Chapter 9
Smart Card Security

Michael Tunstall

Abstract In this chapter, the various attacks and countermeasures that apply to secure smart card applications are described. This chapter focuses on the attacks that could affect cryptographic algorithms, since the security of many applications is dependent on the security of these algorithms. Nevertheless, how these attacks may be applied to other security mechanisms is also described. The aim of this chapter is to demonstrate that a careful evaluation of embedded software is required to produce a secure smart card application.

Keywords Embedded software · Fault analysis · Side channel analysis · Smart card security

9.1 Introduction

The implementation of secure applications on smart cards is different to the development on other platforms. Smart cards have limited computing power, comparatively small amounts of memory and are reliant on a smart card reader to provide power and a clock. There are security considerations that are specific to smart cards, that need to be taken into account when developing a secure smart card-based application.

In this chapter, attacks that are specific to smart cards, and other devices based around a secure microprocessor, will be described. There are other considerations that need to be taken into account when implementing a secure application, but these are generic and beyond the scope of this chapter.

There are three main types of attack that are considered in smart card security. These are:

1. **Invasive Attacks**: These are attacks that require the microprocessor in a smart card to be removed and directly attacked through a physical means. This class of attacks can, at least in theory, compromise the security of any secure microprocessor. However, these attacks typically require very expensive equipment and

M. Tunstall (✉)
Cryptography Research, 425 Market Street, San Francisco, CA 94105, USA
e-mail: michael.tunstall@cryptography.com

K. Mayes and K. Markantonakis (eds.), *Smart Cards, Tokens, Security and Applications*, DOI 10.1007/978-3-319-50500-8_9

a large investment in time to produce results. Invasive attacks are therefore considered to be primarily in the realm of semiconductor manufacturers and students at well-funded universities.

An example of such an attack would be to place probes on bus lines between blocks of a chip (a hole needs to be made in the chip's passivation layer to allow this). An attacker could then attempt to derive secret information by observing the information that is sent from one block to another.

At its most extreme this type of attack could make use of a focused ion beam to destroy or create tracks on the chips surface. In theory, this could, for example, be used to reconnect fuses. Traditionally, chip manufacturers typically used a test mode where it was possible to read and write to all memory addresses whilst a fuse was present. Once the fuse was blown inside the chip (before the chip left the manufacturer's factory) this mode was no longer available. In modern secure microprocessors this test circuit is typically removed when the chip is cut from the die preventing the attack.

Further information on invasive attacks is available in [3, 32]. More recently, Tarnovsky [52] has made videos on how these attacks are conducted publicly available, and they should be easy to find.

2. **Semi-Invasive Attacks**: These attacks require the surface of the chip to be exposed. An attacker then seeks to compromise the security of the secure microprocessor without directly modifying the chip.

   Some examples of this type of attack include injecting faults using laser or white light [6, 50]. More details on these attacks are given in later sections, and further information can be found in [27].

   The first attempts at analysing the electromagnetic emanations around a microprocessor required a suitable probe to be placed very close to the surface of a targeted microprocessor [17, 45]. This has since been shown to be unnecessary and such analyses can be classed as non-invasive.

3. **Non-Invasive Attacks**: These attacks seek to derive information without modifying a smart card, i.e. both the secure microprocessor and the plastic card remain unaffected. An attacker will attempt to derive information by observing information that leaks during the computation of a given command, or attempts to inject faults using mechanisms other than light.

   Some examples of this type of attack would be to observe the power consumption of a microprocessor [31], or to inject faults by putting a glitch into the power supply [3]. Further descriptions of power analysis attacks can be found in [34], and fault attacks in [27].

This chapter will focus on semi-invasive and non-invasive attacks, as the equipment required to conduct these attacks is more readily available. Invasive attacks are of interest, but are extremely expensive to conduct. This chapter will focus more on what can be achieved in a reasonably funded laboratory. However, some information is given on invasive attacks where relevant.

9.1.1 Organisation

Section 9.2 contains a description of the cryptographic algorithms that will be used in later sections to give examples of attacks. Section 9.3 describes certain hardware security features that are typically included in a smart card. Section 9.4 describes the different forms of side channel analysis and how they can be applied to smart card implementations of cryptographic algorithms. Section 9.5 describes how fault attacks can be applied to smart cards. Section 9.6 describes how the techniques given in Sects. 9.4 and 9.5 can be applied to other security mechanisms. Section 9.7 summarises the chapter.

9.1.2 Notation

The base of a value is determined by a trailing subscript, which is applied to the whole word preceding the subscript. For example, $\texttt{FE}_{16}$ is 254 expressed in base 16 and $d = (d_{\ell-1}, d_{\ell-2}, \ldots, d_0)_2$ gives a binary expression for the ℓ-bit integer d.

In all the algorithms described in this chapter, ϕ represents Euler's totient function, where $\phi(N)$ equals the number of positive integers less than N which are coprime to N. In particular, if $N = pq$ is an RSA modulus then $\phi(N) = (p-1)(q-1)$.

9.2 Cryptographic Algorithms

Some of the attacks detailed in later sections will assume a detailed knowledge of some of the commonly used cryptographic algorithms. The Data Encryption Standard (DES) and RSA, are detailed in this section to provide a reference, and to describe the notation that will be used. For brevity, we do not describe the Advanced Encryption Standard (AES) [43], but mention of how the AES affects security will be made where required.

9.2.1 Data Encryption Standard

The Data Encryption Standard (DES) was introduced by NIST in the mid 1970s [42], and was the first openly available cryptography standard. It became, and still is, a worldwide *de facto* standard for numerous purposes. However, it has been practically demonstrated that an exhaustive search is possible, leading to the introduction of triple DES (see below).

DES can be considered as a transformation of two 32-bit variables (L_0, R_0), i.e. the message block, through sixteen iterations of a round function, as shown in Fig. 9.1, to produce a ciphertext block (L_{16}, R_{16}). The Expansion permutation

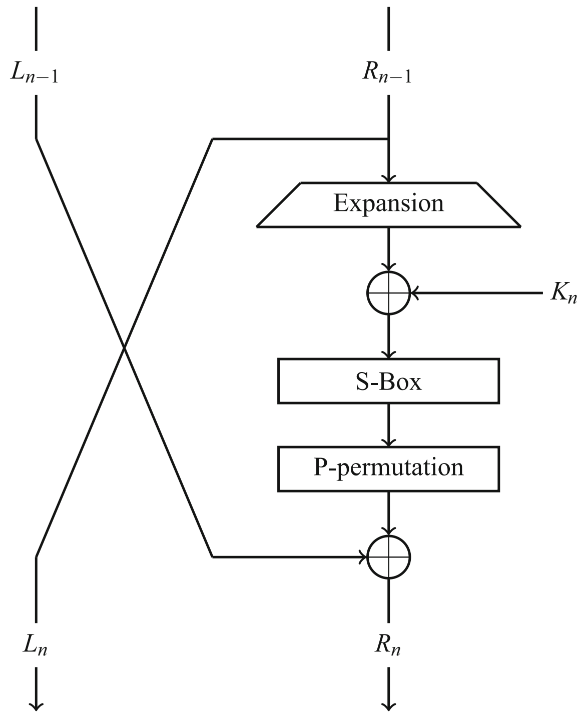


Fig. 9.1 The DES round function for round *n*

selects eight overlapping six-bit substrings from R_n. The P-permutation is a bitwise permutation on the 32-bit output of the S-box function. For clarity of expression, these permutations will not always be considered and the round function will be written as:

$$\begin{aligned} R_n &= S(R_{n-1} \oplus K_n) \oplus L_{n-1} \\ L_n &= R_{n-1} \end{aligned} \tag{9.1}$$

where S is the S-box function. The subkeys K_n, for $1 \leq n \leq 16$, are each 48 bits generated from a 56-bit secret key, by choosing 48 bits from the 56-bit key. This is done by initially conducting a bitwise permutation on the key, referred to as Permuted Choice 1 (PC1). Each round bit shifts are conducted on the key, and 48 bits are chosen from this shifted key to form each subkey using the Permuted Choice 2 (PC2) function.

Eight different S-boxes are applied in each round to sets of six bits, thereby reducing the 48-bit output of the XOR with K_n to 32 bits. Each S-box is a substitution table that is addressed using six bits of information, and each entry is a 4-bit number.

The algorithm also includes an initial and final permutation (these permutations are referred to as IP and IP^{-1} respectively), where the final permutation is the inverse of the initial permutation. More precisely the permutation at the end of DES is conducted on (R_{16}, L_{16}) rather than (L_{16}, R_{16}). These permutations will be ignored in this chapter, as they do not contribute to the security of the algorithm. The permutations IP and IP^{-1} were included because it was the most convenient way of introducing the bits into the first chips used to calculate the DES algorithm (at the time, software implementations were not considered because of the complexity of the algorithm) [38].

9.2.1.1 Triple DES

In order to mitigate the aforementioned key length problem, a modification to DES was proposed to make an exhaustive key search prohibitively complex. Triple DES is a construction that uses two different DES keys and is defined in [42]. In the algorithm below the two DES keys are denoted K_1 and K_2, and in order to generate a ciphertext block C from a plaintext block M the following calculation is performed:

$$C = \mathrm{DES}\,(\mathrm{DES}^{-1}\,(\mathrm{DES}\,(M,\ K_1),\ K_2),\ K_1) \tag{9.2}$$

where DES(M, K) denotes the output of the DES encryption algorithm applied to message block M with key K. Deciphering the ciphertext block C uses the function,

$$M = \mathrm{DES}^{-1}\,(\mathrm{DES}\,(\mathrm{DES}^{-1}(C,\ K_1),\ K_2),\ K_1) \tag{9.3}$$

The structure of triple DES allows for backward compatibility as if K_1 and K_2 are equal the resulting ciphertext will the equivalent to that produced with a single DES. The triple DES requires that three instantiations of the DES algorithm are used, since it has been shown that two instantiations of DES only increase the security of the algorithm by one bit (see meet-in-the-middle attacks [36]).

Another version of triple DES is also proposed in [42], in which three different keys are used rather than two.

9.2.2 RSA

RSA was first introduced in 1978 [47], and was the first published example of a public key cryptographic algorithm. The security of RSA depends on the difficulty of factorising large numbers, meaning that RSA keys need to be quite large. Advances in factorisation algorithms and the constantly increasing processing power available in modern computers has led to constantly increasing key sizes. At the time of writing RSA is typically used with 1024 or 2048-bit key sizes.

To generate a key pair for use with RSA, two prime numbers, p and q, of a similar bit length, are generated; they are then multiplied together to create a value N, the modulus, whose bit length is equal to that desired for the cryptosystem. That is, in order to create a 1024-bit modulus, $2^{511.5} < p, q < 2^{512}$ [1] (if values of p or q are chosen from the interval $(2^{511}, 2^{511.5})$ the product of p and q is not guaranteed to have a bit length of 1024 bits). A public exponent, e, is chosen that is coprime to both $p - 1$ and $q - 1$.

A private exponent, d, is generated from the parameters previously calculated, using the formula:

$$\begin{aligned} e\,d &\equiv 1 \pmod{(p-1)(q-1)}, \text{ or equivalently} \\ e\,d &\equiv 1 \pmod{\phi(N)} \end{aligned} \tag{9.4}$$

where ϕ is Euler's Totient function.

9.2.2.1 The RSA Cryptosystem

In the RSA cryptosystem, to encrypt a message, M, and create ciphertext, C, one calculates:

$$C = M^e \mod N \tag{9.5}$$

The value of e is often chosen as 3 or $2^{16} + 1$, as these values are small, relative to N, and have a low Hamming weight, which means that the encryption process is fast (see below). To decrypt the ciphertext, the same calculation is carried out but using the private exponent, d, which generally has a similar bit length as N:

$$M = C^d \mod N \tag{9.6}$$

9.2.2.2 The RSA Signature Scheme

The RSA digital signature scheme involves the reverse of the operations used in the RSA cryptosystem. The generation of a signature, S, uses the private exponent d. By convention, this is expressed as:

$$S = M^d \mod N \tag{9.7}$$

The verification therefore uses the public exponent and is expressed as:

$$M = S^e \mod N \tag{9.8}$$

[1]This is possibly overly strong, as it typically recommend that the bit lengths of p and q are approximately equal. However, it will provide the most security for a modulus of a given bit length assuming that $p - q$ is sufficiently large to prevent an attacker from guessing their values by calculating $\sqrt{N}$.

9.2.2.3 Padding Schemes

Applying the RSA primitive to a message, as described above, will not yield a secure signature or encryption scheme (for reasons beyond the scope of this chapter). To achieve a secure scheme it is necessary to apply the RSA operation to a transformed version of the message, e.g. as can be achieved by hashing the message, adding padding, and/or masking the result. This process is termed padding, and the interested reader is referred to [36] for a treatment of padding schemes.

Some of the attacks presented in this chapter will not be realistic when a padding scheme is used, since padding schemes mean that an attacker cannot entirely control a message. However, it is important that an implementation of RSA is secure against all possible attacks. If a given implementation does not use padding or, more realistically, contains a bug that allows an attacker to remove the padding function, the implementation should still be able to resist all the attacks described in this chapter.

9.2.2.4 Computing a Modular Exponentiation

Many different algorithms can be used to calculate the modular exponentiation algorithm required for RSA. In practice, a large number of algorithms cannot be implemented on smart cards, as the amount of available memory does not usually allow numerous intermediate values to be stored in RAM. The manipulation of large numbers is typically performed using a coprocessor (see Sect. 9.3), as implementing a multiplication on an 8-bit platform would not give a desirable performance level.

The simplest exponentiation algorithm is the square and multiply algorithm [36], and is given in Algorithm 1 for an exponent d of bit length ℓ:

Algorithm 1: The Square and Multiply Algorithm

```
Input: $M, d = (d_{\ell-1}, d_{\ell-2}, \ldots, d_0)_2, N$
Output: $S = M^d \bmod N$

$A \leftarrow 1$ ;
for $i = \ell - 1$ to 0 do
    $A \leftarrow A^2 \bmod N$ ;
    if $(d_i = 1)$ then
        $A := A \cdot M \bmod N$ ;
    end
end
return $A$
```

The square and multiply algorithm calculates $M^d \bmod N$ by loading the value one into the accumulator A and d is read bit-by-bit. For each bit a squaring operation modulo N takes place on A, and when a bit is equal to one A is subsequently multiplied by M. It is because of this multiplication that e is typically chosen as 3 or $2^{16} + 1$, as both values only have two bits set to one; therefore minimising the number of multiplications required. It is not possible to only have one bit set to one as it is necessary for e to be an odd number in order for it to have an inverse modulo N. The most significant bit of a number will always be set to one, and the least significant bit will need to be set to one to produce an odd number.

9.2.2.5 Using the Chinese Remainder Theorem

The RSA calculation using the private exponent (i.e. where $S = M^d \bmod N$ and $N = p \cdot q$) can be performed using the Chinese Remainder Theorem (CRT) [29]. Initially, the following values are calculated,

$$\begin{aligned} S_p &= M^{d \bmod (p-1)} \bmod p \\ S_q &= M^{d \bmod (q-1)} \bmod q \end{aligned} \tag{9.9}$$

which can be combined to form the RSA signature S using the formula $S = a\, S_p + b\, S_q \bmod N$, where:

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

This can be implemented in the following manner:

$$S = S_q + \left(\left(S_p - S_q\right) q^{-1} \bmod p\right) \cdot q \tag{9.10}$$

This provides a method of calculating an RSA signature that is approximately four times quicker than a generic modular exponentiation algorithm, i.e. two exponentiations, each of which can be computed eight times faster than an exponentiation using d (the bit length of $d \bmod (p-1)$ and $d \bmod (p-1)$ will be half that of d). This advantage is offset by an increase in the key information that needs to be stored. Rather than storing just the value of d and N, the values of $(p, q, d \bmod (p-1), d \bmod (q-1), q^{-1} \bmod p)$ need to be precalculated and stored.

9.3 Smart Card Security Features

This section will detail some of the features of smart cards that are pertinent when considering their security. Smart cards have traditionally been based on 8-bit Complex Instruction Set Computer (CISC) architectures [40]. Usually built around a Motorola 6805 or Intel 8051 core, often with extensions to the instruction set. More sophisticated smart cards are emerging based on 32-bit Reduced Instruction Set Computer (RISC) architecture chips, containing dedicated peripherals (cryptographic coprocessors, memory managers, large memories, ...) [39].

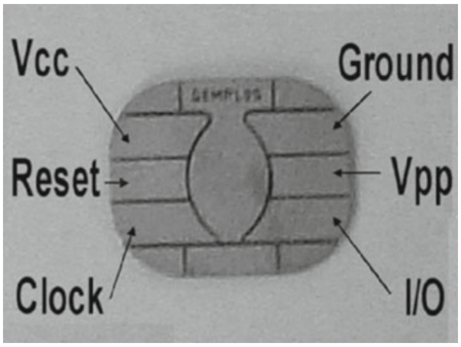


Fig. 9.2 The Contacts used to power and communicate with a smart card

9.3.1 Communication

A smart card has five contacts that it uses to communicate with the outside world defined in the ISO/IEC 7816-2 standard [25]. Two of these are taken by the power supply (usually 3 or 5 v), referred to as Vcc, and the ground used to power the chip. Another contact is used to supply a clock, which is allowed to vary between 1 and 5 MHz but is typically set to 3.57 MHz. The remaining two contacts are used to communicate with the microprocessor. A sixth contact was originally used to provide a higher voltage to program the EEPROM (referred to as Vpp), but is no longer in use for reasons described in Sect. 9.6. The location of the different contacts is shown in Fig. 9.2.

One of the contacts, referred to as the I/O, is used for communication and to send commands to the chip in a smart card. The protocols used to communicate with a smart card are referred to as T = 0 and T = 1 and are defined in the ISO/IEC 7816-3 standard [24]. This section will describe both protocols, as they are nearly identical. The extra requirements of T = 1 are detailed where relevant.

The remaining contact is used to reset the smart card (there are a further 2 contacts defined in the ISO/IEC 7816-3 standard but they are not currently used). Resetting a smart card is a physical event (i.e. moving the voltage applied to this contact from 0 to 1) and it will always provoke a response from a smart card. A user can apply the reset at any time. The smart card will respond by sending an Answer To Reset (ATR) via the I/O contact, which is a string of bytes that defines the protocols the smart card can use, the speeds at which the smart card can communicate and the order in which bits are going to be sent during the session (i.e. most or least significant bit first).

9.3.2 Cryptographic Coprocessors

Traditionally, smart cards have been based around 8-bit architectures. In order to manipulate large numbers, e.g. to calculate the RSA algorithm, dedicated coprocessors can be appended to the CPU. In more modern 32-bit chips [39] this is not always necessary, as efficient software implementations can be achieved. However, coprocessors are still often used for increased performance. DES and AES is also often implemented in a coprocessor to help increase performance, where hardware implementations of these secret key algorithms can typically be expected to require one or two clock cycles per round of the block cipher. The inclusion of coprocessors increases the size of a microprocessor and the overall power consumption. This means that chips with coprocessors are usually more expensive and are not always ideal in environments where the amount of available current is limited.

9.3.3 Random Number Generators

Random number generators are usually included in smart cards, as unpredictable numbers are an important element in many secure protocols. A "true" random number generator is typically based on a signal generated by an analogue device which is then treated to remove any bias that may exist, or has been induced, in the bits generated. The correct functioning of all aspects of a smart card chip under varied environmental conditions is important, but is critical for random number generation because the quality of the generated random values can have a profound effect on cryptographic schemes. Random number generators are therefore designed to function correctly in a large range of environmental conditions, including temperature, supply voltage, and so on. However, if an attacker succeeds in modifying the environmental conditions such that the physical source of randomness is affected, the output is typically treated such that an attacker will not be able to determine if the change in conditions had any effect.

Pseudo-random number generators are also often included in a secure microprocessor. These are typically based on Linear Feedback Shift Registers (LFSRs) that are able to generate a new pseudo-random value every clock cycle, but are deterministic over time and are not usually used for critical security functions.

Where random values are required in cryptographic algorithms, a true random number generator is used when the quality of the random value is important, e.g. for use in a cryptographic protocol. Where the quality of the random value is less important, a pseudo-random number generator can be used. In some secure microprocessors only pseudo-random number generators are available. In this case, mechanisms that combine a random seed (that can be inserted into the chip during manufacture) with pseudo-random values can be used to provide random values.

An example of this latter type of random number generator is given in the ANSI X9.17 [2, 22] standard, that uses DES to provide random values based on a random seed generated during the initialisation of a given device and another source of pseudo-random information. This functions by taking a 64-bit pseudo-random input (X), a 64-bit random seed (S) and a DES key (K). X is usually generated by calculating $X = \mathrm{DES}(D, K)$, where D is a the date and/or time, but this information is not available to a smart card and is typically replaced with values provided by a pseudo-random number generator. To output a random value R the following calculation takes place:

$$R = \mathrm{DES}(X \oplus S, K), \tag{9.11}$$

and the random seed is updated using:

$$S = \mathrm{DES}(R \oplus X, K). \tag{9.12}$$

For increased security the DES function can be replaced with triple DES, as the key length used by DES has proven to be too short to entirely resist an exhaustive key search.

9.3.4 Anomaly Sensors

There are usually a number of different types of anomaly detectors present in smart cards. These are used to detect unusual events in the voltage and clock supplied to the card, and the environmental conditions (e.g. the temperature). These enable a smart card to detect when it is exposed to conditions that are outside the parameters within which it is known to function correctly. When unusual conditions are detected, the chip will cease to function until the effect has been removed (i.e. initiate a reset or execute an infinite loop when the sensor is activated). However, it is considered prudent not to rely solely on these sensors and to implement further countermeasures (see Sect. 9.5).

9.3.5 Chip Features

The surface of the chip used in a smart card can be uncovered by removing the plastic body of the card and using fuming nitric acid to remove the resin used to protect the microprocessor. Once the chip has been revealed the easiest form of analysis is to simply look at it under a microscope. The various different blocks can often be identified, as shown in Fig. 9.3.

Reverse engineering can target the internal design to understand how a given chip or block functions. An attacker can use such information to improve their knowledge of chip design and find potential weaknesses in the chip, which may allow them to compromise the chip's integrity.

In modern smart cards, various features used to inhibit reverse engineering are implemented using glue logic: important blocks are laid out in a randomised fashion that makes reverse engineering difficult. This technique increases the size of the block, and is therefore not used in the design of large blocks such as Read-Only Memory (ROM) and Electrically Erasable Programmable Read-Only Memory (EEPROM).

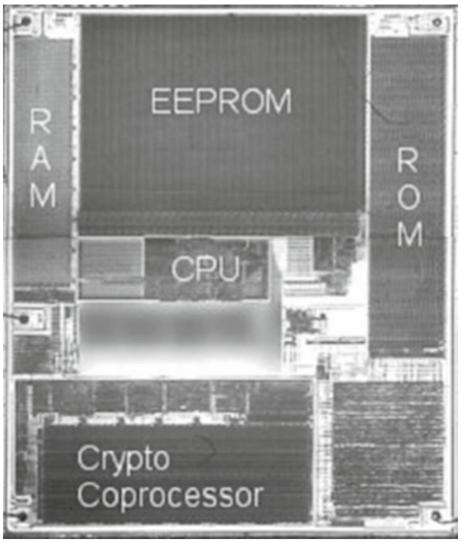


Fig. 9.3 A chip surface with readily identifiable features

Another common technique to prevent this sort of identification and targeting is to overlay the chip with another metal layer that prevents the chip's features being identified. This can be removed using hydrofluoric acid that eats through the metal layer; this reaction is then stopped using acetone before further damage is done and the chip surface can be analysed. The chip becomes non-functional but the layout of the chip can be determined, so that other chips of the same family can be attacked. The result of such a process is shown in Fig. 9.4.

Discovering the layout and functioning of a chip is particularly important when using a laser as a fault injection mechanism (see Sect. 9.5). Different areas of a chip can be targeted through the metal layer once the layout of a chip is known. Tarnovsky [52] has made videos on how these attacks are conducted publicly available and they should be easy to find.

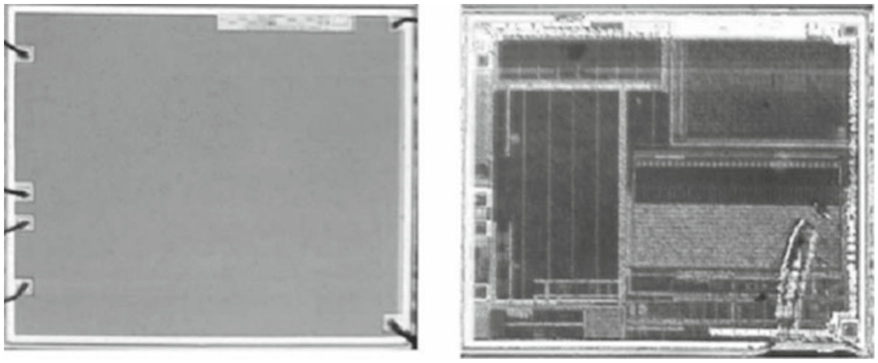


Fig. 9.4 A chip with a shield present and removed

9.4 Side Channel Analysis

Side-channel attacks are a class of attacks, where an attacker will attempt to deduce what is occurring inside a device by observing information that leaks during the normal functioning of the device. If this information can be related to any secret information being manipulated the security of the device can be compromised. It should be noted that side channel analysis is a passive form of attack, i.e. an attacker will simply observe what is occurring when a device is functioning. In the case of smart cards the message being manipulated can be controlled, but this is not necessary to construct a valid side channel attack.

The first publication that mentions a side-channel attack is described in [53]. In 1956, MI5 mounted an operation to decipher communications between Egyptian embassies. The communications were enciphered using Hagelin machines [28]. These machines did not function using a key value as described in Sect. 9.2. Enciphering occurred by routing electronic signals from a keyboard through seven rotating wheels to generate a ciphertext. The "key" was the initial setting of these seven wheels. The machine was reset every morning by the clerk who would be sending messages. MI5 managed to plant a microphone in close proximity to one of these machines. This allowed the initial settings to be determined by listening to the initial settings being made every morning. This would have allowed them to decipher intercepted communications with another Hagelin machine set to the same key. In practice, MI5 was only able to determine a certain amount of wheel settings because of the difficulty of distinguishing the noise of the wheels being set from background noise. This made deciphering more complex, but not impossible, as the number of possible keys was significantly reduced by the partial information.

9.4.1 Timing Analysis

The first modern example of a side channel attack was proposed in [30]. This involved observing the differences in the amount of time required to calculate an RSA signature for different messages to derive the secret key. This attack was conducted against a PC implementation, but a similar analysis could potentially 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 a trace acquired with an oscilloscope that would provide this sort of information is shown in Fig. 9.5. The I/O events on the left-hand side of the figure represent the reader sending a command to the smart card. The I/O events on the right hand side of the figure show the response of the smart card. The time taken by a given command can be determined by observing the amount of time that passes between these two sets of events.

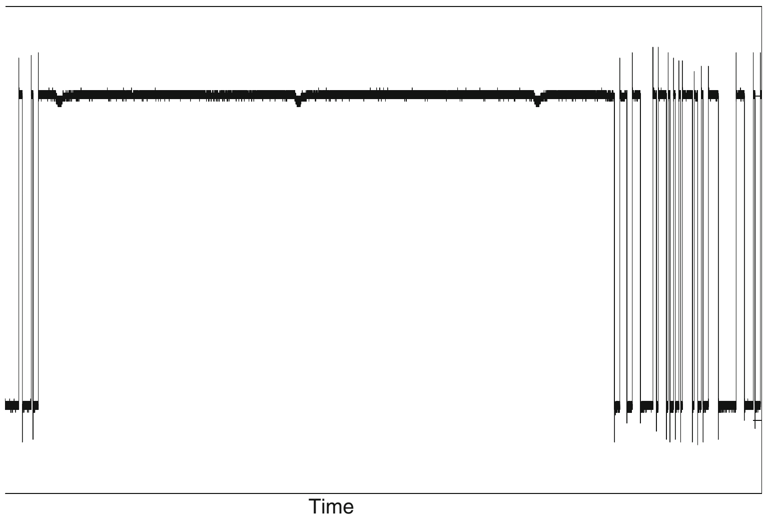


Fig. 9.5 The I/O of a smart card command

9.4.2 Power Analysis

The most common form of side-channel attack, when considering smart cards, is the analysis of the instantaneous power consumption [31]. This is measured by placing a resistor in series with a smart card and the power supply (or ground), and measuring the potential difference across the resistor with an oscilloscope. The acquired information can be analysed *a posteriori* to attempt to determine information on what is occurring within a secure microprocessor. There are two main types of power attack; these are Simple Power Analysis (SPA) and Differential Power Analysis (DPA).

9.4.2.1 Simple Power Analysis

A powerful form of power analysis is to search for patterns within an acquired power consumption trace. An attacker can attempt to determine the location of individual functions within a command. For example, Fig. 9.6 shows the power consumption of a smart card during the execution of DES. A pattern can be seen that repeats 16 times, corresponding to the 16 rounds that are required during the computation of DES.

This analysis can be further extended by closely inspecting the power consumption during the computation of one round, to attempt to determine the individual functions within each round. This is shown in Fig. 9.7, where the functions in the second round of a DES implementation are evident in the power consumption trace. This may not

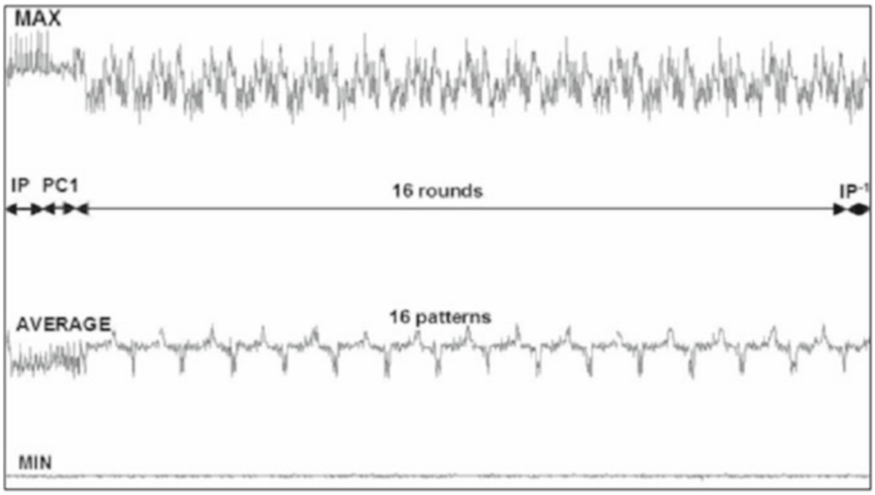
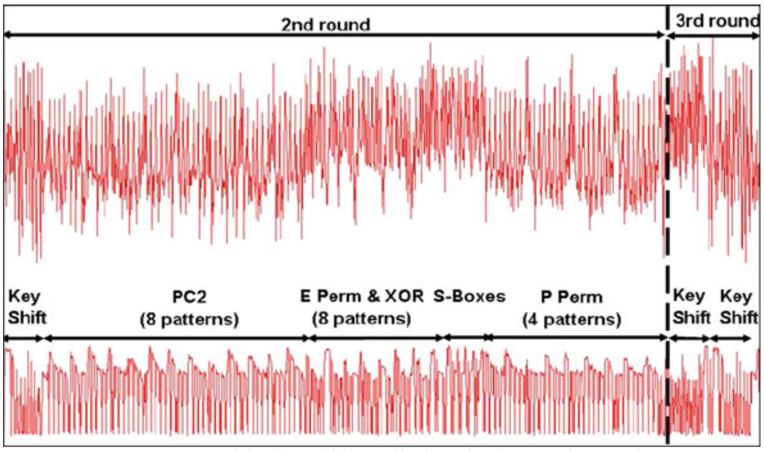


Fig. 9.6 The power consumption of a DES implementation showing the rounds of the algorithm



Key:	**Key Shift**	A bitwise shift applied to the key each round.
	PC2	PC2 used to generate a 48-bit round key each round.
	E Perm	Expansion permutation applied to R_i, for $1 \leq i \leq 16$, to produce a 48-bit output.
	XOR	The XOR with the round key.
	S-boxes	Eight substitution tables reducing 48 bits to 32 bits.
	P Perm	The P permutation, a bitwise transformation.

Fig. 9.7 The power consumption of a DES implementation showing the round functions

be immediately apparent, as close inspection of the trace's features is necessary to identify the individual functions. For example, if an attacker is seeking to determine where the compression permutation (PC2) is computed, they will look for eight patterns of six events. This is because the compression permutation selects 48 bits from the 56-bit DES key, where the 48-bit result is divided into segments of 6 bits (for use in the S-box function). The natural method of implementing this permutation will, therefore, be to construct a loop that will repeat eight times. Each loop will move 6 bits from the DES key to the 48-bit output. This should therefore produce eight patterns of six events because of the individual bits being selected and written.

The use of this information is not necessarily immediately apparent; an attacker can use this information to improve the effectiveness of other attacks. The efficiency of the statistical treatment required for Differential Power Analysis (DPA) [31] can be increased by taking more precise acquisitions. This is because the area that needs to be analysed can be defined, and therefore reducing the amount of data that needs to be acquired. A more detailed analysis is given below.

The same is true for fault injection techniques, detailed in Sect. 9.5, as it is often necessary to target specific events. If arbitrary functions can be identified using the power consumption, the point in time at which an attacker wishes to inject a fault can be discovered. This can greatly decrease the time required to conduct a successful attack, as less time is wasted injecting faults into areas of the computation that will not produce the desired result.

The examination of the power consumption can also be used to determine information on the private/secret keys used in a naïve implementations of cryptographic algorithms. For example, if the power consumption of a smart card during the generation of an RSA signature using the square and multiply algorithm is analysed, it may be possible to determine some bits of the private key. An example of the power consumption during the generation of an RSA signature is shown in Fig. 9.8.

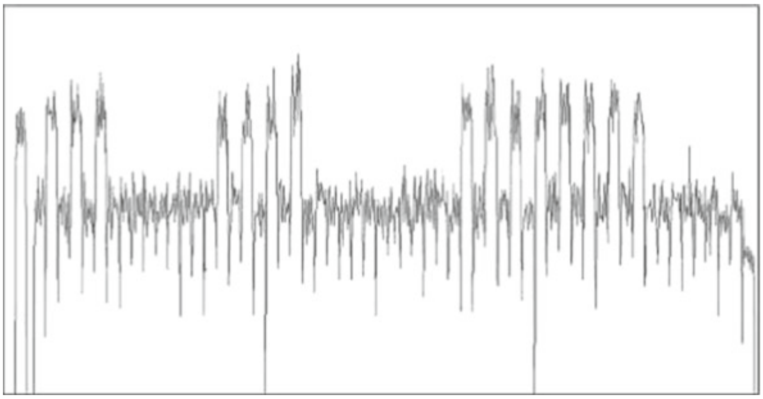


Fig. 9.8 The power consumption of an RSA implemented using the square and multiply algorithm

Looking closely at the acquired power consumption, a series of events can be seen. There are two types of events at two different power consumption levels, with a short dip in the power consumption between each event. This corresponds well to the square and multiply algorithm described in Sect. 9.2. Given the ratio of the two features, it can be assumed that the feature with the lower power consumption represents the squaring operation and the higher power consumption represents the multiplication. From this, the beginning of the exponent can be read from the power consumption, in this case the exponent used is $\texttt{F00F000FF00}_{16}$.

It should be noted that all the examples given in this section have been taken from chips that display the differences in an obvious manner. Modern secure microprocessors rarely display the functions being executed as clearly as in the examples given.

9.4.2.2 Differential Power Analysis

The idea of statistically treating power analysis traces was first presented to the cryptographic community in [31], and is referred to as Differential Power Analysis (DPA). DPA is based on the relationship between the power consumption and the data being manipulated at a given point in time. The differences in power consumption are potentially extremely small, and cannot be interpreted individually, as the information will be lost in the noise incurred during the acquisition process. The small differences produced can be seen in Fig. 9.9, where traces were taken using a chip where the acquisition noise is exceptionally low. Different power levels, corresponding to different Hamming weights of the data being manipulated, are clearly visible.

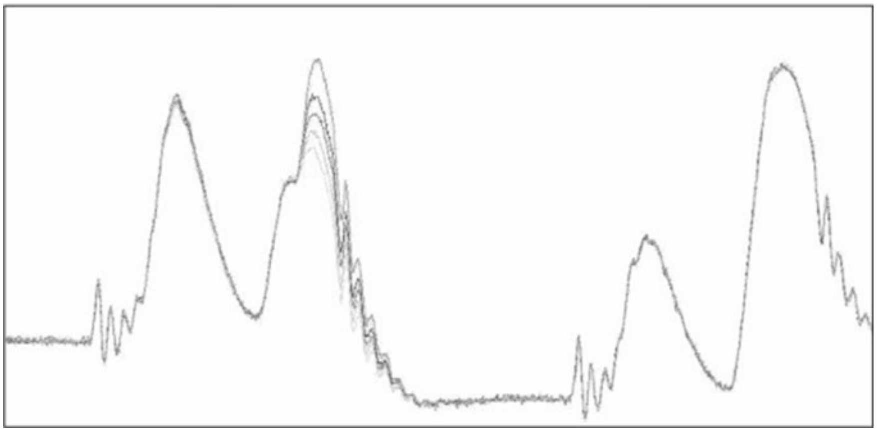


Fig. 9.9 Overlaid acquisitions of the power consumption produced by the same instruction but with varying data

Differential Power Analysis (DPA) can be performed on any algorithm in which an intermediate operation of the form $\beta = S(\alpha \oplus K)$ is calculated, where α is known and K is the key (or some segment of the key). The function S is typically a nonlinear function, usually a substitution table (referred to as an S-box), which produces an intermediate output value β.

The process of performing the attack initially involves running a microprocessor N times with N distinct message values M_i, where $1 \leq i \leq N$. The encryption of the message M_i under the key K to produce the corresponding ciphertext C_i will result in power consumption traces w_i, for $1 \leq i \leq N$. These traces can be captured with an oscilloscope, and sent to a computer for analysis and processing.

To find K, one bit of β is chosen, which we will refer to as b. For a given hypothesis for K, this bit will classify whether each trace w_i is a member of one of two possible sets. The first set S_0 will contain all the traces where b is equal to zero, and the second set S_1 will contain all the remaining traces, i.e. where the output bit b is equal to one.

A differential trace Δ_n is calculated by finding the average of each set and then subtracting the resulting values from each other, where all operations on traces are conducted in a pointwise fashion, i.e. this calculation is conducted on the first point of each acquisition to produce the first point of the differential trace, the second point of each acquisition to produce the second point of the differential trace, etc.

$$\Delta_n = \frac{\sum_{w_i \in S_0} w_i}{|S_0|} - \frac{\sum_{w_i \in S_1} w_i}{|S_1|}$$

A differential trace is produced for each value that K can take. In DES the first subkey will be treated in groups of six bits, so 64 (i.e. 2^6) differential traces will be generated to test all the combinations of six bits. The differential trace with the highest peak will validate a hypothesis for K, i.e. $K = n$ corresponds to the Δ_n featuring a maximum amplitude. An example of a differential trace produced by predicting one bit of the output a DES S-box, with a correct key guess, is shown in Fig. 9.10.

The differential trace in Fig. 9.10 shows a large difference in the power consumption at five different points, which are referred to as DPA peaks. The first peak

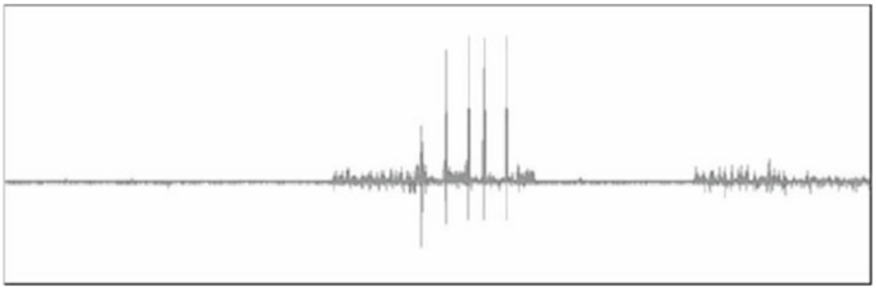


Fig. 9.10 A differential trace

corresponds to the output of the S-box, i.e. where the output of the S-box function is determined and written to memory. The four subsequent peaks correspond to the same bit being manipulated in the P-permutation. This occurs because the output of each S-box consists of four bits, and the memory containing those four bits will be accessed each time one of those bits is required in the output of the P-permutation.

A more complete version of this attack uses Pearson's correlation coefficient to demonstrate the correlation between the Hamming weight and the instantaneous power consumption. This can be used to validate key hypotheses in an identical manner to DPA. Details of this method are beyond the scope of this chapter, but the interested reader is referred to [11].

9.4.3 Electromagnetic Analysis

An alternative side channel to measuring the power consumption of a smart card is to measure the instantaneous electromagnetic emanations as a cryptographic algorithm is being computed [17, 45]. This is typically implemented using a small probe, an example of which can be seen in Fig. 9.11. Such probes can measure the electromagnetic emanations for different blocks of a chip, as such probes are an equivalent size to the chip's features. This means that the probe can be placed just above a given feature to try and get a strong signal from that part of the chip, e.g. the bus between two areas of the chip, while excluding noise from other areas of the chip.

Measuring the electromagnetic field can be done using a handmade probe (such as that shown in Fig. 9.11), although commercially available probes are also sufficient (and has the advantage that the frequency response is defined). The signals from a probe is passed through an amplifier and can be acquired using an oscilloscope in the same way one would acquire a power consumption trace.

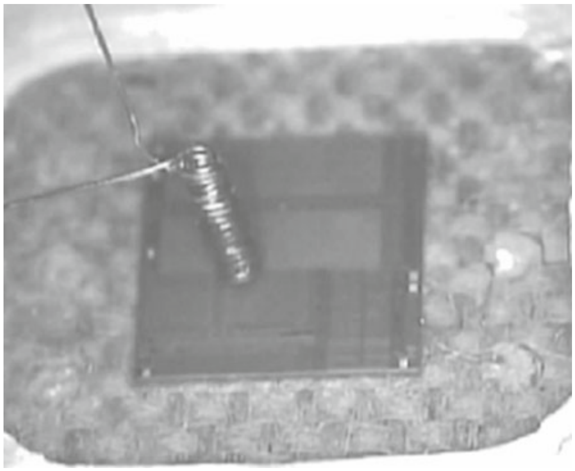
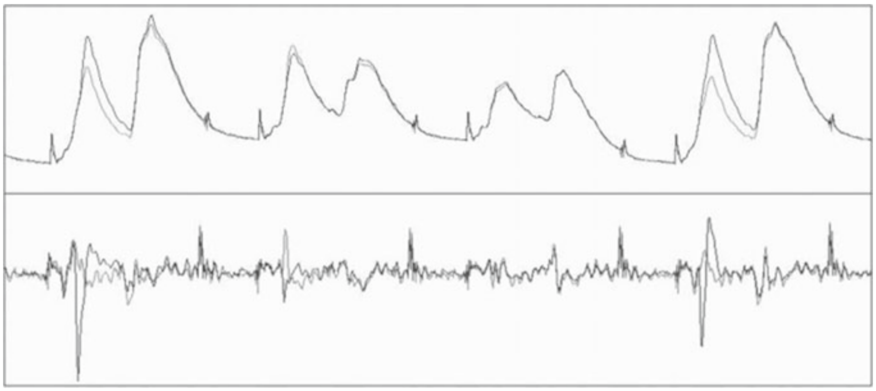


Fig. 9.11 Electromagnetic probing of a chip



Key: The upper traces represents the power consumption, and the lower traces represent the electromagnetic emanations during the same command. The black traces were taken were $\mathtt{FF}_{16}$ is being manipulated, and the grey traces where $\mathtt{00}_{16}$ is being manipulated.

Fig. 9.12 Power and electromagnetic measurements

The initial descriptions of attacks were based on acquired traces of the electromagnetic emanations of a microprocessor, as shown in Fig. 9.11. More recent descriptions have shown that at attacker can find low frequency leakage in complex System-on-Chip microprocessors [33] and even the metal case of a Hardware Security Module (HSM) or PC [18]. In these cases, an attack was realised by extracting a signal at a well-chosen frequency by filtering the signal before acquisition with an oscilloscope.

The signals that are acquired using this method are also different to those acquired by reading the instantaneous power consumption. The signals acquired during two executions of a selected command by an 8-bit microprocessor is shown in Fig. 9.12. The black traces show the acquired power and electromagnetic signals when the chip manipulates $\mathtt{FF}_{16}$, and the grey traces shows the same command where the microprocessor is manipulating $\mathtt{00}_{16}$. The difference in the black and grey traces representing the power consumption can be seen as a increase in the power consumption for short periods. The difference in the traces representing the electromagnetic emanations is caused by sudden changes in the electromagnetic field, shown by spikes in the signal at the same moment in time the difference in the power consumption can be observed.

The traces acquired from measuring the instantaneous electromagnetic emanations can be treated in exactly the same way as power consumption acquisitions [17, 45]. The acquisitions can be analysed individually, referred to as Simple ElectroMagnetic Analysis (SEMA), or treated statistically, referred to as Differential ElectroMagnetic Analysis (DEMA).

9.4.3.1 Leakage Detection

Differential power analysis, and similar attacks, target specific intermediate values generated during the computation of a cryptographic algorithm. When developing a side channel resistant implementation of a cryptographic implementation one needs to be able to determine that no attack is possible from *any* intermediate state. Initial methods to determine if an implementation could be attacked via side-channel analysis would attempt to attack the commonly targeted intermediate states.

Recent results have suggested that an implementation of a cryptographic algorithm can be tested, by determining if the leakage acquired during the processing of a fixed input can be distinguished from the leakage acquired during the processing of a random input [20].

One of the tests in the Test Vector Leakage Assessment (TVLA) methodology is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed plaintext and the other with random plaintexts. In applying this, one would take two sets of data, and conduct Welch's t-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same.

Consider two sets of acquisitions, of n_1 and n_2 samples, respectively. We can compute their respective sample means, $\bar{X}_1$ and$\bar{X}_2$, and respective sample standard deviations, σ_1 and σ_2. One can then compute a t-statistic using Welch's t-test:

$$\alpha = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}} \ , \tag{9.13}$$

where the result is distributed over a t-distribution with ν degrees of freedom, i.e. $\alpha \sim t(\nu)$. In practice, one would use the asymptotic result where the t-distribution is equivalent to the standard normal distribution, so ν does not need to be defined.

Goodwill et al. use $\alpha > 4.5$ to indicate the presence of leakage. Specifically, an $\alpha > 4.5$ gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of $\sim 1 \times 10^{-6}$.

Further results on how this type of test can be used are given in [49].

9.4.4 Countermeasures

There are several countermeasures for protecting cryptographic algorithms against side-channel attacks. Some countermeasures can either be implemented in hardware or software; only software implementations are considered here for simplicity. These countermeasures are listed below:

Constant Execution can be used to fix the time taken by an algorithm, so that no deductions on secret information can be made though timing analysis or SPA. This extends to individual processes being executed by a smart card. If a process takes different lengths of time depending on some secret information and the difference in time is made up by a dummy function, there is a good chance that this will be visible in the power consumption or electromagnetic emanations. It is, therefore, important that an algorithm is written so that the same code is executed for all the possible input values.

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. The algorithm can no longer be said to comply with the constant execution criteria given above, but any variation is completely independent of any secret information. This does not prevent any attacks, 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 problem of attempting to conduct side-channel attacks in the presence of random delays is described in [14].

Randomisation (or data whitening) is where the data is manipulated in such a way that the value present in memory is always masked with a random value. This randomisation remains constant for one execution, but will vary from one acquisition to another. This mask is then removed at the end of the algorithm to produce the ciphertext. Some ideas for implementing this countermeasure were proposed in [12], and an example of this sort of implementation applied to block ciphers can be found in [1].

The size of the random value used in block ciphers is generally limited as S-boxes need to be randomised before the execution of the block cipher. This is generally achieved by creating an alternative S-box in memory for each execution of the cryptographic algorithm using the algorithm given in Algorithm 2.

Algorithm 2: Randomising S-box Values

```
Input: $S = (s_0, s_1, s_2, \dots, s_n)_x$ containing the S-box, $\mathbf{R}$ a random $\in \{0, 1, \dots, n\}$, and $r$ a
       random $\in \{0, 1, \dots, x-1\}$
Output: $RS = (rs_0, rs_1, rs_2, \dots, rs_n)_x$

for $i = 0$ to $n$ do
    $rs_i \leftarrow s_{(i \oplus \mathbf{R})} \oplus r$ ;
end
return $RS$ ;
```

The random value used for masking the input data can be no larger than n, and the random value used for the output value can be no larger that x. In an implementation of DES $\mathbf{R} \in \{0, 1, \dots, 63\}$ and $r \in \{0, 1, \dots, 15\}$, the rest of the algorithm needs to be a carefully designed to produce values masked with R, and to be able to manipulate returned values masked with r.

This is not possible in the case of RSA, where the calculation methods do not facilitate the method described above. A method for randomising the calculation of an RSA signature is given in [26], where the signature generation can be calculated using the formula:

$$S = \left((M + r_1 \cdot N)^{d + r_2 \cdot \phi(N)} \bmod (r_3 \cdot N)\right) \bmod N \tag{9.14}$$

where ϕ is Euler's totient function and, r_1, r_2 and r_3 are small random values. The effect of the each of the small random values does not change the outcome, but the order of the squaring operations and multiplications required to compute S is randomised. This does not provide a totally secure algorithm as the modular exponentiation itself also has to be secured against SPA attacks. A discussion of these algorithms is given in [13].

Randomised Execution is the manipulation of data in a random order so that an attacker does not know what is being manipulated at a given moment in time. If, for example, n bytes are being XORed with n key bytes then it is prudent to do it in a random order. If an attacker wishes to determine which byte has been XORed at any particular time this will be infeasible given that the order that bytes are being manipulated is unknown.

This also inhibits any statistical analysis of a side channel (i.e. using DPA), 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. A discussion of this technique applied to DES is described in [37].

Limiting Key Usage can prevent an attack, since conducting a statistical attack, such as Differential Power Analysis, requires a certain number of traces with a fixed key. An effective countermeasure is to limit the number of uses of a given key to a threshold lower than that required to conduct a side channel attack. However, using this countermeasure places a burden on the smart card as keys need to be updated and some mechanism for synchronising the key with a server needs to be implemented. We refer the reader to [15] for further discussion of this countermeasure.

9.4.4.1 Remarks

The above list gives the countermeasures that would need to be applied to a cryptographic algorithm to render it secure against side-channel analysis. An attacker would, therefore, have to overcome the combination of all these countermeasures. For an extensive treatment of side-channel analysis, the interested reader is referred to [34].

9.5 Fault Analysis

The problem of faults occurring in microprocessors has existed for a relatively long time. One of the initial observations of faults being provoked in microprocessors was accidental. It was observed that radioactive particles produced by elements naturally present in packaging material caused faults in chips [35]. Specifically, these faults were caused by Uranium-235, Uranium-238 and Thorium-230 residues present in the packaging decaying to Lead-206 and releasing α particles. These particles were energetic enough to cause bits stored in RAM to change.

Further research involved the analysis of the effect of cosmic rays on semiconductors [54]. While cosmic rays are very weak at ground level, their effect in the upper atmosphere and outer space is important for the aero-spacial industry. This provoked research into integrity measures that need to be included in semiconductors used in the upper atmosphere and space.

In 1997, it was pointed out that a fault present in the generation of an RSA signature, computed using the Chinese Remainder Theorem, could reveal information on the private key [10] (this attack is detailed below). This led to further research into the effect of faults on the security of implementations of cryptographic algorithms in secure microprocessors, and the possible mechanisms that could be used to inject faults in a microprocessor.

9.5.1 Fault Injection Mechanisms

There are a variety of different mechanisms that can be used to inject faults in microprocessors. These are listed here:

Variations in Supply Voltage [3, 9] during execution may cause a processor to misinterpret or skip instructions.

Variations in the External Clock [3, 4, 32] may cause data to be misread (the circuit tries to read a value from the data bus before the memory has time to latch out the correct value) or an instruction miss (the circuit starts executing instruction $n + 1$ before the microprocessor has finished executing instruction n).

Extremes of Temperature [10, 21] may cause unpredictable effects in microprocessors. When conducting temperature attacks on smart cards, two effects can be obtained [6]: the random modification of RAM cells due to heating, and the exploitation of the fact that read and write temperature thresholds do not coincide in most Non-Volatile Memories (NVMs). By tuning the chip's temperature to a value where write operations work but read operations do not, or the other way around, a number of attacks can be mounted.

Laser Light [16, 23, 44] can be used to simulate the effect of cosmic rays in microprocessors. Laser light is used to test semiconductors that are destined to be used in the upper atmosphere or space. The effect produced in semiconductors is based on the photoelectric effect, where light arriving on a metal surface will induce a current. If the light is intense, as in laser light, this may be enough to induce a fault in a circuit.

White Light [3] has been proposed as an alternative to laser light to induce faults in microprocessors. This can be used as a relatively inexpensive means of fault induction [50]. However, white light is not directional and cannot easily be used to illuminate small portions of a microprocessor.

Electromagnetic flux [48] has also been shown to be able to change values in RAM, as eddy currents can be made strong enough to affect microprocessors. However, this effect has only been observed in insecure microprocessors.

9.5.2 Modelling the Effect of a Fault

The fault injection methods described above may have many different effects on silicon. They can be modelled in ways that depend on the type of fault injection that has been used. The following list indicates the possible effects that can be created by these methods:

Resetting Data: an attacker could force the data to the blank state, i.e. reset a given byte, or bytes, of data back to `00` or $\texttt{FF}_{16}$, depending on the logical representation of, for example, RAM cells.

Data Randomisation: an attacker 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.

Modifying Opcodes: an attacker could change the instructions executed by the chip's CPU, as described in [3]. 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.

These three types of attack cover everything that an attacker could hope to do to an implementation of an algorithm. It is not usually possible for an attacker to create all of these possible faults in any particular implementation. 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 effects available, if that effect is not taken into account then it may have catastrophic consequences for the security of a given implementation.

In the literature one-bit faults are often considered. This is a useful model for developing theoretical attacks, but has proven to be extremely difficult to produce on a secure microprocessor. The model given above is based on published descriptions of implementations of fault attacks.

9.5.3 Faults in Cryptographic Algorithms

The faults mechanisms and fault model described above can be used to attack numerous cryptographic algorithms. Two examples of fault attacks on cryptographic algorithms are described below.

9.5.3.1 Faults in RSA Signature Generation

The first published fault attack [10], proposed an attack focused on an implementation of RSA using the Chinese Remainder Theorem (CRT). The attack allows for a wide range of fault injection methods, as it only requires one fault to be inserted in order to factorise the RSA modulus.

The technique requires an attacker to obtain two signatures for the same message, where one signature is correct and the other is the result of the injection of a fault during the computation of S_p or S_q (see above). That is, the attack requires that one of S_p and S_q is computed correctly, and the other is computed incorrectly.

Without loss of generality, suppose that $S' = aS_p + bS'_q \bmod N$ is the faulty signature, where S_q is changed to $S'_q \neq S_q$. We then have:

$$\begin{aligned} \Delta &\equiv S - S' \pmod N \\ &\equiv (aS_p + bS_q) - (aS_p + bS'_q) \pmod N \\ &\equiv b(S_q - S'_q) \pmod N \ . \end{aligned} \tag{9.15}$$

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, we can derive the factors of N by observing that $p = \gcd(\Delta \bmod N, N)$ and $q = N/p$.

In summary, all that is required to break RSA is one correct signature and one faulty one. This attack will be successful regardless of the type or number of faults injected during the process, provided that all faults affect the computation of either S_p or S_q.

Although initially theoretical, this attack stimulated the development of a variety of fault attacks against a wide range of cryptographic algorithms. One of the first descriptions of an implementation of this attack is given in [5].

9.5.3.2 Faults in DES

A type of cryptanalysis of ciphertext blocks produced by injecting faults into DES was proposed in [8], based on using techniques used in differential cryptanalysis [36]. One-bit faults were assumed to occur in random places throughout an execution of DES. The ciphertext blocks corresponding to faults occurring in the fourteenth and fifteenth round were taken, enabling the derivation of the key. This was possible as the effect of a one-bit fault in the last three rounds of DES is visible in the ciphertext block when it is compared with a correct ciphertext block. This allowed the key to be recovered using between 50 and 200 different ciphertext blocks. It is claimed in [8] that, if an attacker can be sure of injecting faults towards the end of the algorithm, the same results could be achieved with only ten faulty ciphertext blocks, and that, if a precise fault could be induced, only three faulty ciphertext blocks would be required.

This algorithm was improved upon in [19]. When searching for a key, the number of times a given hypothesis is found is counted. This means that faults from earlier rounds can be taken into account. It is claimed in [19] that faults from the eleventh round onwards can be used to derive information on the key, and that in the ideal situation only two faulty ciphertext blocks are required.

The simplest case of a fault attack on DES involves injecting a fault in the fifteenth round, and such an attack is well-known within the smart card industry.

The last round of DES can be expressed in the following manner:

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

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

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

and

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

This provides an equation in which only the last subkey, K_{16}, is unknown. All of the other variables are available from the ciphertext block. This equation holds for each S-box in the last round, which means that it is possible to search for key hypotheses in sets of six bits, i.e. the 48-bit output after the XOR is divided into eight groups of six bits before being substituted with values from the S-boxes.

All 64 possible key values corresponding to the XOR just before each individual S-box can be used 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 eight-key bits that are not included in the last subkey, to find the entire key.

If R'_{15} is randomised by a fault, then the expected number of hypotheses that are generated can be predicted using the methods given in [7]. Table 9.1 shows the statistically expected number of key hypotheses E_k that would be returned by a fault producing a difference across each S-box in the last round. This is an average of the non-zero elements in the expected number of hypotheses that are generated using the tables defined in [7].

Table 9.1 The expected number of hypotheses per S-box for one faulty ciphertext block

S-box	E_k
1	7.54
2	7.67
3	7.58
4	8.36
5	7.73
6	7.41
7	7.91
8	7.66

The expected number of hypotheses for the last subkey will be the product of all eight expected values E_k; this gives an expected number of around 2^{24}. This is just for the last subkey, an actual exhaustive search will need to take into account the eight bits that are not included in the last subkey, giving an overall expected keyspace size of 2^{32}.

This substantially reduces the number of possible keys that would need to be tested to try and determine the secret key used. The size of the keyspace can be further reduced if the fault attack is repeated and the intersection of the two resulting keyspaces is determined.

The same attack can also be applied if small faults occur in the last five rounds of DES, but the treatment is statistical in nature and requires many more faults to determine information on the key. Further details of this attack, and a brief description of an implementation, are given in [19]. A more detailed analysis of how DES can be attacked using faults is given in [46].

9.5.4 Countermeasures

The countermeasures that can be used to protect microprocessors from fault attacks are based on methods previously employed for integrity purposes. However, countermeasures only need to be applied in processes where an attacker could benefit from injecting a fault, although a careful analysis of a given application is required to determine where countermeasures are required. This has proven to be true even where algorithms are based on one-time random numbers, as it has been shown that the manipulation of the random number can compromise the security of a cryptographic algorithm [41]. The list of countermeasures is given below:

Checksums can be implemented in software or hardware. This prevents data (such as key values) being modified by a fault, as the fault can be detected followed by appropriate action (see below).

Execution Randomisation can be used to change the order in which operations in an algorithm are executed from one execution to another, making it difficult to predict what the machine is doing at any given cycle. For most fault attacks this countermeasure will only slow down a determined attacker, as eventually a fault will hit the desired instruction. However, this will thwart attacks that require faults in specific places or in a specific order.

Random Delays can be used to increase the time required to attack. As with execution randomisation, a determined attacker can attempt to inject a fault until the moment the fault is injected coincides with the target. However, this can take significantly more time than would otherwise be required, especially if an attacker is able to identify a target through a side channel (e.g. using Simple Power Analysis).

Execution Redundancy is the repeating of algorithms and comparing the results to verify that the correct result is generated. This is most effective when the second calculation is different to the first, e.g. the inverse function, to prevent an attacker from trying to inject an identical fault in each execution.

Variable Redundancy is the reproduction of a variable in memory. When a variable is tested or modified the redundant copy is also tested or modified. This is most effective when the copy is stored in a different form to the original, e.g. the bitwise complement, to avoid a fault being applied to each variable in the same way.

Ratification Counters and Baits can be included to prevent an attacker from successfully completing a fault attack by rendering a microprocessor inoperative once a fault attack is detected. Baits are small ($<$10 byte) code fragments that perform an operation and test its result. A typical bait writes, reads and compares data, performs XORs, additions, multiplications and other operations whose results can be easily checked. When a bait detects an error it increments a counter in Non-Volatile Memory (NVM), and when this counter exceeds a tolerance limit (typically three) the microprocessor ceases to function.

9.5.4.1 Remarks

Many of the countermeasures in this list can be implemented in either hardware or software. A more complete list of the countermeasures (in hardware and software), along with a description of certain fault attacks is given in [27].

9.6 Embedded Software Design

The attacks described in the previous sections of this chapter have focused on attacking cryptographic algorithms to determine a secret or private key. In this section some example of how the attack methods presented in Sects. 9.4 and 9.5 can be applied to other security mechanisms are described. This is to demonstrate that implementing a secure application on a smart card is not trivial, and requires the careful evaluation of every implemented function. It should also be noted that the attacks described below are only possible where no specific countermeasures are implemented.

9.6.1 PIN Verification

As described in Sect. 9.3, the first smart cards included a contact that was called Vpp used to supply power to the microprocessor so that it could program the EEPROM present in the chip. At the time, the voltage supply (Vcc) did not supply enough power to allow a microprocessor to modify EEPROM and a higher voltage needed to be applied to the Vpp contact. The Vpp contact is no longer used as it led to security problems, as described below.

If the Vpp contact was masked (e.g. covered with nail varnish) then no power would be available for the microprocessor to program the EEPROM, but power would be available through the Vcc to run every other function of the smart card. This meant that an attacker could try every single PIN number without decrementing the PIN counter (typically a PIN counter is set to three and decremented with every false PIN presentation, once the PIN number is zero a smart card will render itself non-functional). This process could be automated using a standard PC and a smart card reader to determine a PIN number in matter of minutes.

After the Vpp contact was removed further problems were encountered. The most natural way to implement a PIN verification would be as shown in Algorithm 3.

Algorithm 3: Insecure PIN Verification Algorithm

```
Input: PIN
Output: Whether the PIN is valid, true or false.

if (PINcounter > 0) then
    PIN ← RequestPIN();
else
    return false ;
end
if PIN ≠ UserPIN then
    PINcounter ← PINcounter − 1 ;
    return false ;
else
    return true ;
end
```

The PIN entered by a user is returned by the RequestPIN function and compared with the PIN in Non-Volatile Memory (NVM). If the entered PIN is not equal to the stored PIN, the PINcounter will be decremented.

It was observed that the power consumption increased when a smart card modified the value of the PIN counter, i.e. it was visible using the SPA techniques described in Sect. 9.4 as an increase in the power consumption. Attackers then developed tools to cut the power being supplied to a microprocessor once this increase in power consumption was detected. This allowed automated tools to attempt every PIN number until the correct PIN was found. The correct PIN would be the only value where the command would finish as the microprocessor would not attempt to modify the NVM.

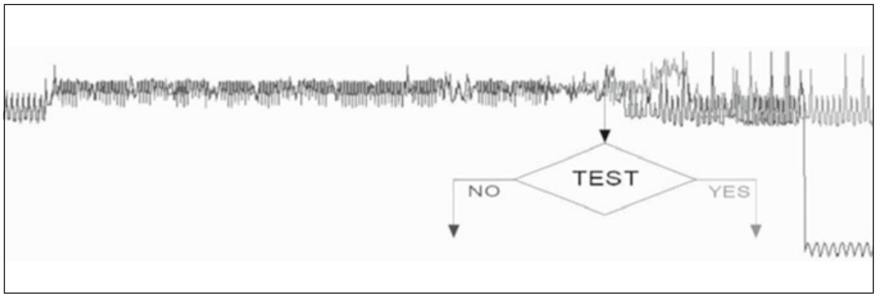
This made it necessary to change the algorithm used to verify a PIN number. Typically, a secure PIN verification will be implemented in the following manner: where the PINcounter is decremented before it is tested, and only incremented if the PIN is entered correctly. The power supply can be removed at any point during the command without producing a security problem. However, further modifications would need to be made to render it resistant to fault attacks. An example of a fault attack against an operating system is described in Algorithm 4.

Algorithm 4: Secure PIN Verification Algorithm

```
Input: PIN
Output: Whether the PIN is valid, true or false.
if (PINcounter > 0) then
    PIN ← RequestPIN() ;
else
    return false ;
end
PINcounter ← PINcounter − 1; if (PIN = UserPIN) then
    PINcounter ← PINcounter + 1; return true ;
else
    return false ;
end
```

9.6.2 *File Access*

Another possible target within a smart card operating system is the file structure. All personalisation information, e.g. PIN numbers etc., is stored in a file structure situated in NVM. Each file will have a set of access conditions that determine who can read and write to each file or directory. For example a user's PIN number on a SIM card, unless intentionally disabled, will grant access to the files that contain SMS messages once verified. If the PIN is not verified access to these files will be denied. There are often administrator identification codes (essentially eight digit PIN numbers) that grant access to more files and allow the modification of files that the end user is not able to directly modify, e.g. the user's PIN number.



Key: Black Power consumption where file access is denied.
Grey Power consumption where file access is granted.

Fig. 9.13 Determining the moment file access is granted using the power consumption

If an attacker wishes to attempt to access information stored in files without any of the codes mentioned above, a fault attack could be attempted. An attacker can attempt to inject a fault at the moment a smart card is evaluating whether the right to read a file, for example, should be granted. If successful, the evaluation will be erroneous and the right to access the file will be temporarily granted.

In order to determine the point in which a fault would need to be injected, an attacker can use SPA (see Sect. 9.4). An attacker could compare a trace of the power consumption where file access is granted with a trace where file access has been denied. An example of this is shown in Fig. 9.13. The black trace represents the power consumption where access has been denied. The grey trace represents the power consumption where access has been granted. It can be see at the point indicated in the figure that the two traces diverge. This should represent the moment at which the access conditions are evaluated and will, therefore, be the targeted area for a fault attack.

On a smart card there are typically files that contain serial numbers and such information, which can be read by anyone. Finding two files to attempt to read in order to generate traces, as shown in Fig. 9.13, should be straightforward.

This type of fault attack means that the access conditions to files, and other security mechanisms such as PIN verification, need to include redundancy in their tests to ensure that a fault attack is not possible. The various countermeasures that can be implemented are described in Sect. 9.5.

9.7 In Conclusion

This chapter presents the particular security considerations that need to be taken into account when implementing a secure smart card-based application. Implementations of all the commands on a smart card need to be subjected to careful analysis to prevent power analysis and fault injection techniques from compromising the security of the smart card.

Research in the domain of smart card security is typically a cyclic process. New attacks are developed against algorithm implementations, standards, etc. and countermeasures are proposed. The modifications are then reviewed for potential vulnerabilities and further countermeasures proposed if required. The aim of this process is to remain sufficiently ahead of what can be achieved by an individual attacker that smart cards remain secure throughout the period that they are active.

References

1. Akkar, M.-L. and Giraud, C. (2001). An implementation of DES and AES secure against some attacks. In Koç, C. K., Naccache, D., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems—CHES 2001,* volume 2162 of *Lecture Notes in Computer Science,* pages 309–318. Springer-Verlag.
2. American National Standards Institute (1985). *Financial Institution Key Management (Wholesale).* American National Standards Institute.
3. Anderson, R. and Kuhn, M. (1996). Tamper resistance—a cautionary note. In *Proceedings of the Second USENIX Workshop of Electronic Commerce,* pages 1–11.
4. Anderson, R. and Kuhn, M. (1997). Low cost attacks on tamper resistant devices. In Christianson, B., Crispo, B., Lomas, T. M. A., and Roe, M., editors, *Security Protocols,* volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag.
5. Aumüller, C., Bier, P., Hofreiter, P., Fischer, W., and Seifert, J.-P. (2002). Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In Kaliski, B. S., Koç, C. K., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems—CHES 2002,* volume 2523 of *Lecture Notes in Computer Science,* pages 260–275. Springer-Verlag.
6. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. (2006). The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE,* 94(2):370–382.
7. Biham, E. and Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. In Menezes, A. and Vanstone, S., editors, *Advances in Cryptology—CRYPTO '90,* volume 537 of *Lecture Notes in Computer Science,* pages 2?-21. Springer-Verlag.
8. Biham, E. and Shamir, A. (1997). Differential fault analysis of secret key cryptosystems. In Kaliski, B. S., editor, *Advances in Cryptology—CRYPTO '97,* volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag.
9. Blömer, J. and Seifert, J.-P. (2003). Fault based cryptanalysis of the advanced encryption standard (AES). In Wright, R. N., editor, *Financial Cryptography—FC 2003,* volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer-Verlag.
10. Boneh, D., DeMillo, R. A., and Lipton, R. J. (1997). On the importance of checking computations. In Fumy, W., editor, *Advances in Cryptology—EUROCRYPT '97,* volume 1233 of *Lecture Notes in Computer Science,* pages 37–51. Springer-Verlag.
11. Brier, E., Clavier, C., and Olivier, F. (2004). Correlation power analysis with a leakage model. In Joye, M. and Quisquater, J.-J., editors, *Cryptographic Hardware and Embedded Systems—CHES 2004,* volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer-Verlag.
12. Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. (1999). Towards approaches to counteract power-analysis attacks. In Wiener, M., editor, *Advances in Cryptology—CRYPTO '99,* volume 1666 of *Lecture Notes in Computer Science,* pages 398–412. Springer-Verlag.
13. Chevallier-Mames, B., Ciet, M., and Joye, M. (2004). Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers,* 53(6):760–768.
14. Clavier, C., Coron, J.-S., and Dabbous, N. (2000). Differential power analysis in the presence of hardware countermeasures. In Koç, C. K. and Paar, C., editors, *Cryptographic Hardware*

and Embedded Systems—CHES 2000, volume 1965 of *Lecture Notes in Computer Science,* pages 252–263. Springer-Verlag.

15. Dobraunig, C., Eichlseder, M., Mangard, S. and Mendel, F. (2014). On the Security of Fresh Re-keying to Counteract Side-Channel and Fault Attacks. In Joye, M. and Moradi, A., editors, *Smart Card Research and Advanced Applications—13th International Conference, CARDIS 2014*, volume 8968 of *Lecture Notes in Computer Science*, pages 233–244. Springer-Verlag.
16. Fouillat, P. (1990). *Contribution à l'étude de l'interaction entre un faisceau laser et un milieu semiconducteur, Applications à l'étude du Latchup et à l'analyse d'états logiques dans les circuits intégrés en technologie CMOS.* PhD thesis, University of Bordeaux.
17. Gandolfi, K., Mourtel, C., and Olivier, F. (2001). Electromagnetic analysis: Concrete results. In Koç, C. K., Naccache, D., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems—CHES 2001,* volume 2162 of *Lecture Notes in Computer Science,* pages 251–261. Springer-Verlag.
18. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E. (2015). Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation. In Güneysu, G. and Handschuh, H., editors, *Cryptographic Hardware and Embedded Systems—CHES 2015,* volume 9293 of *Lecture Notes in Computer Science,* pages 207–228. Springer-Verlag.
19. Giraud, C. and Thiebeauld, H. (2004). A survey on fault attacks. In Deswarte, Y. and Kalam, A. A. El, editors, *Smart Card Research and Advanced Applications VI—18th IFIP World Computer Congress,* pages 159–176. Kluwer Academic.
20. Goodwill, G., Jun, B., Jaffe, J. and Rohatgi, P. (2011). A testing methodology for side-channel resistance validation. In *The Non-Invasive Attack Testing Workshop—NIAT 2011.*
21. Govindavajhala, S. and Appel, A. W. (2003). Using memory errors to attack a virtual machine. In *IEEE Symposium on Security and Privacy 2003,* pages 154–165.
22. Gutmann, P. (2004). *Security Architecture.* Springer.
23. Habing, D. H. (1992). The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits. *IEEE Transactions On Nuclear Science,* 39:1647–1653.
24. International Organization for Standardization (1997). *ISO/IEC 7816–3 Information technology—Identification cards—Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols.* International Organization for Standardization.
25. International Organization for Standardization (1999). *ISO/IEC 7816–2 Identification cards—Integrated circuit cards—Part 2: Cards with contacts—Dimensions and location of the contacts.* International Organization for Standardization.
26. Joye, M. and Olivier, F. (2005). Side-channel attacks. In van Tilborg, H., editor, *Encyclopedia of Cryptography and Security,* pages 571–576. Kluwer Academic Publishers.
27. Joye, M. and Tunstall, M., Eds (2015). Fault Analysis in Cryptography. Springer.
28. Kahn, D. (1997). *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Simon & Schuster Inc., second edition.
29. Knuth, D. (2001). *The Art of Computer Programming,* volume 2, Seminumerical Algorithms. Addison–Wesley, third edition.
30. Kocher, P. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., editor, *Advances in Cryptology—CRYPTO '96,* volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag.
31. Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In Wiener, M. J., editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag.
32. Kommerling, O. and Kuhn, M. (1999). Design principles for tamper resistant smartcard processors. In *USENIX Workshop on Smartcard Technology,* pages 9–20.
33. Longo Galea, J., De Mulder, E., Page, D. and Tunstall, M. (2015). SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip. In Güneysu, G. and Handschuh, H., editors, *Cryptographic Hardware and Embedded Systems—CHES 2015,* volume 9293 of *Lecture Notes in Computer Science,* pages 620–640. Springer-Verlag.
34. Mangard, S., Oswald, E., and Popp, T. (2007). *Power Analysis Attacks—Revealing the Secrets of Smart Cards.* Springer-Verlag.

35. May, T. and Woods, M. (1978). A new physical mechanism for soft errors in dynamic memories. In 16*th* International Reliability Physics Symposium.
36. Menezes, A., van Oorschot, P., and Vanstone, S. (1997). *Handbook of Applied Cryptography.* CRC Press.
37. Messerges, T. S. (2000). *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms.* PhD thesis, University of Illinois, Chicago.
38. Meyer, C. (2000). Private communication. Carl Meyer was one of the designers of the DES algorithm.
39. MIPS-Technologies (2001). MIPS$^{\text{TM}}$ architecture for programmers volume I: Introduction to the MIPS32$^{\text{TM}}$ architecture. Technical Report MD00082, Revision 0.95.
40. Murdocca, M. and Heuring, V. P. (2000). *Principles of Computer Architecture.* Addison-Wesley.
41. Naccache, D., Nguyen, P. Q., Tunstall, M., and Whelan, C. (2005). Experimenting with faults, lattices and the DSA. In Vaudenay, S., editor, *Public Key Cryptography—PKC 2005,* volume 3386 of *Lecture Notes in Computer Science,* pages 16–28. Springer-Verlag.
42. NIST (1999). *Data Encryption Standard (DES) (FIPS-46–3).* National Institute of Standards and Technology.
43. NIST (2001). *Advanced Encryption Standard (AES) (FIPS-197).* National Institute of Standards and Technology.
44. Pouget, V. (2000). *Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés.* PhD thesis, University of Bordeaux.
45. Quisquater, J.-J. and Samyde, D. (2001). Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Attali, I. and Jensen, T. P., editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards—E-smart 2001,* volume 2140 of *Lecture Notes in Computer Science,* pages 200–210. Springer-Verlag.
46. Rivain, M. (2009). Differential Fault Analysis on DES Middle Rounds. Clavier, C. and Kris, G., editors, *Cryptographic Hardware and Embedded Systems—CHES 2009,* volume 5747 of *Lecture Notes in Computer Science,* pages 457–469. Springer-Verlag.
47. Rivest, R., Shamir, A., and Adleman, L. M. (1978). Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM,* 21(2):120–126.
48. Samyde, D., Skorobogatov, S. P., Anderson, R. J., and Quisquater, J.-J. (2002). On a new way to read data from memory. In *Proceedings of the First International IEEE Security in Storage Workshop,* pages 65–69.
49. Schneider, T. and Moradi, A. (2015). Leakage Assessment Methodology—A Clear Roadmap for Side-Channel Evaluations. In Güneysu, G. and Handschuh, H., editors, *Cryptographic Hardware and Embedded Systems—CHES 2015,* volume 9293 of *Lecture Notes in Computer Science,* pages 495–513. Springer-Verlag.
50. Skorobogatov, S. and Anderson, R. (2002). Optical fault induction attacks. In Kaliski, B. S., Ç. K. Koç, and Paar, C., editors, *Cryptographic Hardware and Embedded Systems—CHES 2002,* volume 2523 of *Lecture Notes in Computer Science,* pages 2–12. Springer-Verlag.
51. Skorobogatov, S. P. (2005). *Semi-Invasive Attacks—A New Approach to Hardware Security Analysis.* PhD thesis, University of Cambridge. available at http://www.cl.cam.ac.uk/TechReports/.
52. Tarnovsky, C., (2015) https://en.wikipedia.org/wiki/Christopher_Tarnovsky. Accessed 16 November 2015.
53. Wright, P. (1987). *Spycatcher.* Heineman.
54. Ziegler, J. (1979). Effect of cosmic rays on computer memories. *Science,* 206:776–788.