheses that are generated can be predicted. For a given input and output difference there are certain number of values that could create the pair of differences, as described in [4]. The expected number of hypotheses for the last subkey will be around 2^{24}, giving an overall expected keyspace of 2^{32}.

If the attack is repeated to acquire two faulty ciphertexts the intersection of the two keyspaces can be taken. This will greatly reduce the keyspace that will need to be searched through to derive the key. It would be expected that two faulty ciphertexts with the properties described above would give around 2^6 hypotheses for the last subkey, leading to an exhaustive search of around 2^{14} for the entire key.

An implementation of this is described in [11] where two faulty ciphertexts are acquired, leading to a small exhaustive search to find the DES key used. Again, it would be expected that any of the fault models given would enable this attack to be conducted. As any fault during the calculation of R_{15} will give a ciphertexts with the correct properties.

This attack is also extended in [11] to allow for any faults from the eleventh round onwards, the details of which are beyond the scope of this article. Another attack [12] attempts an attack of a similar nature but using faults at the beginning of the DES algorithm.

This gives two different attacks against commonly used algorithms. There are a plethora of other attacks on other algorithms so they can not all be listed here. These two attacks were chosen to represent the fault attacks that have been proposed and implemented.

Countermeasures. There are various types of countermeasure that can be implemented to defend against fault attacks. These are usually based on existing techniques used for integrity purposes. It would normally be expected that anomaly sensors would be implemented in a smart card that would detect an attempted fault attack, but this cannot be relied upon as a new fault injection technique may be able to circumvent this.

A detailed list of the possible fault resistent hardware implementations is given in [3]. Some of the software countermeasures that can be used are listed below.

Checksums can be used to verify the integrity of data at all times. This is especially important when data is being moved from one memory area to another. For example, a fault injected in an RSA variable can compromise the secret key, as described above.

Variable redundancy is the reproduction of a variable in memory and functions are performed on each variable independently. The result will be known to be correct is both answers are identical.

Execution redundancy is the repetition of the same function, part of the function or it's inverse. In the case of the DES algorithm it would be prudent to repeat the first 3 rounds to protect against the attack described in [12], and the last 5 rounds to protect against the attack described in [5, 11]. This greatly increases the execution time but is less costly than repeating the entire algorithm.

In the case of RSA the simplest solution to protect a the signature generation function is to verify the result with the public key. The is efficient as signature verification is extremely fast when compared to signature generation. In some standards it is not always possible to have access to the public key, countermeasures have been proposed that verify certain conditions after signature generation. An example of this type of countermeasure, and an account of previous methods, is given in [9].

Execution Randomisation: If all the functions are conducted in a random order it is not be possible to determine exactly where a fault needs to be injected. The presents a similar problem to that described in Section I.3.1, as an attacker is unsure of what function is being attacked. However, this merely slows an attacker as an attack can be repeated until successful.

Ratification counters and baits: A countermeasure described in [3] involves including small functions in sensitive code that perform a small function and verify the result calculated. When an incorrect result is detected a fault in known to have been injected. The reaction to such an event would be to decrement a counter. When this counter reaches 0 the smart card would then cease to function. When combined with random delays described in Section I.3.1 this can be a very effective countermeasure.

In order to achieve a secure implementation the above countermeasures would need to be combined with those presented in Section I.3.1. This implies a significant overhead when implementing cryptographic algorithms for smart cards but is necessary to defend against modern attack techniques.

I.4 Application and Platform attacks

I.4.1 GSM and 3G Security

I.4.2 Java card and Platform Specific Attacks

I.5 Summary

Bibliography

[1] M.-L. Akkar and C. Giraud. An implementation of DES and AES secure against some attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptogaphic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pp. 309–318. Springer-Verlag, 2001

[2] C. Aumüller, P. Bier, P. Hofreiter, W. Fischer, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Cryptogaphic Hardware and Embedded Systems – CHES 2000*, volume 2523 of *Lecture Notes in Computer Science*, pages 260–275. Springer-Verlag, 2002

[3] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerers apprentice guide to fault attacks. *IEEE Special Issue on Cryptography and Security*, 94(2):370–382, 2006

[4] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In A. Menezes and S. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pp. 2–21. Springer-Verlag, 1991

[5] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B. S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pp. 513–525. Springer-Verlag, 1997

[6] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu and T. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults, the Proceedings of the *5th Workshop on Secure Protocols*, volume 1361 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 115–124, 1997

[7] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importantce of checking computations. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997

[8] S. Chari and C. S. Jutla and J. R. Rao and P. Rohatgi. Towards approaches to counteract power-analysis attacks. In M. Wiener, editor, *Advances in Cryptology*

– *CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 398–412, 1999

[9] M. Ciet and M. Joye. Practical fault countermeasures for chinese remaindering based RSA. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography 2005 – FDTC 2005*, pp. 124–131, 2005

[10] C. Clavier, J.-S. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pp. 252–263. Springer-Verlag, 2000

[11] C. Giraud and H. Thiebeauld. A survey on fault attacks. In Y. Deswarte and A. A. El Kalam, editors, *Smart Card Research and Advanced Applications VI – 18th IFIP World Computer Congress*, pp. 159–176. Kluwer Academic, 2004

[12] L. Hemme. A differential fault attack against early rounds of (triple-)DES. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pp. 254–267. Springer-Verlag, 2004.

[13] J. Blömer and J.-P. Seifert. Fault based cryptanalysis of the advanced encryption standard (AES). In R. N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pp. 162–181. Springer-Verlag, 2003

[14] Global Platfom, available from `http://www.globalplatform.org`

[15] M. Joye and F. Olivier. Side-channel attacks. In H. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, pp. 571–576. Kluwer Academic Publishers, 2005

[16] M. Joye, J.-J. Quisquater, F. Bao, and R.H. Deng, RSA-type signatures in the presence of transient faults, In M. Darnell, editor, *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pp. 155-160, Springer-Verlag, 1997

[17] P. Kocher. Timing attacks on implementations of diffe-hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pp. 104–113. Springer-Verlag, 1996.

[18] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pp. 388–397. Springer-Verlag, 1999

[19] D.H Habing. The Use of Lasers to Simulate Radiation-Induced Transients in Semiconductor Devices and Circuits, In *IEEE Transactions On Nuclear Science*, volume 39, pp. 1647–1653, 1992

[20] D. Naccache, P. Q. Nguyễn, M. Tunstall, and C. Whelan. Experimenting with faults, lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography – PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 16–28. Springer-Verlag, 2005

[21] D. Samyde, S. P. Skorobogatov, R. J. Anderson, and J.-J. Quisquater. On a new way to read data from memory. In *Proceedings of the First International IEEE Security in Storage Workshop*, pp. 65–69, 2002

[22] Season 2 Interface, available from `http://www.maxking.co.uk/season2.htm`

[23] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. Kaliski Jr. and Ç. K. Koç and C. Paar, editors, *Cryptogaphic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pp. 2–12. Springer-Verlag, 2002.

[24] J. Ziegler. Effect of Cosmic Rays on Computer Memories, *Science*, volume 206, pp. 776–788, 1979