Techniques of Side Channel Cryptanalysis

by

James Alexander Muir

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Combinatorics and Optimization

Waterloo, Ontario, Canada, 2001

Abstract

The traditional model of cryptography examines the security of cryptographic primitives as mathematical functions. This approach does not account for the physical side effects of using these primitives in the real world. A more realistic model employs the concept of a *side channel*. A side channel is a source of information that is inherent to a physical implementation of a primitive. Research done in the last half of the 1990s has shown that the information transmitted by side channels, such as execution time, computational faults and power consumption, can be detrimental to the security of ciphers like DES and RSA.

This thesis surveys the techniques of side channel cryptanalysis presented in [30], [10], and [31] and shows how side channel information can be used to break implementations of DES and RSA. Some specific techniques covered include the timing attack, differential fault analysis, simple power analysis and differential power analysis. Possible defenses against each of these side channel attacks are also discussed.

Acknowledgements

I was introduced to the concept of power analysis at the 1st CACR Information Security Workshop in November 1998. The following May, I joined the Secure Systems group at Pitney Bowes in Shelton, CT where I gained practical experience in this area by working in their power analysis lab. I owe many thanks to both the CACR and Pitney Bowes; the opportunities they have provided me have enriched both my experience as a Masters student and the contents of this thesis.

I would like to thank the Department of Combinatorics and Optimization and NSERC for their financial support during my studies. Also, a special thank-you is extended to Kim Gingerich and Lori McConnell for their help and encouragement in pursuing NSERC funding.

Many of my friends and fellow graduate students lent me their attention and patience as I worked through this thesis. In particular, I would like to thank Kristi Herridge, John Irving, Debbie Maclean, John Proos, Chris Snyder, and Kerri Webb.

Above all, I would like to thank my family for their love and support.

The Road goes ever on and on Down from the door where it began.
Now far ahead the Road has gone, And I must follow, if I can,
Pursuing it with eager feet, Until it joins some larger way,
Where many paths and errands meet. And whither then? I cannot say.

Contents

List of Figures

Chapter 1

Background

Mathematical abstraction can be a very useful tool in the study of cryptographic primitives. Cryptographers often evaluate the security of ciphers by considering them as mathematical functions used in a scenario similar to the one described in Figure 1.1.

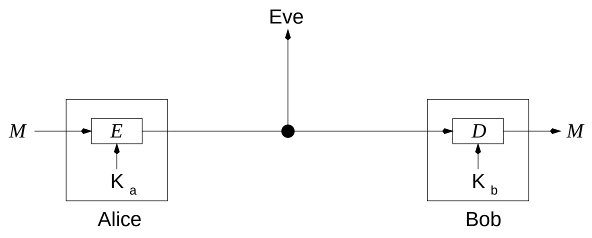

FIGURE 1.1: The traditional cryptographic model.

In this model, two people, Alice and Bob, attempt to use a cipher to engage in a private conversation across a public channel. An eavesdropper, Eve, monitors the public channel and tries to deduce what Alice and Bob are talking about. Eve has at her disposal all the details of the cipher, except for the secret key (this is known as Kerckhoff's assumption), a few plaintext-ciphertext pairs generated by either Alice or Bob, as well some reasonable amount of computing power.

Traditionally, any cipher which resisted Eve's scrutiny in this model was thought to be secure. Whether or not such a cipher would be implemented in the real world was then a matter of practicality (e.g., key length, encryption speed, memory requirements). However, as this thesis will illustrate, ciphers which are secure when specified as mathematical functions are not necessarily secure in real world implementations.

In reality, ciphers are implemented on physical devices which interact with and are influenced by their environments. Electronic devices, like pagers and smart cards, consume power and emit radiation as they operate; they also react to temperature changes and electromagnetic fields. These physical interactions can be instigated and monitored by adversaries, like Eve, and may result in information which is useful in cryptanalysis.

An insightful demonstration of this point is related by Peter Wright in [52]. He explains that in 1956, the British intelligence organization, MI5, was trying to break a cipher used by the Egyptian Embassy in London, but their efforts were stymied by the limits of their computational power. Wright, a scientist with GCHQ at the time, suggested that a carefully placed microphone might help. The Egyptians were using a Hagelin machine, a rotor based cipher, and after some tests Wright discovered that the audible click which occurred as the rotors turned could be exploited. During a special service call to fix a faulty telephone in the embassy, a microphone was placed close to the Hagelin machine. By listening to the clicks of the rotors as cipher clerks reset them each morning, MI5 was able to deduce the core position of 2 or 3 of the machine's rotors. This extra information allowed the task of calculating the initial setting of the Hagelin machine to fall within the means of MI5 computing resources, and subsequently allowed them to read the embassy's communications for several years.

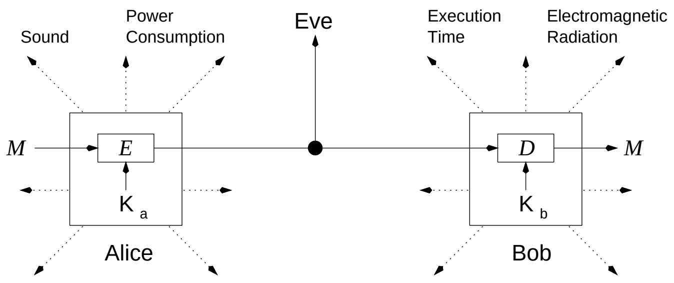

FIGURE 1.2: A model which includes side channels.

The traditional cryptographic model does not account for the physical side effects of using ciphers in the real world. A more realistic model can be described using the concept of a *side channel*, as shown in Figure 1.2. A side channel is a source of information that is inherent to a physical implementation. MI5's break of the Hagelin cipher exploited a side channel consisting of sound, but there are many others.

The chapters of this thesis demonstrate how the analysis of side channel information can be used in cryptanalysis. In particular, three kinds of side channels are examined: execution time, computational faults and power consumption. The academic research in these three topics was initiated by Kocher [30], Boneh, DeMillo and Lipton [10], and Kocher, Jaffe, and Jun [31], respectively. The techniques of side channel cryptanalysis presented here comprise a survey of their work.

Chapter 2

Timing Analysis

2.1 Introduction

Commercial cryptographers have long been concerned with how much execution time their cryptographic implementations require. The amount of time used to encrypt a message or produce a digital signature is often used as a benchmark when comparing different cryptographic schemes; with all other factors being equal, the fastest scheme is considered the most efficient and is hence the most marketable.

The amount of time it takes to compute a cryptographic function depends not only on what that function does but also what inputs are passed to it. Certain encodings of messages may require less time to encrypt because of the mathematical operations used. For example, an encryption function based on integer multiplication might be quick to evaluate with pen and paper if the message to encrypt is a power of ten. A prudent cryptographer might then try to express every message as a power of ten to exploit this computational shortcut. However, in addition to messages, cryptographic functions often take secret keys as input and so the value of a key might influence publicly observably timing characteristics.

On 29 November 1995, Paul Kocher described how the timing characteristics of cryptosystems such as RSA, DSS and Diffie-Hellman can be correlated to the values of their secret keys. He further outlined how an attacker is able to analyze measurements of the time it takes to compute several, say, RSA signatures and deduce the signing entity's secret key. After a preliminary version of Kocher's results circulated, the cryptographic community began to realize that some products and protocols currently in use were vulnerable to the attack (e.g., SSL). With the growing popularity of electronic commerce, this new method of cryptanalysis made quite a good story; it even made the front page of the New York Times [35].

Outline

We first describe the idea behind Kocher's timing attack on modular exponentiation. Next, we give details about how the attack can be applied to the modular exponentiation routine in the freely available RSAREF 2.0 cryptographic toolkit. An analysis of the attack is then presented which allows us to estimate the number of timing measurements required to extract a secret exponent. A modification of the attack is then discussed, as well as other cryptosystems and operations which are potentially vulnerable to timing analysis. We end by presenting a countermeasure which makes RSA immune to this version of the timing attack.

2.2 The Idea

An operation which is fundamental to the RSA cryptosystem is modular exponentiation. It is used to encrypt and decrypt as well as to sign message blocks. When RSA was introduced, the inventors suggested a *repeated square-and-multiply* algorithm (see Figure 2.1) as a way to implement this operation efficiently [46]. Several RSA implementations followed this example including RSAREF, a reference implementation authored by RSA Laboratories.

Figure 2.1 describes the left-to-right square-and-multiply algorithm. The algorithm's parameters are labeled using notation from common descriptions of the RSA cryptosystem. The output S can be thought of as a digital signature. The private exponent d can be represented using at most n bits where n is the bit length of the RSA modulus N.

```
INPUT: $M, N, d = (d_{n-1}d_{n-2}\ldots d_1d_0)_2$
OUTPUT: $S = M^d \mod N$

1 $S \leftarrow 1$
2 **for** $j = n-1 \ldots 0$ **do**
3     $S \leftarrow S^2 \mod N$
4     **if** $d_j = 1$ **then**
5         $S \leftarrow S \cdot M \mod N$
6 **return** $S$
```

FIGURE 2.1: The left-to-right repeated square-and-multiply algorithm for modular exponentiation.

Kocher made some important observations about square-and-multiply algorithms. In Figure 2.1, the conditional expression at line 4 causes the execution path of this algorithm to vary according to the value of the exponent. In any loop iteration, if the relevant bit of d is one, then both a modular square and multiply are performed (lines 3 and 5 respectively); if the relevant bit is zero only a modular square is performed. So, the required amount of computation, and hence execution time, to complete the n loop iterations is influenced by the value of the exponent.

If an attacker could observe and compare the execution time of several loop iterations in the square-and multiply algorithm, he or she may be able to deduce the value of the corresponding exponent bits. This technique, when applied against an RSA signature operation, would reveal bits of the signer's private key. However, it is not clear how an attacker might observe the timing characteristics of individual loop iterations[1]. Kocher's timing attack describes how an attacker can use the *total* execution time of the algorithm to deduce bits of the private exponent. This timing information can be easily observed by a passive attacker.

Suppose that a malicious user, Marvin, sends a series of signature requests to a PC that implements RSA using the repeated square-and-multiply algorithm. Marvin records the times $T_1, T_2, \ldots, T_k$ it takes the PC to return a signature on each of the known messages $M_1, M_2, \ldots, M_k \in \mathbb{Z}_N$. The attack now proceeds to allow Marvin to recover the bits of d one at a time.

Since $d < N$ and n is the bit length of N, the binary representation of d may contain leading zeroes[2], but to simplify our discussion we will assume that $d_{n-1} = 1$.

[1]We will see in Chapter 4 how the execution time of individual loop iterations can be deduced using power analysis.

[2]In practice, any leading zeroes in $d_{n-1}d_{n-2}\ldots d_1d_0$ are skipped to reduce the number of loop iteration required in the square-and-multiply algorithm. However, this implementation detail is

Tracing through the pseudo-code of Figure 2.1, Marvin knows that at the start of the second loop iteration $S = M \mod N$ and then, after the squaring step, $S = M^2 \mod N$. If $d_{n-2} = 1$, the PC computes the product $M \cdot M^2 \mod N$, otherwise it does not. Using his knowledge of the physical specifications of the target PC, Marvin simulates on an identical PC (i.e., same processor, RAM cache, etc.) the time $\hat{t}_i$ it takes to compute $M_i^2 \cdot M_i \mod N$ for each of the known messages. The value of M_i influences the amount of time required to perform this calculation[3].

Kocher noticed that, when $d_{n-2} = 1$, the two ensembles $\{\hat{t}_i\}$ and $\{T_i\}$ are correlated. For example, if $\hat{t}_i$ is much larger than its expectation, then T_i is also likely to be larger than its expectation. If $d_{n-2} = 0$, then the two ensembles behave as independent random variables. By measuring the correlation Marvin can decide the value of d_{n-2}. Now Marvin knows the value of S at the start of the third loop iteration. To get d_{n-3} Marvin reconstructs the ensemble $\{\hat{t}_i\}$ by simulating the time it takes the PC to compute $S \cdot M \mod N$, where the value of S is known, and compares it with the ensemble $\{T_i\}$. Marvin continues in this way to recover the remaining bits of d.

2.3 Attack Details

The principles underlying the timing attack are elementary, but there are several details which must be addressed when putting it into practice. For example, it is

incidental to our method of attack.

[3]In classical implementations of modular multiplication the product $M_i \cdot M_i^2$ is first calculated in $\mathbb{Z}$ and then reduced modulo N. Since it takes more time to multiply large numbers together than small ones the value of M_i influences the required computation time.

not clear how Marvin might measure the correlation between the various ensembles or how many timing measurements are required for a successful attack. Answers to these points depend upon the characteristics of the target implementation, but the techniques Kocher describes in [30] offer some direction.

We analyze Kocher's method of attack and explain how it can be applied against modular exponentiation in the RSAREF cryptographic library.

2.3.1 RSAREF 2.0

RSAREF 2.0 was released by RSA Laboratories in 1994. It was intended as an educational reference implementation of some common cryptographic schemes. Included in the RSAREF 2.0 library are routines for Diffie-Hellman key agreement and RSA signatures. In both systems, modular exponentiation is accomplished via the function `NN-ModExp`. Pseudo-code is given for `NN-ModExp` in Figure 2.2.

The algorithm in Figure 2.2 is a generalization of the basic square-and-multiply algorithm presented earlier and it inherits similar timing properties. When used to calculate an RSA signature, the algorithm first computes the values $M^2 \mod N$ and $M^3 \mod N$, and then 2 bits of the private exponent are processed at a time. Each loop iteration does two squaring operations and, if either exponent bit is nonzero, one multiply operation.

RSAREF does multiplication in $\mathbb{Z}_N$ by first calculating a product in $\mathbb{Z}$ and then reducing it modulo N. Squares are calculated using the same technique. The execution time of this simple method is related to the Hamming weight of the factors. The function `NN-ModMult` is used to evaluate each operation.

```
INPUT: $M, N, d = (d_{n-1}d_{n-2}\dots d_1d_0)_2$
OUTPUT: $S = M^d \mod N$

1  $m_1 \leftarrow M \mod N$
2  $m_2 \leftarrow m_1 \cdot M \mod N$
3  $m_3 \leftarrow m_2 \cdot M \mod N$
4  $S \leftarrow 1$
5  for $j = n-1 \dots 0$ by 2 do
6      $S \leftarrow S^2 \mod N$
7      $S \leftarrow S^2 \mod N$
8      if $(d_jd_{j-1})_2 \neq 0$ then
9          $S \leftarrow S \cdot m_{(d_jd_{j-1})_2} \mod N$
10 return $S$
```

FIGURE 2.2: A left-to-right repeated square-and-multiply method which uses a two bit window.

A common misconception about the timing attack is that it only determines whether or not a conditional multiplication is performed. If this were the case then the attack would not succeed against the algorithm in Figure 2.2. Knowing that a multiplication is executed in a particular loop iteration would only eliminate one of four possible values for the relevant pair of exponent bits. To determine the value of a pair of exponent bits it is necessary to know what operands were used in the conditional multiplication. The timing attack is able to exploit timing variation in the multiplications and the squares to do just that.

Suppose Alice and Marvin engage in a signature protocol using their PC's.

When Marvin sends a message to Alice, she uses the RSAREF routines and her private key pair $\langle N, d\rangle$ to sign it. Alice then sends her signature to Marvin. Marvin records the time T_i that it takes Alice to respond after he sends her the message M_i.

There are several factors which contribute to the value of T_i. Returning to Figure 2.2, the time required to perform the operations on lines 1 to 4 makes a contribution which we denote by c_i. In the loop of Figure 2.2, for particular value of j, the time required to execute lines 6, 7 and 9 also contributes to T_i. We denote these contributions by $r_{i,j}$, $s_{i,j}$, and $t_{i,j}$, respectively. Note that $r_{i,j}$ and $s_{i,j}$ are strictly positive values, but $t_{i,j}$ may be zero. Other factors, such as measurement error and transmission distance, also contribute to T_i and may be treated as sources of error. We denote these contributions by e_i. Now, we can write:

$$\begin{aligned}T_i &= e_i + c_i + (r_{i,n-1} + s_{i,n-1} + t_{i,n-1}) + (r_{i,n-3} + s_{i,n-3} + t_{i,n-3})\\ &\quad + \cdots + (r_{i,1} + s_{i,1} + t_{i,1})\\ &= e_i + c_i + \sum_j (r_{i,j} + s_{i,j} + t_{i,j}).\end{aligned}$$

The bits of Alice's secret exponent influence the value of almost all of the components in this sum. For a particular value of j, the operands used in the two squaring operations are completely determined by the value of the exponent bits $d_{n-1}d_{n-2}\ldots d_{j+2}d_{j+1}$. The operands used in the multiplication step are affected by these same bits as well as the bits $d_j d_{j-1}$. Thus, the the components $r_{i,j}$, $s_{i,j}$, and $t_{i,j}$ are all influenced by exponent bits. The value of c_i is influenced only the by the value of M_i.

Consider the first loop iteration of `NN-ModExp`. Using a PC identical to Alice's, Marvin can simulate and time the four possible sets of calculations Alice performed in the first loop iteration when she signed the message M_i. Effectively, Marvin generates four candidates for the value of $c_i + r_{i,n-1} + s_{i,n-1} + t_{i,n-1}$. To construct each candidate, Marvin can simply sign the message M_i four times using the exponents $00, 01, 10$ and 11. Denote the time required for these four signatures by $\hat{T}_{i,n-1,0}, \hat{T}_{i,n-1,1}, \hat{T}_{i,n-1,2}$ and $\hat{T}_{i,n-1,3}$ where the first two indices indicate the relevant message and loop iteration, and the last index represents a guess for the bits $d_{n-1}d_{n-2}$. Marvin can construct the following table:

00	01	10	11
$T_1 - \hat{T}_{1,n-1,0}$	$T_1 - \hat{T}_{1,n-1,1}$	$T_1 - \hat{T}_{1,n-1,2}$	$T_1 - \hat{T}_{1,n-1,3}$
$T_2 - \hat{T}_{2,n-1,0}$	$T_2 - \hat{T}_{2,n-1,1}$	$T_2 - \hat{T}_{2,n-1,2}$	$T_2 - \hat{T}_{2,n-1,3}$
$T_3 - \hat{T}_{3,n-1,0}$	$T_3 - \hat{T}_{3,n-1,1}$	$T_3 - \hat{T}_{3,n-1,2}$	$T_3 - \hat{T}_{3,n-1,3}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

In one of the four columns, Marvin's simulated operations will be the same as the operations Alice actually performed up to the end of the first loop iteration. In this column, Marvin's candidate value will hopefully be closer to $c_i+r_{i,n-1}+s_{i,n-1}+t_{i,n-1}$ than the three other candidate values. As the analysis in the following section shows, with high probability this will cause the *sample variance*[4] of the correct column to be lower than others. By comparing the four sample variances, Marvin can determine the value of $d_{n-1}d_{n-2}$.

[4]This statistic is usually denoted S^2. If $Y_1, Y_2, \ldots Y_k$ is a set of observations and $\overline{Y}$ is their arithmetic mean, then $S^2 = \frac{1}{k-1}\sum_{i=1}^{k}(Y_i - \overline{Y})^2$

The next pair of exponents bits, $d_{n-3}d_{n-4}$, can be deduced by timing the four possible sets of calculations Alice performed before the end of the second loop iteration. For each message, Marvin can measure the time it takes to sign M_i using the four exponents $d_{n-1}d_{n-2}00$, $d_{n-1}d_{n-2}01$, $d_{n-1}d_{n-2}10$ and $d_{n-1}d_{n-2}11$. Denote the time required for these four signatures by $\hat{T}_{i,n-3,0}$, $\hat{T}_{i,n-3,1}$, $\hat{T}_{i,n-3,2}$ and $\hat{T}_{i,n-3,3}$. Marvin then reconstructs his table with rows of the form:

$T_i - \hat{T}_{i,n-3,0}$	$T_i - \hat{T}_{i,n-3,1}$	$T_i - \hat{T}_{i,n-3,2}$	$T_i - \hat{T}_{i,n-3,3}$

Again, Marvin calculates the sample variance of each column to determine the actual bit values. The value of the other pairs of bits may be decided in turn using similar tables.

2.3.2 Analysis

Let j_0 be a particular value of j in the square-and-multiply algorithm of Figure 2.2, and let $g \in \{0, 1, 2, 3\}$. Marvin proceeds with the timing attack by filling in table columns with values of the form $T_i - \hat{T}_{i,j_0,g}$ where g is a guess for value of the exponent bits $d_{j_0}d_{j_0-1}$. Assuming that Marvin has correctly determined the value of the bits $d_{n-1}d_{n-2}\dots d_{j_0+2}d_{j_0+1}$, we have:

$$\hat{T}_{i,j_0,g} = c_i + \sum_{j>j_0}(r_{i,j} + s_{i,j} + t_{i,j}) + (r_{i,j_0} + s_{i,j_0} + \hat{t}_{i,j_0,g}),$$

where $\hat{t}_{i,j_0,g}$ is a candidate value for t_{i,j_0}. If $g = 0$ then $\hat{t}_{i,j_0,g} = 0$, otherwise $\hat{t}_{i,j_0,g} > 0$. Now, we have:

$$\begin{aligned} T_i - \hat{T}_{i,j_0,g} &= e_i + c_i + \sum_j (r_{i,j} + s_{i,j} + t_{i,j}) \\ &\quad - c_i - \sum_{j>j_0} (r_{i,j} + s_{i,j} + t_{i,j}) - (r_{i,j_0} + s_{i,j_0} + \hat{t}_{i,j_0,g}) \\ &= e_i + \sum_{j<j_0} (r_{i,j} + s_{i,j} + t_{i,j}) + (t_{i,j_0} - \hat{t}_{i,j_0,g}). \end{aligned}$$

Either $\hat{t}_{i,j_0,g}$ is a correct measure of the time it took Alice to calculate the multiplication at line 9 of Figure 2.2 when $j = j_0$ or it is not. If it is correct, then $\hat{t}_{i,j_0,g}$ equals t_{i,j_0}, and so:

$$T_i - \hat{T}_{i,j_0,g} = e_i + \sum_{j<j_0} (r_{i,j} + s_{i,j} + t_{i,j}).$$

If it is not correct, then $\hat{t}_{i,j_0,g}$ does not usually equal t_{i,j_0}, so there will be no cancellation. Marvin can use statistics to determine whether or not this cancellation occurs and hence check the guess g.

The subtraction of the term $\hat{t}_{i,j_0,g}$ affects the variance of a column of data. To see this, we treat the timing measurements as occurrences of random variables. The random variable T describes how long it takes to sign a message in $\mathbb{Z}_N$ using Alice's private exponent, d. The random variable $\hat{T}_{j_0,g}$ describes how long it takes to exponentiate a message using the $n - j_0$ most significant bits of d appended with a two bit guess (dependent on the value of g). The random variables r and s describe how long it takes to square an element of $\mathbb{Z}_N$. The random variable t

describes how long it takes to multiply two elements of $\mathbb{Z}_N$ (note that t is strictly positive). Lastly, the random variable e describes the effects of error.

Assuming the time for squares and multiplications in successive loop iterations are independent from each other and the error, the variance of the random variable $T - \hat{T}_{j_0,g}$, when the guess g is correct, is:

$$\begin{aligned}\mathrm{Var}(T - \hat{T}_{j_0,g}) &= \mathrm{Var}\left(e + \sum_{j<j_0}(r+s) + \sum_{\substack{j<j_0 \\ d_j d_{j-1} \neq 00}} t \right) \\ &= \mathrm{Var}(e) + \frac{j_0 - 2}{2}\mathrm{Var}(r) + \frac{j_0 - 2}{2}\mathrm{Var}(s) + \ell \cdot \mathrm{Var}(t).\end{aligned}$$

The variable ℓ is an integer which is determined by the number of pairs of bits from d which are not equal to 00 (for a random value of d, ℓ is roughly $\frac{3}{4}\frac{(j_0-2)}{2}$). Recall that the random variables r and s both describe the time it takes to do a squaring operation. Thus, r and s are identically distributed and the variance of $T - \hat{T}_{j_0,g}$ can be further simplified to:

$$\mathrm{Var}(T - \hat{T}_{j_0,g}) = \mathrm{Var}(e) + (j_0 - 2)\mathrm{Var}(s) + \ell \cdot \mathrm{Var}(t).$$

When the guess g is incorrect then there are two possibilities for the variance of $T - \hat{T}_{j_0,g}$, depending on the value of g. Recall that:

$$T_i - \hat{T}_{i,j_0,g} = e_i + \sum_{j<j_0}(r_{i,j} + s_{i,j} + t_{i,j}) + (t_{i,j_0} - \hat{t}_{i,j_0,g}).$$

First, suppose that both t_{i,j_0} and $\hat{t}_{i,j_0,g}$ are nonzero. Then, the value $t_{i,j_0} - \hat{t}_{i,j_0,g}$ is the difference of two (usually unequal) occurrences of the random variable t. The variance of the random variable $t - t$ is $\mathrm{Var}(t) + \mathrm{Var}(-t) = 2 \cdot \mathrm{Var}(t)$, thus for the relevant table column(s):

$$\mathrm{Var}(T - \hat{T}_{j_0,g}) = \mathrm{Var}(e) + (j_0 - 2)\mathrm{Var}(s) + (\ell + 2)\mathrm{Var}(t).$$

Next, suppose that one of t_{i,j_0} or $\hat{t}_{i,j_0,g}$ is zero. Then, for any column(s) of data with this property:

$$\mathrm{Var}(T - \hat{T}_{j_0,g}) = \mathrm{Var}(e) + (j_0 - 2)\mathrm{Var}(s) + (\ell + 1)\mathrm{Var}(t).$$

So, the column of data based on a correct guess has a variance which is $\mathrm{Var}(t)$ or $2 \cdot \mathrm{Var}(t)$ lower than the other data columns. The sample variance, S^2, is a good estimator of the true variance and we will now present a heuristic estimate of the probability that this statistic will distinguish the correct column.

To develop our estimate, we first introduce some notation and state two facts which are established in most introductory texts on probability (e.g., [8]). We write $X \sim N(\mu, \sigma^2)$ to indicate that the random variable X is normally distributed with mean μ and variance σ^2. The mean of a random variable X is also denoted by $E(X)$. If Y is a random variable with $Y = aX + b$, where a and b are constants, and $X \sim N(\mu, \sigma^2)$, then $Y \sim N(a\mu + b, a^2\sigma^2)$. If $X \sim N(\mu_X, \sigma_X^2)$ and $Y \sim N(\mu_Y, \sigma_Y^2)$, where X and Y are independent, then $X + Y \sim N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$.

The column of data in Marvin's table which corresponds to a correct guess has an expected variance of $\mathrm{Var}(e) + (j_0 - 2)\mathrm{Var}(s) + \ell \cdot \mathrm{Var}(t)$. There is a second column in

Marvin's table that has an expected variance of $\mathrm{Var}(e)+(j_0-2)\mathrm{Var}(s)+(\ell+1)\mathrm{Var}(t)$. These two variances differ by $\mathrm{Var}(t)$. Suppose there is a third column of data with expected variance $\mathrm{Var}(e)+(j_0-2)\mathrm{Var}(s)+(\ell+2)\mathrm{Var}(t)$. Its variance differs from the first column by $2\cdot\mathrm{Var}(t)$. The success probability of Marvin's statistical test, which consists of calculating S^2, is lower when he applies it to the first and second columns, as opposed to when he applies it to the first and third columns. We derive an estimate of this worst-case probability of success. An estimate of the probability in the other case can be derived similarly.

Suppose r, s and t are normally distributed. Let $N(\mu_s, \sigma_s^2)$ denote the distribution of r and s, and let $N(\mu_t, \sigma_t^2)$ denote the distribution of t. Both:

$$\sum_{j<j_0}(r+s) \quad \text{and} \quad \sum_{\substack{j<j_0 \\ d_j d_{j-1}\neq 00}} t$$

are normally distributed and the data in the correct and incorrect table columns are distributed according to the sums:

$$\sum_{j<j_0}(r+s) \;+ \sum_{\substack{j<j_0 \\ d_j d_{j-1}\neq 00}} t, \quad \text{and} \quad \sum_{j<j_0}(r+s) \;+ \sum_{\substack{j<j_0 \\ d_j d_{j-1}\neq 00}} t \;+\; t.$$

Both of these random variables are normally distributed. Denote the distribution of the former one by $N(\mu_0, \sigma_0^2)$. Note that, $\sigma_0^2 = (j_0-2)\sigma_s^2 + \ell\sigma_t^2$.

Suppose we have a total of k accurate timing measurements. Let $X_1, X_2, \ldots, X_k$ and $Y_1, Y_2, \ldots, Y_k$ be standard normal variates. If the effects of error are negligible, we can model the data in the two columns as:

$\sigma_0 X_1 + \mu_0$	$(\sigma_0 X_1 + \mu_0) + (\sigma_t Y_1 + \mu_t)$
$\sigma_0 X_2 + \mu_0$	$(\sigma_0 X_2 + \mu_0) + (\sigma_t Y_2 + \mu_t)$
$\vdots$	$\vdots$
$\sigma_0 X_k + \mu_0$	$(\sigma_0 X_k + \mu_0) + (\sigma_t Y_k + \mu_t)$

To simplify our notation, we let $V_i = \sigma_0 X_i + \mu_0$ and $W_i = (\sigma_0 X_i + \mu_0) + (\sigma_t Y_i + \mu_t)$. We want to estimate:

$$\begin{aligned}\Pr(S_W^2 > S_V^2) &= \Pr\left(\frac{1}{k-1}\sum_{i=1}^{k}(W_i - \overline{W})^2 > \frac{1}{k-1}\sum_{i=1}^{k}(V_i - \overline{V})^2\right)\\ &= \Pr\left(\sum_{i=1}^{k}(W_i - \overline{W})^2 > \sum_{i=1}^{k}(V_i - \overline{V})^2\right).\end{aligned}$$

The random variables V_i and W_i are normally distributed with means of μ_0 and $\mu_0 + \mu_t$, respectively. So, if k is large, then $\overline{V} \approx \mu_0$ and $\overline{W} \approx \mu_0 + \mu_t$. Using this approximation gives us:

$$\begin{aligned}\Pr(S_W^2 > S_V^2) &\approx \Pr\left(\sum_{i=1}^{k}(\sigma_0 X_i + \sigma_t Y_i)^2 > \sum_{i=1}^{k}(\sigma_0 X_i)^2\right)\\ &= \Pr\left(\sum_{i=1}^{k}(\sigma_0^2 X_i^2 + 2\sigma_0\sigma_t X_i Y_i + \sigma_t^2 Y_i^2) > \sum_{i=1}^{k}\sigma_0^2 X_i^2\right)\\ &= \Pr\left(2\sigma_0 \sum_{i=1}^{k} X_i Y_i + \sigma_t \sum_{i=1}^{k} Y_i^2 > 0\right)\end{aligned}$$

The identity $\mathrm{Var}(X) = E(X^2) - E(X)^2$, shows that $E(X_i^2) = 1$ and $E(Y_i^2) = 1$. Now, $E(\sum_{i=1}^{k} Y_i^2) = \sum_{i=1}^{k} E(Y_i^2) = k$, and we will use this value to approximate $\sum_{i=1}^{k} Y_i^2$. Since X_i and Y_i are independent, $E(X_i Y_i) = E(X_i)E(Y_i) = 0$. Also, $\mathrm{Var}(X_i Y_i) = E(X_i^2 Y_i^2) = E(X_i^2)E(Y_i^2) = 1$. Applying the central limit theorem,

$\sum_{i=1}^{k} X_i Y_i$ approximately follows a $N(0, k)$ distribution. If Z is a standard normal variate, Marvin's probability of success (in the worst-case) is roughly:

$$\begin{aligned}
\Pr(S_W^2 > S_V^2) &\approx \Pr\left(2\sigma_0(\sqrt{k}Z) + \sigma_t k > 0\right) \\
&= \Pr\left(Z > -\frac{\sigma_t}{\sigma_0}\frac{\sqrt{k}}{2}\right) \\
&= \varPhi\left(\frac{\sigma_t}{\sigma_0}\frac{\sqrt{k}}{2}\right),
\end{aligned}$$

where $\varPhi(z)$ is the area under the standard normal curve from $-\infty$ to z. By reapplying the steps of our approximation, we can estimate Marvin's probability of success in the alternate case as:

$$\varPhi\left(\frac{\sigma_t}{\sigma_0}\sqrt{\frac{k}{2}}\right).$$

Notice that, as expected, this probability is larger than the first case.

Recall that $\sigma_0^2 = (j_0 - 2)\sigma_s^2 + \ell\sigma_t^2$. Guessing that ℓ is $\frac{3}{4}\frac{(j_0-2)}{2}$, we have $\sigma_0^2 = \frac{(j_0-2)}{2}(2\sigma_s^2 + \frac{3}{4}\sigma_t^2)$. Now:

$$\frac{\sigma_t}{\sigma_0} = \sqrt{\frac{\sigma_t^2}{\frac{(j_0-2)}{2}(2\sigma_s^2 + \frac{3}{4}\sigma_t^2)}} = \sqrt{\frac{2}{(j_0-2)(2(\frac{\sigma_s}{\sigma_t})^2 + \frac{3}{4})}}.$$

Thus the probability of success, in each of the two cases, depends on the values of σ_s, σ_t, j_0 and k. As Marvin proceeds with the timing attack, j_0 ranges from $n-1$ to 1. As more bits of the secret exponent are recovered, j_0 decreases, and so the probability of success should increase. Also, with more timing measurements, k increases, so the probability of success should increase.

In the next section, we evaluate many of the assumptions made in this approximation using experimental data collected from a simulation of the timing attack.

2.3.3 Experimental Results

The instruction set of many PC processors includes a Read Time Stamp Counter (RDTSC) function. The time stamp counter is a 64-bit counter which is zeroed on power-up and is incremented with each CPU clock cycle. By reading this counter immediately before and after a particular task is executed on a PC, it is possible to determine the number of CPU cycles consumed by this task[5]. This number can then be converted into standard units of time (e.g., microseconds), according to the speed of the processor, but, for the purposes of the timing attack, this is not necessary.

To estimate the distributions of the time required for RSAREF modular squares, multiplications and exponentiations, we timed several of these operations, using the RDTSC function, as they were executed on a PC running MS-DOS®. The PC's processor was a 450-MHz Pentium II®. The modulus used throughout all of our experiments was fixed as the 512-bit sample prime, `PRIME1`, from RSAREF's Diffie-Hellman demonstration program.

Figure 2.3 displays the distribution of the time required to square random values of $\mathbb{Z}_N$. The data was collected by timing 10^6 squaring operations. Each of 10^6 values squared were drawn uniformly from $\mathbb{Z}_N$. The resulting distribution is approximately normal with $\mu_s = 2.7131 \times 10^5$ ticks and $\sigma_s = 1.4719 \times 10^3$ ticks. This supports

[5]For an example of how to call the RDTSC function using standard C, see [27].

the assumption in the previous section that the random variable s is normally distributed. Also, we see that $\mathrm{Var}(s) \approx (1.4719 \times 10^3)^2 = 2.1665 \times 10^6$.

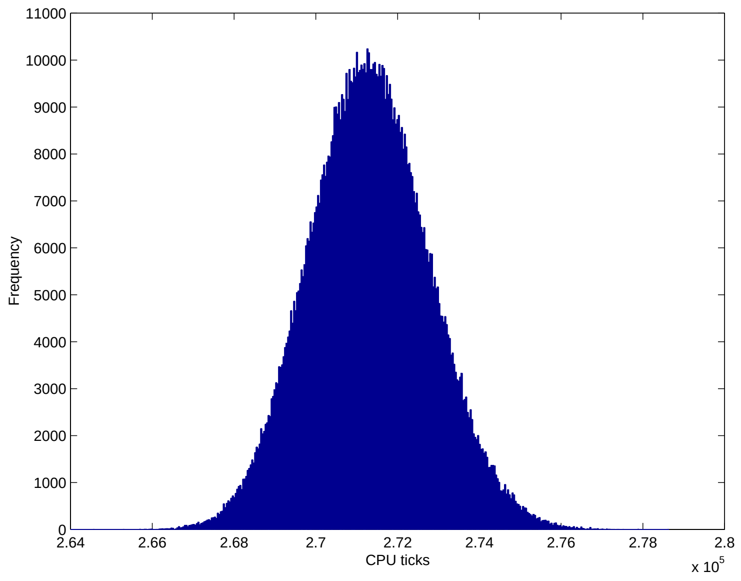

FIGURE 2.3: The timing distribution of one million modular squares.

Figure 2.4 displays the distribution of the time required to multiply random values of $\mathbb{Z}_N$. The data was collected by timing the multiplication of 10^6 pairs of values drawn uniformly from $\mathbb{Z}_N$. The resulting distribution is approximately normal with $\mu_t = 2.7119 \times 10^5$ ticks and $\sigma_t = 1.3186 \times 10^3$ ticks. Again, the distribution of the data supports the previous assumption that the random variable t is normally distributed. Also, $\mathrm{Var}(t) \approx (1.3186 \times 10^3)^2 = 1.7387 \times 10^6$.

It is interesting to note that although the function `NN-ModMult` is used by RSAREF to do both modular squares and multiplications, their respective timing distributions differ. The standard deviation of the multiplication times is lower than that of the squares. This is evidence that the value of the operands used in

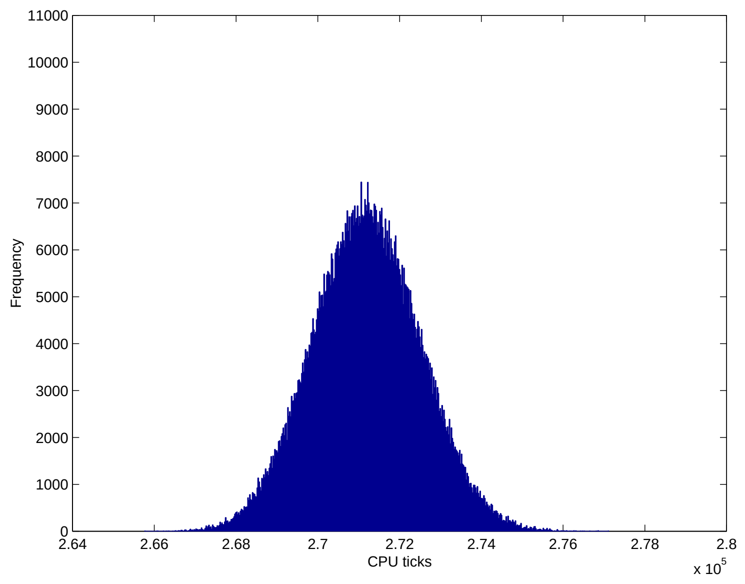

FIGURE 2.4: The timing distribution of one million modular multiplications.

the `NN-ModMult` function do indeed have an influence on the observed execution times.

If we ignore the effects of error, we can use the previous two distributions to predict the value of the parameters μ and σ in the distribution of modular exponentiation times, assuming that the length of the exponent is known. Consider a 64-bit exponent drawn uniformly from the space of all such exponents. On average, 24 of the 32 pairs of bits in this exponent will be nonzero. Thus, when an element of $\mathbb{Z}_N$ is exponentiated with this exponent, we expect 24 conditional multiplications to be performed. The number of squaring operations performed is exactly 64 since two squares are calculated for each pair of bits. Using the linearity of expected values and variances, we predict that:

$$\mu \approx 64 \cdot 2.7131 \times 10^5 + 24 \cdot 2.7119 \times 10^5 = 2.3872 \times 10^7 \text{ ticks}$$
$$\sigma \approx \sqrt{64 \cdot 2.1665 \times 10^6 + 24 \cdot 1.7387 \times 10^6} = 1.3431 \times 10^4 \text{ ticks.}$$

Figure 2.5 displays the timing distribution of 10^5 modular exponentiations using a fixed 64-bit exponent. The values exponentiated were drawn uniformly from $\mathbb{Z}_N$ and the value of the fixed exponent was `0xFEDCBA9876543210` (i.e., the 16 hexadecimal[6] digits written in decreasing order). Exactly 24 of 32 pairs of bits in this exponent are nonzero, so we expect the predicted values above to be quite close to the observed ones. The distribution is approximately normal with $\mu = 2.3685 \times 10^7$ ticks and $\sigma = 1.5026 \times 10^4$ ticks. The observed value of σ is larger than our prediction and this may be caused by the effects of measurement error.

To demonstrate that a comparison of sample variances is a valid method for distinguishing bits of a secret exponent, we conducted two experiments, each one targeting the toy exponent $d =$ `0xFEDCBA9876543210`. In the first experiment, we measured the time it took to exponentiate 100 values, $M_1, M_2, \ldots, M_{100}$, drawn uniformly from $\mathbb{Z}_N$. Using the resulting timings, $T_1, T_2, \ldots, T_{100}$, we attempted to deduce every second pair of exponent bits.

For example, the first pair of bits considered were $d_{61}d_{60}$; the bits $d_{63}d_{62}$ were skipped since we considered only every second pair of bits. To determine $d_{61}d_{60}$, the messages $M_1, M_2, \ldots, M_{100}$ were exponentiated using the four exponents `0xC`, `0xD`, `0xE`, and `0xF`. This resulted in four sequences of timing measurements which

[6]Values written in hexadecimal are prefixed with `0x`.

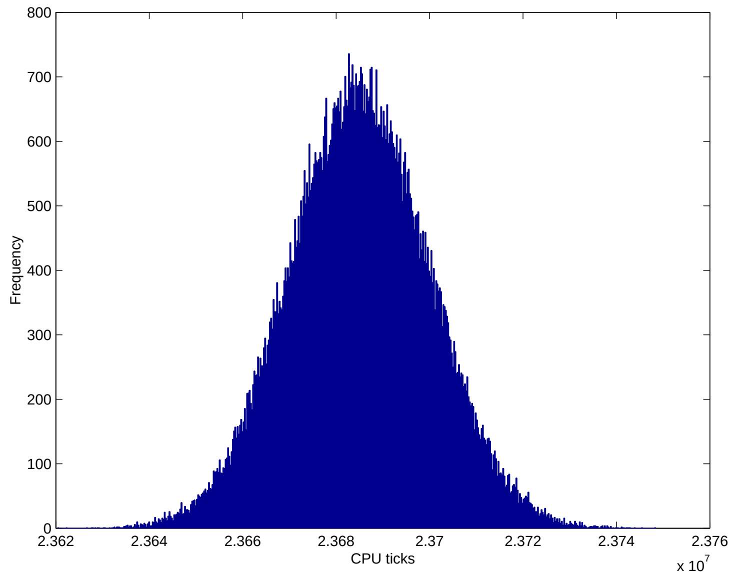

FIGURE 2.5: The timing distribution of ten thousand modular exponentiations.

were respectively subtracted from the sequence $T_1, T_2, \ldots, T_{100}$. The exponent that produced the data set with the lowest sample variance was then determined and compared to the actual value of the bits $d_{61}d_{60}$. If the two values coincide, then the attack was successful. The next pair of bits considered were $d_{57}d_{56}$ Again, four sequences of timing measurements were generated, this time using the exponents `0xFC`, `0xFD`, `0xFE`, and `0xFF`, and the resulting sample variances were compared. For subsequent pairs of bits, the experiment proceeded in a similar manner.

We can use the approximation in the previous section to estimate the probability of successfully determining the bits $d_{61}d_{60}$ using 100 timing measurements. This event will occur only if the sample variance of the correct data set is lower than that of *all three* of the other data sets. If we identify the four data sets with their corresponding bit guesses, we see that the expected variance of data set 11 (the correct guess) differs from that of the data sets 00, 01 and 10 by $\mathrm{Var}(t)$, $2 \cdot \mathrm{Var}(t)$ and $2 \cdot \mathrm{Var}(t)$, respectively. So, naively, we might estimate the probability of success as:

$$\varPhi\left(\frac{\sigma_t}{\sigma_0}\frac{\sqrt{k}}{2}\right)\varPhi\left(\frac{\sigma_t}{\sigma_0}\sqrt{\frac{k}{2}}\right)^2.$$

However, this estimate assumes that the sample variance of the incorrect data sets are independent of each other and this is not the case. If the sample variance of data set 00 is large then it is likely that the sample variance of data sets 01 and 10 will be large as well. Without digressing into any further statistical analysis, we will simply treat the product above as a lower bound on the probability of success.

Making the relevant substitutions, we have:

$$\begin{aligned}\frac{\sigma_t}{\sigma_0} &= \sqrt{\frac{2}{(j_0-2)(2(\frac{\sigma_s}{\sigma_t})^2+\frac{3}{4})}}\\ &\approx \sqrt{\frac{2}{(61-2)(2(\frac{1.4719}{1.3186})^2+\frac{3}{4})}}\\ &\approx 0.1023,\end{aligned}$$

and so:

$$\varPhi\left(0.1023 \cdot \frac{\sqrt{100}}{2}\right)\varPhi\left(0.1023 \cdot \sqrt{\frac{100}{2}}\right)^2 \approx 0.6954 \cdot 0.7652^2 \approx 0.41.$$

When considering a pair of bits that have an actual value of 00, the approximate probability of success is calculated differently. For example, the first pair of bits our experiment considers with this value is $d_{49}d_{48}$. In this case, the expected variance of data set 00 (the correct guess) differs from each of the data sets 01, 10 and 11 by Var(t). Thus, a lower bound on the probability of success is $\mathit{\Phi}\left(\frac{\sigma_t}{\sigma_0}\frac{\sqrt{k}}{2}\right)^3$. Substituting $j_0 = 49$ into the expression for $\frac{\sigma_t}{\sigma_0}$ we have:

$$\mathit{\Phi}\left(\frac{\sigma_t}{\sigma_0}\frac{\sqrt{k}}{2}\right)^3 \approx \mathit{\Phi}\left(0.1146 \cdot \frac{\sqrt{100}}{2}\right)^3 \approx 0.37.$$

The experiment was repeated 25 times, using new values of $T_1, T_2, \ldots, T_{100}$ in each iteration. For each pair of bits, the observed probability of success was calculated and compared to the estimated probability of success. The results are summarized in Figure 2.6.

bits	observed	estimated	bits	observed	estimated
F	0.44	0.41	7	0.44	0.57
E	0.68	0.42	6	0.56	0.61
D	0.64	0.43	5	0.72	0.66
C	*0.12*	*0.37*	4	*0.32*	*0.60*
B	0.44	0.47	3	0.84	0.80
A	0.56	0.49	2	0.88	0.90
9	0.88	0.51	1	1	0.99
8	*0.08*	*0.44*	0	*1*	–

FIGURE 2.6: Result of the timing attack with 100 timings.

The table of Figure 2.6 identifies every second pair of exponent bits with their relevant hex digit. The first entry indicates that $0.44 \times 25 = 11$ of the 25 trials to determine the bits $d_{61}d_{60}$ were successful, which is slightly better than our estimate of $0.41 \times 25 \approx 10$. As expected, the observed probability of success increases, in the two respective cases, as the attack progresses. Overall, the observed probability of success was $\frac{240}{25 \cdot 16} \approx 0.60$.

Some of the experimental data deviates quite drastically from our estimates. Most noticeably, the estimated probability of success in the case when the correct bits are 00 is significantly higher than the observed probability. One possible reason might be that the variance of the error in the timing measurements, $\mathrm{Var}(e)$, is non-negligible. For example, $\mathrm{Var}(e)$ and $\mathrm{Var}(t)$ might be close to the same value.

The second experiment followed the same design as the first one except that 1000 timing measurements were used rather than 100. With this number, the estimated probability of determining $d_{61}d_{60}$ is:

$$\Phi\left(0.1023 \cdot \frac{\sqrt{1000}}{2}\right)\Phi\left(0.1023\sqrt{\frac{1000}{2}}\right)^2 \approx 0.93,$$

and, the estimated probability of determining $d_{49}d_{48}$ is:

$$\Phi\left(0.1146\sqrt{\frac{1000}{2}}\right)^3 \approx 0.90.$$

The results of the second experiment are displayed in Figure 2.7.

bits	observed	estimated	bits	observed	estimated
`F`	0.88	0.93	`7`	0.80	0.99
`E`	0.88	0.94	`6`	0.96	0.99
`D`	0.96	0.95	`5`	1	0.99
`C`	*0*	*0.90*	`4`	*0.36*	*0.99*
`B`	0.60	0.96	`3`	1	0.99
`A`	0.96	0.97	`2`	1	0.99
`9`	1	0.98	`1`	1	0.99
`8`	*0*	*0.96*	`0`	*1*	–

FIGURE 2.7: Result of the timing attack with 1000 timings.

As expected, nearly all of the observed probabilities have increased from their values in the first experiment. However, the probabilities in the case when the correct bits are 00 deviates even further from the estimated values. More experimental data is required to investigate this aberrant behaviour. Overall, the observed probability of success was $\frac{310}{25 \cdot 16} \approx 0.78$.

The purpose of using a small exponent size in our experiments was to simplify our explanation, however, Kocher's timing attack has successfully been implemented by other researchers who have targeted exponents of practical sizes [18].

2.3.4 An Improvement

In our description of the timing attack, four exponents, each one representing a guess for a pair of bits, are used to generate four sets of timing data. The exponents `0xC`, `0xD`, `0xE` and `0xF`, were used to determine the value of the third and fourth most significant bits of $d =$ `0xFEDCBA9876543210`, in the experiment described in the previous section. According to our analysis, the expected variances of the four resulting sets of timing data are:

`0xC`	$\text{Var}(e) + (j_0 - 2)\text{Var}(s) + (\ell + 1)\text{Var}(t)$
`0xD`	$\text{Var}(e) + (j_0 - 2)\text{Var}(s) + (\ell + 2)\text{Var}(t)$
`0xE`	$\text{Var}(e) + (j_0 - 2)\text{Var}(s) + (\ell + 2)\text{Var}(t)$
`0xF`	$\text{Var}(e) + (j_0 - 2)\text{Var}(s) + \ell \cdot \text{Var}(t)$

Appending the pair of bits 00 to each of these four exponents can exaggerate the difference between the expected variance of these data sets.

Suppose that four new sets of timing data are generated using the exponents `0x30`, `0x34`, `0x38` and `0x3C`. In binary, the exponent `0x30` is 110000 (leading zeroes removed) which is just the value `0xC` concatenated with 00; similarly, for the other three exponents. The new timing data will differ from the previous data because the appended pair of bits causes two additional squaring operations to be performed. If the third and fourth most significant bits of these exponents do not agree with the bits of d, the addition two squaring operations increase the variance of the data set by $2 \cdot \mathrm{Var}(s)$. Alternately, if these bits do agree with the bits of d, the variance of the data set will decrease by the same amount. The respective variances are now:

`0x30`	$\mathrm{Var}(e) + j_0 \cdot \mathrm{Var}(s) + (\ell + 1)\mathrm{Var}(t)$
`0x34`	$\mathrm{Var}(e) + j_0 \cdot \mathrm{Var}(s) + (\ell + 2)\mathrm{Var}(t)$
`0x38`	$\mathrm{Var}(e) + j_0 \cdot \mathrm{Var}(s) + (\ell + 2)\mathrm{Var}(t)$
`0x3C`	$\mathrm{Var}(e) + (j_0 - 4)\mathrm{Var}(s) + \ell \cdot \mathrm{Var}(t)$

Using this technique, the correct guess for the pair of exponent bits is more likely to be distinguished by the sample variance of its resulting data set.

2.4 Other Vulnerable Systems

The timing attack can be tailored against virtually any operation which takes a variable amount of time. The algebraic operations used in public key systems and signature schemes such as ECC, RSA and ElGamal often run in non-constant time. Block ciphers such as Rijndael and IDEA are also at risk since they use multiplication in their encryption processes [33, 29]. The bit rotations used in RC5 and DES can leak the Hamming weight of their operands if these operations are implemented using a shift and conditional bit “wrap around” [24, 28].

Cryptographic engineers must pay careful attention to the influence of key values on the timing characteristics of table-lookups, bit shifts/rotations, addition, subtraction and multiplication operations to access the vulnerability of specific implementations to timing attacks.

2.5 Countermeasures

Before describing how to defeat the timing attack, we will first consider two other common approaches towards developing countermeasures.

The first and most obvious method is to ensure all operations run in a constant amount of time. Unfortunately, it is difficult to achieve this goal. Compiler optimizations and memory look-ups can introduce unexpected timing variations which are beyond the control of implementors. Withholding the result of an operation until a specified amount of time has expired may seem a promising approach, but the length of the added delay may be conveyed through the system’s power consumption or CPU usage. Using this method would also degrade system efficiency since all operations would behave as if they were processing worst-case inputs.

Unconditionally performing the multiplication in each loop iteration of a square-and-multiply algorithm (see figure 2.8) does not make the execution time of the algorithm constant. Variability in the multiplication and squaring operations will still remain and this can be exploited. As we emphasized earlier, the timing attack can determine what operands were used in each step of the algorithm as well as the

```
INPUT: $M, N, d = (d_{n-1}d_{n-2}\dots d_1d_0)_2$
OUTPUT: $S = M^d \mod N$
1 $S \leftarrow 1$
2 for $j = n-1 \dots 0$ do
3     $S \leftarrow S^2 \mod N$
4     $T \leftarrow S \cdot M \mod N$
5     if $d_j = 1$ then
6         $S \leftarrow T$
7 return $S$
```

FIGURE 2.8: This modification of the square-and-multiply algorithm is still vulnerable to the timing attack.

path of execution.

If the multiplication and squaring operations ran in constant time, then the time for a modular exponentiation would only be correlated to the Hamming weight of the exponent. For random exponents, the Hamming weight does not, on average, reveal much information about its value. Montgomery multiplication runs in almost constant time, but there is a small source of variability resulting from a conditional subtraction. RSA with Montgomery multiplication is vulnerable to the timing attack, as is shown in [18].

The second method is to add noise to the execution time of operations. The intended effect is to increase the required number of timing measurements so that the attack becomes infeasible. Our method of attack and subsequent analysis assumed the effects of noise were negligible, but this may not be the case. Inserting random delays in operations provides a source of noise, but this will reduce efficiency if the mean of the delay is large. For a successful timing attack, the required number of timing measurements roughly increases linearly as a function of the variance of the random delay.

To defeat the timing attack, implementors should prevent an attacker from learning the inputs to a vulnerable operation. In the case of RSA, if Marvin does not know the value of the base used in a modular exponentiation, then the corresponding timing information is of no use. The algebraic structure of $\mathbb{Z}_N$ allows messages to be *blinded* [13] before they are signed. Rather than sign the message $M \in \mathbb{Z}_N^*$ Alice can pick a random $r \in \mathbb{Z}_N^*$ and sign the message $M' = r^e \cdot M \mod N$ instead. Denote the resulting signature by S'. Alice now calculates $r^{-1}S' = r^{-1}r^{ed}M^d = r^{-1}rM^d = M^d \mod N$ to obtain her signature on the message M. The suitability of this technique depends entirely on the details of the cryptosystem, but many public key systems have the required algebraic structure.

2.6 Remarks

Our analysis of the timing attack, as it is applied to modular exponentiation in RSAREF, is complicated by the fact that the exponentiation method there processes exponents using a 2-bit window. Our discussion could be greatly simplified if a method using a one bit window was considered. In [30], Kocher simplifies his analysis by assuming that every second bit of the exponent is known.

Kocher presents results from several experiments in [30] which support his theoretical description of the timing attack. Unfortunately, in that publication, Kocher reveals few practical details of how he actually performed his experiments; this makes the task of reproducing his experiments somewhat difficult for a reader. Other authors have been more forthcoming with the details of their experiments. For example, there is a detailed discussion in [28] which describes how precise timing information (e.g., microseconds or better) can be measured on a PC. In our own experiments, we found Heidenstrom's document "Timing on the PC family under DOS" [27] to be an excellent source of information.

It should be noted that Kocher's timing attack, as presented in [30] and summarized in the previous sections, does not directly apply to the RSA signature operation in RSAREF 2.0. Like many implementations of RSA, RSAREF 2.0 uses the Chinese Remainder Theorem (CRT) to calculate signatures[7]. A consequence of this method is that the inputs to the two component modular exponentiations are effectively blinded, so the timing attack can not be applied. If an adversary has the ability to choose which messages are signed then the timing attack can be applied to CRT implementations as shown in [47].

Timing attacks are more threatening to dedicated cryptographic devices (e.g., smartcards) than they are to multitasking devices like PCs. Unless a PC is operating in some controlled mode where interrupts are disabled, isolating the time required by a single computation can be difficult. Usually, computations are continually interrupted as the operating system makes routine system calls (e.g., updating the system clock). These interruptions can introduce a large amount of error in timing measurements.

[7]More details about RSA with CRT can be found in Chapter 3.

Chapter 3

Fault Analysis

3.1 Introduction

Participating in a cryptographic protocol is a relatively painless process these days; usually, any required computation or transmission is quickly done with digital hardware (e.g., PC, smartcard, cellular phone). Most of these devices seem to operate reliably when we use them so we might not think to question if the security of a protocol depends on the reliability of the device which implements it.

Hardware faults and errors which occur during the operation of a cryptographic module can affect security. For example, a device might transmit ciphertext or plaintext according to the value of a single register bit. If that bit was flipped accidentally by, say, a power surge, then subsequent transmissions would be unintentionally sent in the clear. In this case, the fault changed the operational mode of the module, and had no influence on the strength of any underlying cryptography. Engineering criteria have been developed to ensure cryptographic modules operate correctly in the presence of faults like this one [21]. However, until the mid 1990s it was not clear that cryptographers had to worry that faults might increase a cipher's vulnerability to cryptanalysis.

On 25 September 1996, Boneh, DeMillo and Lipton announced that the occurrence of computational faults can have severe consequences to the strength of cryptographic schemes [36]. In an extreme example, these researchers demonstrated that a *single* erroneous RSA signature can compromise a signer's private key. Their results were particularly relevant to the design of smartcard systems since the small size and intended use of these devices provide an adversary with the opportunity to *induce* faults and cause erroneous output[1]. This discovery received widespread attention and prompted research into the effects of faults in other cryptosystems.

Outline

The first part of this chapter explains two techniques of fault analysis that can be used to break RSA implementations. Both attacks exploit computational errors that occur during an RSA signature operation. The second part of the chapter explains how fault analysis can also be applied to symmetric ciphers; DES, in particular. A number of attacks are presented, the first of which assumes that an adversary is able to obtain two DES encryptions of the same message: a faulty one, and a valid one. Other attacks are suggested which are less restrictive in terms of what ciphertexts are useful to an adversary, but make the assumption that the internal circuitry of the target implementation can be manipulated by

[1]A malicious user may try to induce fault in his or her smartcard by, say, bombarding it with radiation or putting it in a microwave. Inducing faults on a remote PC seems to be more difficult.

the adversary. To end, countermeasures against all of the mentioned attacks are discussed.

3.2 RSA Vulnerabilities

Modular arithmetic is a fundamental component of many cryptographic schemes. One consequence of this fact is that these schemes inherit mathematical properties such as associativity, commutativity and transitivity which may be exploited by both system designers and attackers. In the case of RSA, modular arithmetic allows an adversary to carefully examine the effect faults which occur in a signature operation.

3.2.1 RSA with CRT

The Chinese Remainder Theorem (CRT) can be used to speed up RSA signature generation. Suppose Alice wishes to sign a message $M \in \mathbb{Z}_N$ where N is the product of the primes p and q. Rather than calculate the value $S = M^d \mod N$ directly, she uses the factors of N and computes:

$$S_p = M^{d_p} \mod p \qquad \text{and} \qquad S_q = M^{d_q} \mod q$$

where $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$. She then computes S to be the linear combination $u_p S_p + u_q S_q \mod N$ where:

$$u_p = \begin{cases} 1 & \mod p \\ 0 & \mod q \end{cases} \qquad \text{and} \qquad u_q = \begin{cases} 0 & \mod p \\ 1 & \mod q \end{cases}$$

The values d_p, d_q, u_p, u_q can be pre-computed, and the time required to calculate a linear combination of S_p and S_q is negligible compared to the two component exponentiations.

The speed-up in using the CRT comes from the fact that doing two exponentiations with moduli half the size of N is quicker than doing one exponentiation modulo N. If n is the bit length of N then calculating $M^d \mod N$ using a square-and-multiply algorithm takes time proportional to n^3. The factors of N have bit length $\frac{n}{2}$, so an exponentiation modulo p or q takes time proportional to $(\frac{n}{2})^3 = \frac{n^3}{8}$. Thus, obtaining S from S_p and S_q is $n^3/(2\frac{n^3}{8}) = 4$ times faster than direct exponentiation. For this reason, many RSA implementations use the CRT for signature generation including RSAREF 2.0 which was presented in the previous chapter [44].

Boneh, DeMillo and Lipton observed that if exactly one of the values S_p or S_q is computed incorrectly, then an adversary who has two signatures on the same message, one correct and the other faulty, can factor N. Based on this result, Lenstra noted that knowledge of only the faulty signature is sufficient to factor N. We summarize his technique now.

Suppose an error occurs during the calculation of $M^{d_q} \mod q$, resulting in the value $\hat{S}_q \neq M^{d_q} \mod q$. The resulting signature $\langle M, \hat{S} \rangle$ will be invalid. Consider the value $M - \hat{S}^e$. We have:

$$\begin{aligned} & M - \hat{S}^e \mod p \\ = \ & M - {S_p}^e \mod p \\ = \ & 0 \mod p \end{aligned} \qquad \text{and} \qquad \begin{aligned} & M - \hat{S}^e \mod q \\ = \ & M - \hat{S_q}^e \mod q \\ \neq \ & 0 \mod q \end{aligned}$$

Thus p is a factor of $M - \hat{S}^e$ and q is not. So, an adversary merely needs to calculate $\gcd(N, M - \hat{S}^e) = p$ in order to factor N. With additional access to the correct signature $\langle M, S \rangle$ the adversary could instead calculate $\gcd(N, S - \hat{S}) = p$, as Boneh et al. originally suggested.

This attack does not assume anything about the nature of the error that occurred during the calculation of $M^{d_q} \mod q$. It makes no difference if the miscalculation was the result of a single hardware fault, multiple ones, or even a software bug. For this reason, this method of fault analysis is the most general of the ones we consider in this chapter.

3.2.2 Other Implementations

Not all implementations of RSA use the CRT. However, analyzing these systems under a more restrictive fault model can still lead to some interesting attacks. We now describe a variation of an attack presented in [10] which exploits register faults that occur during modular exponentiation.

Suppose that a non-CRT implementation of RSA uses the right-to-left repeated square-and-multiply algorithm to do modular exponentiation. Figure 3.1 describes such an algorithm where the output, S, can be thought of as an RSA signature. This

```
INPUT: $M, N, d = (d_{n-1}d_{n-2}\dots d_1d_0)_2$
OUTPUT: $S = M^d \mod N$

1 $z \leftarrow M$

2 $S \leftarrow 1$

3 for $j = 0 \dots n-1$ do

* register fault: $z \leftarrow z \pm 2^w$

4    if $d_j = 1$ then

5        $S \leftarrow S \cdot z \mod N$

6    $z \leftarrow z^2 \mod N$

7 return $S$
```

FIGURE 3.1: A modification of the right-to-left repeated square-and-multiply algorithm which models register faults.

algorithm requires at least two data registers to store the intermediate values of the variables z and S. The variable z is used to store the values $M, M^2, M^{2^2}, \dots, M^{2^{n-1}}$ as well as the superfluous value M^{2^n}. A subset of these values is used to form a product which equals $M^d \mod N$, the signature on the message M. A fault in the register which contains the variable z can corrupt the factors used in this product and therefore cause an invalid signature. Assuming that register faults flip individual bits of z, we show that an adversary with access to a number of erroneous signatures resulting from single faults can efficiently deduce the value of d.

The attack works by recovering blocks of bits from the binary representation of d, starting with the most significant bits. To illustrate the technique, suppose that during the signing of the message M a single register fault, denoted in Figure 3.1

at line $*$, occurs when $j = n - 2$. This error propagates and corrupts two of the intermediate values of z. Listing the values of z we have:

$$M, M^2, \ldots, M^{2^{n-3}}, \widetilde{M}, \widetilde{M}^2,$$

where $\widetilde{M} = M^{2^{n-2}} \pm 2^w$ for some w. The value $\widetilde{M}$ is the result of a fault in the z register when $z = M^{2^{n-2}}$. This fault caused the bit in position w of $M^{2^{n-2}}$ to be flipped. Denote the resulting erroneous signature as $\langle M, \hat{S} \rangle$. Now we have:

$$\hat{S} = M^{\sum_{i=0}^{n-3} d_i 2^i} \widetilde{M}^{\sum_{i=n-2}^{n-1} d_i 2^{i-(n-2)}} \mod N.$$

Using binary notation in the exponents, this can be written more simply as:

$$\hat{S} = M^{d_{n-3} \ldots d_0} \widetilde{M}^{d_{n-1} d_{n-2}} \mod N.$$

With the public exponent e, we can derive the following equivalences modulo N:

$$\begin{aligned}
(M^{2^{n-2}})^{d_{n-1}d_{n-2}} \hat{S} &\equiv (M^{2^{n-2}})^{d_{n-1}d_{n-2}} M^{d_{n-3}\ldots d_0} \widetilde{M}^{d_{n-1}d_{n-2}} \pmod N \\
&\equiv M^{d_{n-1}d_{n-2}d_{n-3}\ldots d_0} \widetilde{M}^{d_{n-1}d_{n-2}} \pmod N \\
(M^{e2^{n-2}})^{d_{n-1}d_{n-2}} \hat{S}^e &\equiv M^{ed} (\widetilde{M}^e)^{d_{n-1}d_{n-2}} \pmod N \\
&\equiv M (\widetilde{M}^e)^{d_{n-1}d_{n-2}} \pmod N \\
&\equiv M (M^{2^{n-2}} \pm 2^w)^{e \cdot (d_{n-1}d_{n-2})} \pmod N.
\end{aligned}$$

So, with knowledge of $\langle \hat{S}, M \rangle$ and the fact that $\widetilde{M} = M^{2^{n-2}} \pm 2^w$ for some w, an adversary can exhaust the possible values of w, d_{n-1}, d_{n-2} until the condition above holds, thereby revealing 2 bits of d. Since $\langle\hat{S}, M\rangle$ is erroneous, $\hat{S}^e \not\equiv M \mod N$ and therefore $d_{n-1}d_{n-2} \neq 00$. Thus, there are 3 values for the pair of bits $d_{n-1}d_{n-2}$ to consider. If n is the bit length of N then there are n possible values of w to consider. So it takes at most $(2^2 - 1)n = 3n$ trials to find the correct values of w, d_{n-1} and d_{n-2}.

In practice, an adversary does not know the value of j when the register fault at line $*$ occurred. It is possible to identify the correct value, call it j^*, using the following generalized equivalence:

$$(M^{e2^j})^{d_{n-1}d_{n-2}\ldots d_j}\hat{S}^e \equiv M(M^{2^j} \pm 2^w)^{e\cdot(d_{n-1}d_{n-2}\ldots d_j)} \pmod N. \tag{3.1}$$

Consider the following example. Suppose we verify the signature $\langle S, M\rangle = \langle 5066, 42\rangle$ against the RSA parameters $e = 3, N = 101 \cdot 113$ and determine that it is erroneous. According to the bit length of N, $n = 14$, and substituting this and the other values into equivalence 3.1 gives

$$(42^{3\cdot 2^j})^{d_{13}d_{12}\ldots d_j}5066^3 \equiv 42(42^{2^j} \pm 2^w)^{3\cdot(d_{13}d_{12}\ldots d_j)} \pmod 1)1413,$$

for some values of j, w and bits of d. After some trial and error we find that

$$(42^{3\cdot 2^{11}})^{(011)_2}5066^3 \equiv 42(42^{2^{11}} + 2^0)^{3\cdot(011)_2} \pmod 1)1413,$$

and so, for this erroneous signature, we have $j^* = 11$, but more importantly, we have learned that the two most significant bits of d are 11.

For a particular value of j, equivalence 3.1 allows an attacker to identify any erroneous signature with $j^* = j$ at a computational cost of $O((2^{n-j^*} - 1)n)$. Identifying the correct value of j^* also reveals the value of the bits $d_{n-1}d_{n-2}\dots d_{j^*}$. Knowledge of these bits reduces the effort required to identify j^* for other erroneous signatures since there are now fewer unknown bits of d. An adversary with access to a number of erroneous signatures, with possibly different values of j^*, can exploit this property using the method of attack described in Figure 3.2.

```
INPUT: $e, n, N,\ \langle M_0, S_0\rangle,\ \hat{\mathcal{S}} = \{\ \langle M_1, \hat{S}_1\rangle, \langle M_2, \hat{S}_2\rangle, \dots, \langle M_k, \hat{S}_k\rangle\ \}$
OUTPUT: $d = (d_{n-1}d_{n-2}\dots d_1 d_0)_2$

1 for $j = n-1 \dots 0$ do
2   for each $\langle M_i, \hat{S}_i\rangle \in \hat{\mathcal{S}}$ do
3     if $j_i^* = j$ then
4       update known bits of $d$
5       $\hat{\mathcal{S}} \leftarrow \hat{\mathcal{S}} - \langle M_i, \hat{S}_i\rangle$
6 solve $S_0 = M_0^d \bmod N$ for the unknown bits of $d$
7 return $d$
```

FIGURE 3.2: A randomized algorithm to deduce the value of d from k invalid signatures and one valid signature.

Throughout this attack an adversary manages a set of invalid signatures, $\hat{\mathcal{S}}$. For each value of j the set $\hat{\mathcal{S}}$ is scanned to identify any signature $\langle M_i, \hat{S}_i\rangle$ with $j_i^* = j$. The value j_i^* is the value of j in the square-and-multiply algorithm when the fault that generated $\langle M_i, \hat{S}_i\rangle$ occurred. The first identification[2] made at a particular value of j reveals some bits of d. Subsequent identifications at the same value

[2]We ignore the possibility of false identifications since Boneh et al. show in [10] that if this probability is non-negligible then N can be efficiently factored.

of j can be done quickly using the updated value of d. These identifications do not contribute any previously unknown bits of d, but discarding these signatures ensures that effort is not wasted on them in succeeding loop iterations. Most of the work involved in the attack is spent checking the condition $j_i^* = j$ at line 3. This condition is checked via equation 3.1. Once the set $\hat{\mathcal{S}}$ is exhausted any remaining unknown bits of d are deduced by solving $S_0 = M_0^d \mod N$ where $\langle M_0, S_0 \rangle$ is a valid signature.

We now give a heuristic analysis of the expected running time of the attack. For any i, the value j_i^* lies in the interval $[n-1, \ldots, 0]$. The values $j_1^*, j_2^*, \ldots, j_k^*$ can be ordered and, if necessary, relabeled so that $j_1^* \le j_2^* \le \ldots \le j_k^*$. The first identification made at line 3 of Figure 3.2 recovers $n - j_k^*$ bits of d. The second identification recovers an additional $j_k^* - j_{k-1}^*$ bits, and so on for subsequent identifications. Thus, the running time of the attack is proportional to:

$$\sum_{l=1}^{n-j_k^*} k \cdot (2^l - 1) \cdot n + \sum_{l=1}^{j_k^*-j_{k-1}^*} (k-1) \cdot (2^l - 1) \cdot n + \cdots + \sum_{l=1}^{j_1^*} 1 \cdot (2^l - 1) \cdot n$$
$$\le k \cdot n \left[\sum_{l=1}^{n-j_k^*} 2^l + \sum_{l=1}^{j_k^*-j_{k-1}^*} 2^l + \cdots + \sum_{l=1}^{j_1^*} 2^l \right]$$

Assuming the j_i^* follow a uniform distribution, the probability that none of these values hit a particular interval of width r is $(1 - \frac{r}{n})^k \approx e^{-\frac{r}{n}k}$. Since there are at most n such intervals, the probability that all of them contain a hit is at least $1 - ne^{-\frac{r}{n}k}$. Taking $r = \frac{n}{k} \ln 2n$, we see that this event occurs with probability at least $\frac{1}{2}$. Thus, with probability at least $\frac{1}{2}$, the differences $j_i^* - j_{i-1}^*$ are bounded by

r. So, the expected running time is:

$$O\left(k \cdot n \left[k \sum_{l=1}^{r} 2^l\right]\right) = O\left(k^2 n\ 2^{r+1}\right) = O\left(k^2 n\ 2^{\frac{n}{k}\ln 2n}\right)$$

With $k = n \ln n$ erroneous signatures, the attack takes $O(n^3 \ln^2 n)$ time.

One possible improvement to this method is to check j_i^* against two values at each invocation of line 3 in Figure 3.2. Equivalence 3.1 is used to check the condition $j_i^* = j$, and the following equivalence can be used to check if $j_i^* = n - j$:

$$(\hat{S}^e)^{2^{n-j}} (\widetilde{M}^e)^{d_{(n-j)-1} d_{(n-j)-2} \ldots d_0} \equiv (M^{e2^{n-j}})^{d_{(n-j)-1} d_{(n-j)-2} \ldots d_0} \widetilde{M} \mod N \qquad (3.2)$$

This modification allows blocks of bits to be recovered from both the left and right ends of d, so the set $\hat{\mathcal{S}}$ is exhausted more quickly. There is no advantage in terms of the number of invalid signatures required, however if the value of j_i^* is controlled by the attacker rather than being uniformly distributed over $[n-1, \ldots, 0]$ then the required number of signatures is reduced significantly. An attacker could effectively divide d into, say, 10 bit blocks and recover each one by brute force. This capability might be possible in a more intrusive fault model.

3.3 DES Vulnerabilities

After reviewing the discoveries of Boneh, DeMillo and Lipton, one might consider whether fault analysis can be applied to cryptosystems which do not utilize modular arithmetic. Typically, symmetric ciphers use bit or byte oriented operations (e.g., AND, XOR, ROTATES) and so the techniques previously discussed are not directly applicable.

Biham and Shamir quickly answered this point in [6]. They showed that an implementation of the Data Encryption Standard (DES) could be easily broken if it was subject to the same random register faults that Boneh et al. considered. Their method of attack combined techniques from *differential cryptanalysis* [5] with fault analysis, and was aptly named *differential fault analysis*.

We present a version of Biham and Shamir's attack on DES and then describe how an adversary can attack DES by exploiting permanent register faults. Before dealing with these topics, we give a brief overview of DES.

3.3.1 DES Algorithm

DES is the most widely recognized and implemented block cipher in the world to date. Most readers will be familiar with this cipher so our description will mainly serve as an introduction to the notion which we use in subsequent sections. Further details about DES can be found in [22].

DES is a 16 round Feistel cipher which uses a 56-bit key K to map 64-bit message blocks to 64-bit ciphertext blocks. Each round of DES updates two 32-bit registers, R_i and L_i, using the round function f and some bits of K. A DES encryption is described in Figure 3.3.

The algorithm works as follows. The message M is subject to an initial permutation, IP, and is then halved into L_0 and R_0. Each half is then updated according to the operations described in lines 4 and 5. The round function, f, takes two

```
INPUT: $M, K$
OUTPUT: $C = \text{DES}_K(M)$
1 derive the subkeys $K_1, K_2, \ldots, K_{16}$ from $K$
2 $L_0R_0 \leftarrow IP(M)$
3 for $i = 1 \ldots 16$ do
4     $L_i \leftarrow R_{i-1}$
5     $R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, K_i)$
6 $C \leftarrow FP(R_{16}L_{16})$
7 return $C$
```

FIGURE 3.3: The DES algorithm.

inputs, the value of R_{i-1} and a subkey. Each of the 16 subkeys, $K_1, K_2, \ldots, K_{16}$, is composed of a subset of 48 bits of K. The subkeys are pre-computed in line 1 but to save memory it is also possible to generate them on the fly.

The round function f is defined as:

$$f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i)).$$

Here, the right half, R_{i-1}, is expanded to 48 bits by the expansion permutation, E, and is then xored with K_i. The result is then broken into 6 bit blocks and used to index entries in 8 tables or S-boxes. This operation is denoted by S. Each table entry is 4 bits so the result of S is 32 bits. The bits of the returned table entries are then permuted according to the round permutation P.

At the end of round 16 a final permutation, FP, is applied to the right and left halves. This is the inverse of the initial permutation (i.e., $FP = IP^{-1}$). The output of the algorithm is $C = DES_K(M)$, the encryption of M under the key K.

The definition of all the permutations and tables used in DES are public knowledge. Thus, the security of a DES encryption rests solely in the secrecy of the key.

3.3.2 Differential Fault Analysis

Consider a smartcard which implements DES, as summarized in Figure 3.3. The environment in which the smartcard operates can be controlled by any party in possession of it, so there are several ways in which a malicious user can force a malfunction, including changing the power supply voltage, adjusting the clock frequency or applying radiation.

Suppose that smartcard malfunctions are realized as single bit inversions in the registers which store the 32-bit values L_{i-1} and R_{i-1}. These faults affect intermediate values computed during a DES encryption and can therefore cause erroneous output. From the description of DES, we have $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$, so the only consequence of a single bit error in L_{i-1} is an identical single bit error in R_i. Because of this 1-1 correspondence, we can simplify our fault model to consider errors only in R_{i-1}. In the following analysis, we assume erroneous encryptions are the result of a single bit of R_{i-1} being flipped, for some value of i. Figure 3.4 describes a version of the DES algorithm under this model.

To mount Biham and Shamir's attack, an adversary obtains two encryptions of some (possibly unknown) plaintext from the smartcard. One encryption is carried

```
INPUT: $M, K$
OUTPUT: $C = \text{DES}_K(M)$
1 derive the subkeys $K_1, K_2, \ldots, K_{16}$ from $K$
2 $L_0R_0 \leftarrow IP(M)$
3 for $i = 1 \ldots 16$ do
* register fault:  $R_{i-1} \leftarrow R_{i-1} \oplus e$
4     $L_i \leftarrow R_{i-1}$
5     $R_i \leftarrow L_{i-1} \oplus f(R_{i-1}, K_i)$        $\triangleright$ where $f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i))$
6 $C \leftarrow FP(R_{16}L_{16})$
7 return $C$
```

FIGURE 3.4: A faulted version of DES the algorithm.

out under normal environmental conditions, resulting in the ciphertext C, and the other is carried out under some environmental stress so that the register fault at line $*$ occurs, resulting in the ciphertext $\hat{C}$. We will assume, for the time being, that only one register fault occurs during an erroneous encryption. Denote the value of i, or equivalently, the encryption round, when the fault occurred by i^*. Ciphertext $\hat{C}$ was corrupted by a single bit error in R_{i^*-1}. Figure 3.4 denotes a particular bit error using a 32-bit string, e, which has Hamming weight equal to 1.

By inverting the final permutation, FP, an adversary can construct $R_{16}L_{16}$ from C and $\hat{R}_{16}\hat{L}_{16}$ from $\hat{C}$. Further, since $L_{16} = R_{15}$ (Figure 3.4, line 4) an adversary also knows R_{15} and $\hat{R}_{15}$. If the register fault occurred in round 16 (i.e., $i^* = 16$) then $R_{15} \oplus \hat{R}_{15}$ will reveal precisely which bit of $\hat{R}_{15}$ was inverted. The subsequent steps in the attack may seem more intuitive with reference to a diagram of the execution path of a DES encryption, as shown in Figure 3.5.

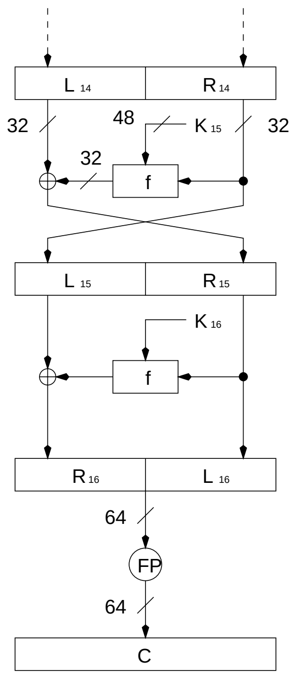

FIGURE 3.5: The computational path of the last few rounds of a DES encryption.

Continuing under the assumption $i^* = 16$, we have $L_{15} = \hat{L}_{15}$. The output of the function f in round 16 is masked by this value, but an attacker can calculate

$$\begin{aligned} R_{16} \oplus \hat{R}_{16} &= \big(L_{15} \oplus f(R_{15}, K_{16})\big) \oplus \big(\hat{L}_{15} \oplus f(\hat{R}_{15}, K_{16})\big) \\ &= f(R_{15}, K_{16}) \oplus f(\hat{R}_{15}, K_{16}) \end{aligned}$$

to reveal the difference in the output of the two round functions. Moreover, since permutations are linear operations, the difference in the output of the S-box table lookups is revealed by:

$$P^{-1}(R_{16} \oplus \hat{R}_{16}) = S(E(R_{15}) \oplus K_{16}) \oplus S(E(\hat{R}_{15}) \oplus K_{16})$$

By design, the S-box operation is nonlinear so the influence of K_{16} is *not* cancelled out in this last calculation. The difference in the input to the S-box operation is revealed by $E(R_{15} \oplus \hat{R}_{15})$.

Differential cryptanalysis uses these input and output differences to derive information about the 48 bit subkey K_{16}. To illustrate, suppose we have:

$$\begin{aligned} E(R_{15} \oplus \hat{R}_{15}) &= \texttt{0x100000000000} & \qquad P^{-1}(R_{16} \oplus \hat{R}_{16}) &= \texttt{0xC0000000} \\ &= 000100|0\ldots0 & &= 1100|0\ldots0 \end{aligned}$$

These values[3] indicate that in round 16, the input difference to the first S-box, S_1, is 000100, or `0x4`, and the output difference is 1100 or `0xC`. Six bits of K_{16} influence the output difference of S_1. Referring to the difference distribution tables compiled in [5], we see that out of all possible 6-bit values only two can produce this output difference. Hence, we are effectively able to deduce 5 bits of K_{16}.

$\mathbf{S_1}$	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD	0xE	0xF
0x1	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
0x2	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
0x4	0	0	0	6	0	10	10	6	0	4	6	4	**2**	8	6	2
0x8	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
0x10	0	0	0	0	0	**0**	2	14	0	6	6	12	4	6	8	6
0x20	0	0	0	10	0	**12**	8	2	0	6	4	4	4	2	0	12

FIGURE 3.6: Rows from the difference distribution table of S_1.

[3]Hexadecimal values are prefixed with `0x`.

The rows of the difference distribution table for S_1 which correspond to single bit errors are presented in Figure 3.6. The leftmost column of the table indicates the input difference and the uppermost row indicates the output difference. The remaining entries enumerate the number of 6-bit values of the key which produce a given output difference. One of the design criteria of the S-boxes was that changing an input bit causes at least two output bits to change. This explains why five columns of zeroes appear in Figure 3.6.

On average, an error in round 16 eliminates all but $\frac{64}{11} \approx 6$ key values for each S-box it affects. The first error which affects an S-box will provide an attacker with about 3 key bits. Because of the definition of the expansion permutation (Figure 3.7) a fault in $\hat{R}_{15}$ is just as likely to affect two S-boxes as one, and this would reveal additional key bits.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

FIGURE 3.7: DES expansion permutation.

Of course, not all faults occur in round 16, but faults in round 15 can be analyzed in a similar manner. Suppose now that $i^* = 15$. In this case we do not know exactly which bit of $\hat{R}_{14}$ was inverted, but the range of possibilities can be narrowed. As before, we can determine R_{15} and $\hat{R}_{15}$, then $P^{-1}(R_{15} \oplus \hat{R}_{15})$ reveals which S-box(es) were affected by the fault in $\hat{R}_{14}$. For example, if this calculation reveals that only S_2 was affected by the fault then we know from Figure 3.7 that the error occurred in bit position 6 or 7. Likewise, if both S_1 and S_2 were affected by the fault, then the error occurred in bit position 4 or 5.

The value of $P^{-1}(R_{15} \oplus \hat{R}_{15})$ also reveals the output difference of any affected S-box in round 15 and this may determine the exact location of the error. Suppose that the fault inverted either bit 1 or 32 and the output difference of S_1 is `0x5`. From the difference distribution table of Figure 3.6 we see there are no 6-bit key values which produce an output difference of `0x5` when the input difference is `0x10`. Therefore, the error could not have inverted bit 1.

The output difference of the S-boxes in round 16 is revealed by $P^{-1}(R_{16} \oplus \hat{R}_{16} \oplus e)$ where e is one of two possible error strings. At least two of the S-boxes will have non-zero input differences and the value of e can mask the output difference of at most one of them. Thus, not knowing the exact value of e is of little consequence since we can work around it if necessary.

Potentially, a fault in round 15 can reveal more information about K_{16} than a fault in round 16. In any case, given a ciphertext pair $\langle C, \hat{C} \rangle$ an adversary can easily determine whether $i^* = 15$ or 16 and then uncover that information. With enough ciphertext pairs all 48 bits of K_{16} can be determined. The remaining 8 bits of K can be found using a brute force search or, alternatively, the last round of DES can be peeled back and then differential fault analysis can be re-applied using a subset of the faulty ciphertext pairs. The latter technique can be used to attack triple DES.

On a personal computer, Biham and Shamir implemented their attack by simulating random faults in the R_i register throughout the 16 rounds of DES; that is, one fault, uniformly distributed over the 16 rounds, per encryption. The two were be able to deduce bits of K_{16} using ciphertext pairs where the erroneous ciphertext resulted from an error in the last three DES rounds.

A particularly clever part of their implementation is illustrated in the way they counted 6-bit key values. Initially, each S-box in round 16 is affected by any one of 64 possible 6-bit key values. As ciphertext pairs are analyzed, input and output differences are derived that narrow the correct 6-bit values to subsets of the 64 possibilities. For each S-box, the number of times that a 6-bit value falls in one of these subsets is counted. After all ciphertext pairs have been analyzed the correct 6-bit values are expected to be counted more frequently than any other value and can whence be identified.

Although on average only $\frac{3}{16}$ of their generated ciphertext pairs were useful in attacking K_{16}, Biham and Shamir found they were able to completely determine this subkey with 50 to 200 ciphertext pairs.

3.3.3 Intrusive Fault Analysis

One criticism of differential fault analysis is that the fault model it assumes is unrealistic. Biham and Shamir responded to this with a host of alternate attacks which exploit permanent or stuck faults in hardware registers, which they hope are less controversial.

These new attacks require an adversary to physically intrude into the circuitry of cryptographic tokens and then fix the contents of some memory cells with the aid of, say, a narrow laser beam. More frugal attackers might choose to probe memory cells [25]. For smart cards, this capability first requires an adversary to expose the circuitry of the embedded chip. Anderson and Kuhn explain how to accomplish this with a process they claim is easy to do [2]. Under this intrusive fault model it is possible to analyze erroneous DES encryptions without the use of the differential tables previously required.

DES may be implemented in hardware using an iterative design so that only one register is used to store the 16 values of L_i. Suppose that the least significant bit of this register is damaged by cutting the wire which either enters or leaves that memory cell, so that its contents are always 0. Figure 3.8 depicts the last round of DES encryption under this assumption.

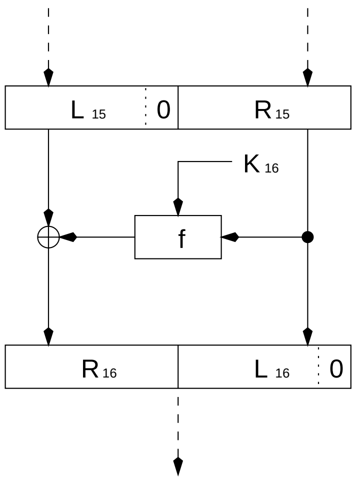

FIGURE 3.8: The effect of a permanent register fault in the last round of DES.

Recall that, given a ciphertext, an adversary can reconstruct R_{16}. The least significant bit of R_{16} equals the least significant bit of L_{15} xored with an output bit of an S-box. Because of the register damage, the least significant bit of L_{15} is 0 so the S-box output bit is revealed. By inverting the round permutation P we find that we can determine an output bit of S_7. The input to S_7 is the xor of 6 unknown key bits and 6 known bits of R_{15}. All of the 64 possible key values can be exhausted to see which ones give an output that agrees with the least significant bit of R_{16}. One ciphertext will eliminate about half of the possible 6-bit key values. With several ciphertexts an attacker can use the key counting technique described previously to identify the correct value. The key input to other S-boxes can be revealed by damaging additional register cells and obtaining more ciphertext.

In this attack it is not necessary for an adversary to process pairs of ciphertext. Ciphertext that results from faulty DES encryptions alone will suffice, although it may be advantageous to obtain a valid plaintext ciphertext pair before any damage is done to the token. With about six ciphertexts per S-box an attacker will be able to uncover K_{16} and then the remaining key bits can be found by brute force search.

If a hardware token implements DES using distinct registers for the values of L_i (i.e., an unrolled implementation) the attack becomes easier. Destroying all the memory cells of L_{15} exposes the output of the S-boxes in round 16. With one ciphertext, the input and output of any S-box can exhaustively compared using each of 64 possible key inputs. This will narrow the key value to one of four possibilities. It takes only about two ciphertexts to determine the last round's subkey.

In iterated implementations of DES it also possible to target key bits across the 16 rounds, rather than the key bits used in a particular round (i.e., a subkey), in what Biham and Shamir describe as a *vertical* attack. In this approach an adversary successively encrypts a constant message, M, 48 times. Each encryption is carried out with an additional one of the wires which transfers subkey bits into the f function severed. Initially, no wires are severed. Denote the resulting ciphertexts by $C_0, C_1, \dots C_{47}$.

Ciphertext C_{47} is the encryption of M with all subkeys equal to zero except in their last bit position. An adversary can now determine the last bit of each subkey by encrypting M under the 2^{16} possible sets of subkeys and comparing it to C_{47}. The value of these bits gives 16 bits of the DES key. Additional key bits can be recovered by repeating this process using C_{46}. The key bits used to form bits 47 and 46 of each subkey are not independent of each other so there are less than 2^{16} values to exhaust in this second step. The attack proceeds by examining each of $C_{47}, C_{46}, C_{45}, \dots$ in this way until the complete key is reconstructed.

3.4 Countermeasures

All of the attacks we have surveyed in this chapter require a cryptographic implementation to somehow provide erroneous output. To resist these attacks, it is sufficient that an implementation simply does not provide this output. To this purpose, the results of cryptographic operations can be verified before they are publicly exposed. Verifying a result requires extra work and the subsequent loss in efficiency depends upon the details of the implementation.

In ciphers such as DES, checking a ciphertext for correctness can be done by computing the encryption function twice and comparing the two results. This decreases efficiency by a factor of 2, and worse, in the random transient fault model, this precaution may fail to detect erroneous output with non-negligible probability. For example, in our discussion of differential fault analysis there were $32 \cdot 16 = 512$ bit positions in which a fault could occur. Given two faulty encryptions, the probability that the same fault occurred in each is $\frac{1}{512}$. Now, the number of encryptions a malicious user must try in order to obtain the required number of faulty ciphertexts increases by a factor of 512. In the intrusive fault model, this countermeasure will fail completely since faults are permanent and they will effect both encryptions in the same way. Using a decryption to verify the correctness of a ciphertext seems to be a better choice for a computational check.

The time it takes to verify an RSA signature depends upon the value of the public exponent, and it is common to use to a small value (e.g., $e = 3$) to exploit this fact. Thus, verifying a signature may not be as costly as generating it, and the overhead of using this countermeasure can be small. When e is large, Shamir has proposed the following check for implementations which use the CRT that is less costly than a full signature verification. Recall that with the CRT, signatures are calculated using the values $S_p = M^d \mod p$ and $S_q = M^d \mod q$ and errors in these computations are particularly disastrous. To facilitate a quick computational check, Shamir instead suggests that a random value, r, about 32 bits in size, be chosen, and then the values $S_{pr} = M^d \mod pr$ and $S_{qr} = M^d \mod qr$ calculated. If $S_{pr} \mod r = S_{qr} \mod r$, then the exponentiations were carried out correctly (with high probability) and the signature can be constructed from a linear combination of $S_p = S_{pr} \mod p$ and $S_q = S_{qr} \mod q$.

Randomization can also be used to resist fault analysis attacks. Padding a message M with random bits before it is encrypted or signed will defeat all of the attacks we have discussed, except for the ones which exploit intrusive faults. For DES and other ciphers with small block sizes, this approach is not likely to be viable since some number of input bits would have to be sacrificed to store random values. For RSA, signature schemes which incorporate randomization have been described in detail and can be implemented with little overhead [3].

Intrusion detection and self-tests are other methods which cryptographic tokens can use to protect against these attacks. Cryptographic hardware is commonly engineered to conform to the FIPS 140-1 standard which encompasses these techniques [21].

3.5 Remarks

Fault analysis has been applied to elliptic curve cryptosystems, as described in [4]. The authors there explain how register faults can perturb points from cryptographically strong curves onto less strong curves. An adversary can then solve the discrete log problem on the weaker curve to gain information about the private key.

The attacks Biham and Shamir proposed which exploit intrusive faults are very similar to the probing attacks described in [25]. The authors in [25] address that fact even if an adversary can intrude into the circuitry of a target device it is unlikely that he or she will be able to target particular memory cells or bus lines. Biham and Shamir do not deal with this issue since they assume that an adversary can damage particular components at will. With this capability, it is likely that an adversary will choose to probe memory cells containing key bits and read the key straight off rather than perform fault analysis.

Chapter 4

Power Analysis

4.1 Introduction

The notion that the power consumption of a cryptographic token can convey sensitive information to an adversary was suggested, almost offhandedly, in [30]. There, Kocher noted that padding the execution time of operations with dummy computations (e.g., empty loops) may be an ineffective defense against timing attacks since the power consumption of dummy computations can be much different from meaningful ones. In this case, an adversary could plot, or *trace*, the power consumption of a token as it executes a particular operation and then deduce a valid timing measurement from the length of the initial pattern in the trace.

It is not difficult to imagine a situation where an adversary might have the opportunity to collect power consumption data. In digital cash systems, a patron typically initiates a purchase by inserting his or her token into a device, such as a reader, which is assigned to a vendor. If the token draws power from the reader then the vendor can potentially monitor this power consumption. So, to evaluate the security of such systems, the information that an adversary can derive from a token's power consumption must be accounted for. Kocher and his newly founded consulting company apparently spent several months investigating this topic.

In 1998, Kocher and the results of his research were again featured in the New York Times [50]. The story there summarized some of the details concerning power analysis that Kocher had recently announced. One particularly startling claim was that for some tokens, a power trace of a *single* cryptographic operation is enough to completely reveal the value of an embedded secret key. Even more startling was the claim that by examining roughly 1000 power traces Kocher and his employees had managed to break *every* smart card product they had examined in the last year and a half. As more technical details [31] concerning these discoveries were released it became clear that power analysis was a serious threat to the security of cryptographic tokens.

Outline

We first explain why the power consumption of tokens is correlated to the calculations they perform. Next, we show how power consumption information can be analyzed to deduce what operation a token is executing at a particular moment, as well as what operands it is manipulating. We describe Simple Power Analysis (SPA), some Hamming weight attacks, and then Differential Power Analysis (DPA). We end by surveying some countermeasures against these attacks.

Much of the author's research into power analysis was conducted during a work term with the Advanced Concepts and Technology group at Pitney Bowes Inc. in Shelton, CT. The graphics displayed in this chapter appear by their courtesies.

4.2 Power Dissipation

Electronic devices draw current from a power source during their operation. The amount of current they draw varies as the paths the current follows through the device changes. To measure the flow of current a small (approximately 10-50 Ω) resistor is put in series with a device's power supply. An oscilloscope can be used to measure the voltage difference across the resistor and the current can then be deduced using Ohm's law[1]. Digital oscilloscopes can be used to sample voltages at high frequencies giving a trace of the flow of current over an interval of time.

The source of current for most devices is supplied at a constant voltage and so the power dissipated by these devices is proportional to the flow of current through them[2]. Because of this, power analysis attacks work just as well with current measurements as they do with power measurements. Hence, the only difference between a *power* analysis attack and a *current* analysis attack is a constant factor.

Most modern cryptographic devices are implemented in CMOS (Complementary Metal Oxide Semiconductor) logic. The basic building block of CMOS logic is the inverter, or NOT gate. As depicted in Figure 4.1, the inverter contains two transistors which act as voltage controlled switches. When the input voltage to the

[1]From Ohm's law, we have $I = \frac{V}{R}$ where V is the voltage measured across a resistance, R, and I is the current.

[2]If P is the power then $P = IV$.

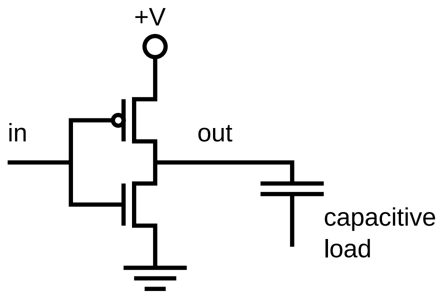

FIGURE 4.1: CMOS logic inverter leading to a capacitor.

inverter is high, the top switch opens while the bottom switch closes. This grounds the inverter's output and so it goes low. Conversely, when the input voltage is high, the top switch closes and bottom opens setting the output to $+V$ which thus goes high.

There is a brief instant, when the inverter is in transition between states, when both transistors conduct current. This causes a short circuit from $+V$ to the ground which temporarily dissipates current. Even in a static state, transistors continuously draw a small amount of current which is dissipated as heat and radiation. The most dominant source of power dissipation is usually caused by the charging and discharging of internal capacitive loads attached to gate outputs. A thorough discussion of all three factors is given in [51] and [16].

Power consumption information is useful to an adversary because it is correlated to the calculations the token is making.

4.3 Correlation with Operations

In devices with microprocessors, such as smart cards, a few primitive operations (e.g., LOAD, STORE, etc.) are used repeatedly during a computation, causing a regular switching of transistors. This regularity is often observable in power traces as repeated patterns. In iterative computations, including most cryptographic algorithms, this regularity is especially apparent and can leak sensitive information to observers.

4.3.1 Simple Power Analysis

Simple power analysis (SPA) is a technique whereby information about the operation of a cryptographic token is deduced directly from a power trace. Depending on how a cipher is implemented, this information may reveal key material.

Figure 4.2 displays two representations of power consumption data acquired from a smart card during the first few rounds of a DES operation. The measurements were collected at a rate of 100 MHz using a digital oscilloscope which converted analog voltages, measured across a resistor, into 12-bit values.

The top trace is composed of raw voltage samples. As transistors in the device switch, the measured voltage either spikes or dips suddenly. Since a large number of transistors switch during this computation the trace appears rather noisy. However, some features, in the early part of the trace at least, can still be discerned. The bottom trace replaces each group of 100 samples in the top trace with their average, which smoothes out most of the erratic spikes and dips. The resulting trace is much clearer and can be compared in greater detail to the description of the DES

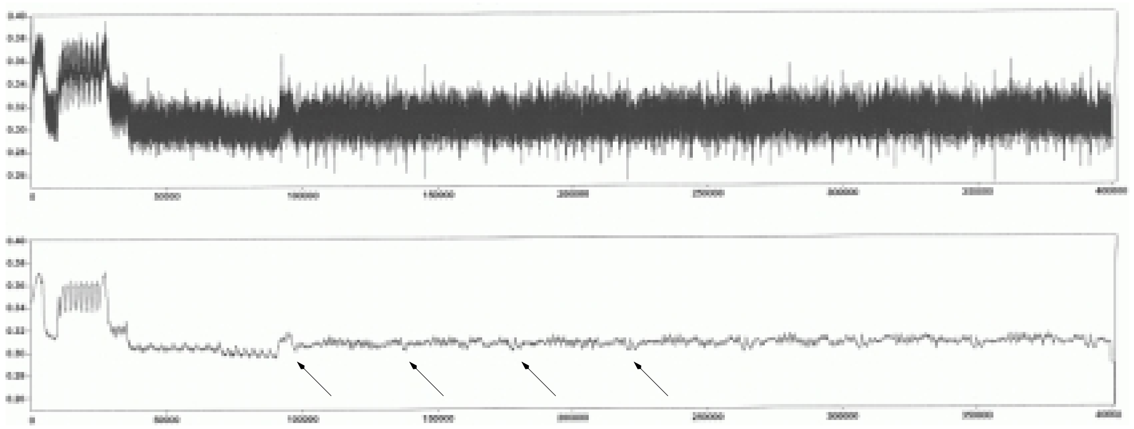

Figure 4.2: SPA traces of a DES operation.

algorithm.

A sequence of operations, constituting a single round, is iterated 16 times during a DES encryption. In the averaged trace of Figure 4.2, we can see a pattern between indices 95 000 and 140 000 which seems to repeat throughout the remainder of the trace. Each occurrence of this pattern is prefixed by a characteristic that resembles either a V or a W. From a trace of the complete DES operation a total of 16 V's and W's appear, marking 16 occurrences of the pattern. This evidence suggests that the pattern may represent the calculations of a single DES round. Figure 4.3 provides a more detailed view of the trace of the first three occurrences of the pattern.

The exact sequence of V's and W's, as read from the complete trace, is: VV-WWWWWWVWWWWWWV. When DES subkeys are generated on the fly, the 56-bit key is initially halved into two registers which are then rotated and permuted as the subkeys are needed. The sequence of V's and W's corresponds exactly to the

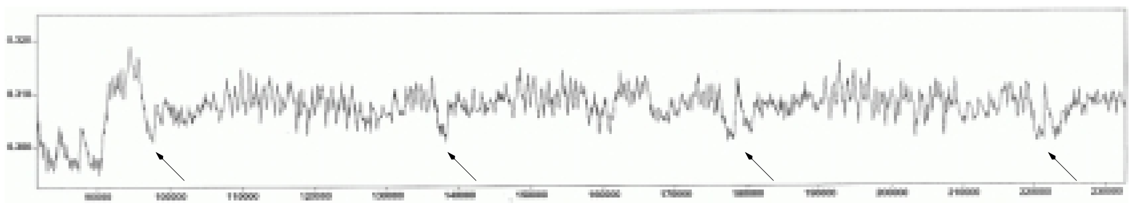

Figure 4.3: An SPA trace of DES rounds one to three.

sequence of rotations described in the DES key schedule [22]: 1122222212222221. From this fact we can infer that the smart card generates its subkeys on the fly.

Identifying the power characteristics of key rotations can sometimes reveal key bits. A common way to implement rotations is to shift one bit off the end of a register and, by default, append a zero on the other [32]. If the bit shifted off the end is a one, then the appended zero bit is flipped. This conditional operation may be detectable in a power trace. In the case of DES, making this determination in each round would reveal all 56 bits of the key.

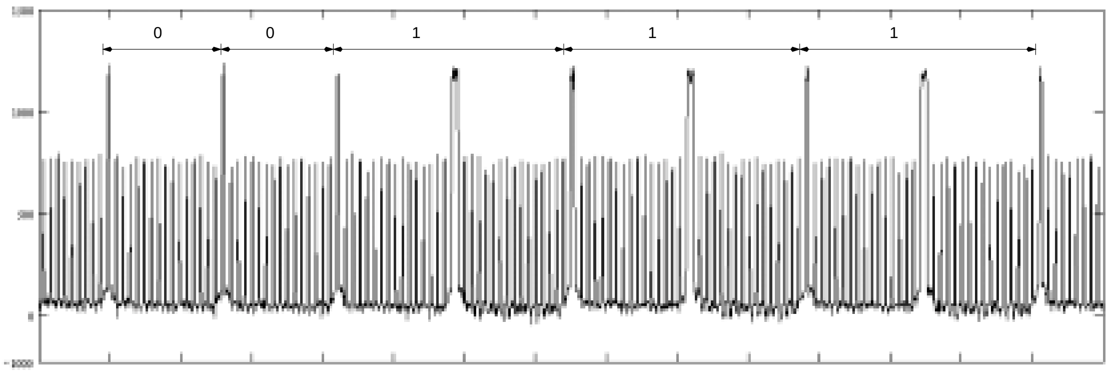

Figure 4.4: An SPA trace of an RSA signature operation.

Exponentiation can also be analyzed using SPA. The conditional branches of square and multiply algorithms can be identified from power traces if the square and multiply operations have different power characteristics. Figure 4.4 shows a portion of a trace from a smart card calculating an RSA signature. Each of the nine spikes indicates the beginning of a square or multiply operation. Initially, registers are loaded with values to be squared or multiplied. Multiplications require additional register loads which increases the width of the leading spike. As a result, square operations (narrow spike) can be distinguished from square-and-multiply operations (narrow spike followed by a wider spike). Thus, five key bits can be determined from the trace: 00111.

Interpreting SPA characteristics is more easily done with some details about the target implementation. With complete details (e.g., source code), an attacker can focus on particular regions of a power trace to try and distinguish the characteristics of specific operations. Generally, any implementation where the path of execution is determined by key bits has a potential vulnerability to this attack.

4.4 Correlation with Operands

Microprocessors retrieve values from memory using a data bus. The data bus has a capacitance associated with it that is charged and discharged according to the values loaded on it. This causes some variation in a device's power consumption, but the effects are usually small and can be overshadowed by measurement error and other sources of noise[3].

Experiments in [42] and [1] have discovered two types of correlations between

[3]Statistically, each power consumption measurement in a trace can be treated as an observation of a random variable. The noise affecting a measurement is just the standard deviation of the corresponding random variable.

data values and power consumption. Hamming weight correlation occurs when power consumption varies with the number of ones driven onto the bus. Transition count correlation occurs when power consumption varies with the number of bits which change on the bus (i.e., the Hamming weight of the xor of the current and previous data value). Which type of correlation is observed in a particular device depends on its design.

The power consumption of operations which manipulate key bits are of particular interest to an adversary. However, without detailed knowledge of an implementation, locating these operations in a single power trace can be difficult. With access to several power traces, an adversary can apply statistical techniques to locate these regions.

In this section we give a brief example of how an adversary might exploit power consumption information correlated to the Hamming weight of operands and then describe some more general attacks which detect power biases due to the value of individual bits.

4.4.1 Hamming Weights

On average, the Hamming weight of a 56-bit DES key conveys only

$$-\sum_{i=0}^{56} \frac{\binom{56}{i}}{2^{56}} \lg \frac{\binom{56}{i}}{2^{56}} \approx 3.95$$

bits of information about its value. The microprocessors used in many cryptographic tokens manipulate data in 8-bit blocks, so power analysis can potentially reveal the Hamming weight of each byte of a DES key. This would provide

$$-\sum_{i=0}^{8} \frac{\binom{8}{i}}{2^8} \lg \frac{\binom{8}{i}}{2^8} \approx 2.54$$

bits of information per key byte for a total of $7 \times 2.54 \approx 17.8$ key bits. This information makes a DES key (even more) susceptible to a brute force attack since the size of key space is now reduced to roughly 2^{38}. However, against ciphers with longer key lengths, such as triple-DES, exhaustive keys searches are infeasible even with Hamming weight information.

10	51	34	60	49	17	33	57
2	9	19	42	3	35	26	25
44	58	59	1	36	27	18	41
22	28	39	54	37	4	47	30
5	53	23	29	61	21	38	63
15	20	45	14	13	62	55	31

FIGURE 4.5: The round one DES subkey.

Depending on the details of the target cipher, it may be possible to use Hamming weight information in more effective attacks. This is the case with DES, as noted in [42] and [7]. To illustrate, denote the bits of a DES key, including parity check bits[4], by $k_1 k_2 \ldots k_{64}$. The key bits which compose the first round's subkey are described in Figure 4.5. The Hamming weight of the first byte of this subkey can be described with the equation:

$$k_{10} + k_{51} + k_{34} + k_{60} + k_{49} + k_{17} + k_{33} + k_{57} = w_1.$$

[4]Every eighth bit is set so that each key byte has an odd Hamming weight.

Expressing the Hamming weight of all key bytes throughout all subkeys in this way generates a total of 96 equations in 56 unknowns. A calculation using linear algebra software shows that the coefficient matrix of this system has full rank and so there is a unique solution for any vector of Hamming weights.

Practically speaking, it is likely that any vector of Hamming weights deduced from a power trace will contain errors which can cause difficulties in finding an integral solution to the system. This problem can be overcome in two ways. The redundancy in the system of equations can be exploited using standard techniques from error correcting codes. Alternately, a careful study of the DES key schedule shows that each of the 96 equations contains variables from only one of two subsets of 28 key bits[5]. Thus, the original system can be split into two independent systems of 48 equations and 28 unknowns. Each of the 2^{28} possible solutions in each system can be tried to see which ones agree most closely with the observed Hamming weights. Thus, the value of the DES key can be deduced.

4.4.2 Differential Power Analysis

Differential power analysis (DPA) is probably the most threatening attack to result from Kocher's research. To carry out a DPA attack, an adversary must have a number of power traces collected from a token as it repeatedly executes a cryptographic operation. The attack proceeds by deducing bits of the secret key, used in each operation, from the observed power consumption. An adversary must also have knowledge of either the inputs or outputs processed by the device during

[5]This property results from the definition of the DES permutation PC-2.

each operation. Usually, an encryption token will use the same key over multiple operations and any generated ciphertext can be freely obtained by an eavesdropper.

The basic technique of DPA is as follows. Suppose an adversary is able to partition power traces from several cryptographic operations into two groups according to the intermediate value of some bit, b, calculated during each operation. This bit is manipulated during each operation and its value may affect the observed power consumption. If this is the case then the two groups of traces should show respectively different power biases at locations when b is manipulated. Averaging the traces in each group helps reduce any noise that may be obscuring these usually small biases. Plotting the difference of the two average traces reveals any locations in the traces where these biases occur.

More precisely, let $T_1, T_2, \ldots, T_n$ be the traces collected from a token. Each trace is an array of k power consumption measurements and represents the power consumed during each cryptographic operation. For example, a token might execute, say, 1000 encryptions allowing an adversary to collect $n = 1000$ traces and 1000 corresponding ciphertexts. The number of measurements in each trace, k, depends on the sampling rate and memory capacity of an adversary's equipment, as well as the duration of the cryptographic operation. Typically, we might have $10^4 \leq k \leq 10^6$.

The two halves of the partition are defined as:

$$\mathcal{T}_0 = \{T_i : b = 0\}$$
$$\mathcal{T}_1 = \{T_i : b = 1\}.$$

The value of b is usually related to the inputs or outputs processed by the token. If these inputs or outputs are sufficiently random, then both $\mathcal{T}_0$ and $\mathcal{T}_1$ will contain roughly the same number of traces. The partitioning bit b might simply be a particular bit of ciphertext.

For $j = 1 \dots k$, the average traces are defined as:

$$A_0[j] = \frac{1}{|\mathcal{T}_0|} \sum_{T_i \in \mathcal{T}_0} T_i[j]$$
$$A_1[j] = \frac{1}{|\mathcal{T}_1|} \sum_{T_i \in \mathcal{T}_1} T_i[j]$$

where $|\mathcal{T}_1| + |\mathcal{T}_0| = n$ and $T_i[j]$ is the jth power consumption measurement in trace T_i. Each of A_0 and A_1 is an array of k averages. The difference, or differential trace, of A_0 and A_1 is defined for $j = 1 \dots k$ as:

$$\Delta[j] = A_1[j] - A_0[j].$$

It might be that the token manipulates bit b more than once throughout an operation. This is the case with the plaintext bits that enter a DES implementation. Suppose the bit b is manipulated by the token at times j^*. If the expected difference in power when the token manipulates the two values of b is ϵ, then we have:

$$E[\ T_i[j^*] \mid b = 1\ ] - E[\ T_i[j^*] \mid b = 0\ ] = \epsilon.$$

At times $j \neq j^*$ the power consumption is independent of the value of b, so:

$$E[\ T_i[j] \mid b = 1\ ] - E[\ T_i[j] \mid b = 0\ ] = E[\ T_i[j]\ ] - E[\ T_i[j]\ ] = 0.$$

As the number of traces grows, $A_1[j]$ and $A_0[j]$ converge to $E[\ T_i[j] \mid b = 1\ ]$ and $E[\ T_i[j] \mid b = 0\ ]$ respectively. Thus we have:

$$\lim_{n\to\infty} \Delta[j] = \lim_{n\to\infty} A_1[j] - A_0[j] = \begin{cases} \epsilon & \text{for } j = j^* \\ 0 & \text{otherwise} \end{cases}$$

so the differential trace will appear flat with spikes of height ϵ at times j^*.

Figure 4.6 displays the result of this technique when applied to an implementation of DES. Using known plaintexts, traces of the first two rounds of one thousand DES encryptions were partitioned into two sets according to the value of the first bit of the register R_0. This bit is just a copy of a particular plaintext bit, and the distribution of the plaintexts determined that roughly half of the traces were placed in each partition. For reference, the differential trace is plotted below an average of all the traces collected. A clear bias or spike can be distinguished in the first round.

To see how this technique can be used to recover bits of the secret key, consider another iteration of this last experiment where the first bit of R_1 is used to partition the traces. Recall that $R_1 = L_0 \oplus f(R_0, K_1)$. Since the plaintext used in each encryption is known, the only unknowns in this equation are the key bits. Without knowledge of the key bits, we cannot determine the value of the first bit of R_1 and

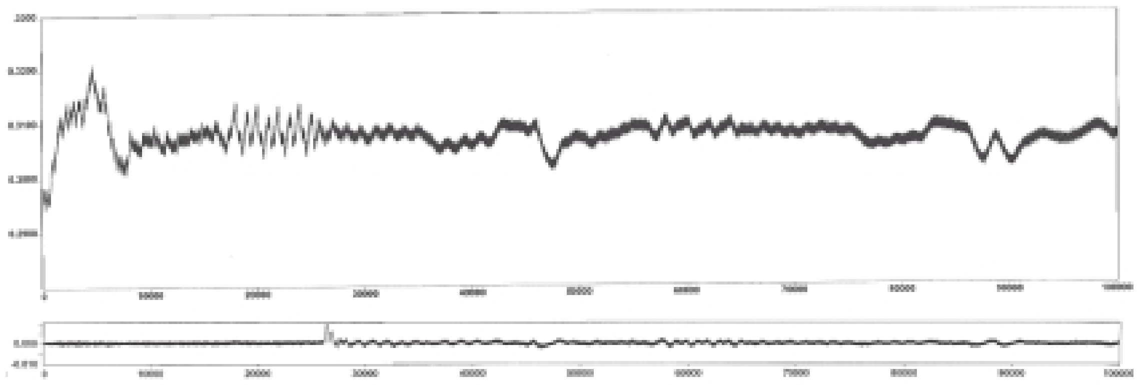

Figure 4.6: The average of 1000 traces and a differential trace.

hence we cannot partition the traces. However, from the definition of the round function f in Figure 4.7, we see that any bit of R_0 is influenced by only 6 key bits.

By exhausting each of the 2^6 key values we can calculate 2^6 different partitions of the traces. Only the correct 6-bit key value will partition the traces according the value of the bit actually calculated in the device. Thus, only one of 2^6 differential traces will show biases and can therefore be identified. Figure 4.8 shows the differential trace for the correct key.

Proceeding in this way, the subkey used in the first round can be reconstructed 6 bits at a time. Once the complete subkey is known, the remaining 8 bits of the DES key can be found using an exhaustive search. If an exhaustive search is not possible, as is the case with triple-DES, the attack can be repeated using the bits of R_2 to partition the traces.

The attack can also implemented using known ciphertexts. In this case, the traces are partitioned using bits of L_{15}. Since $L_{15} = R_{16} \oplus f(L_{16}, K_{16})$, this variation of the attack extracts bits of the last round's subkey.

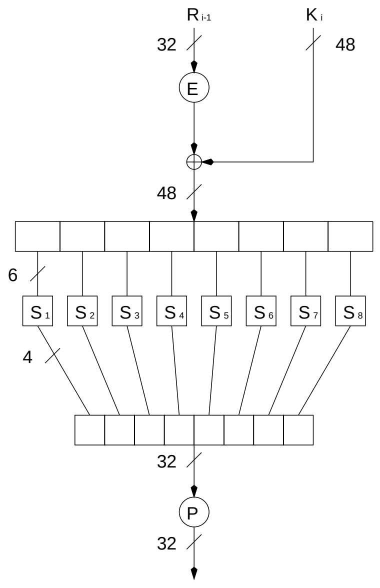

FIGURE 4.7: The DES f function.

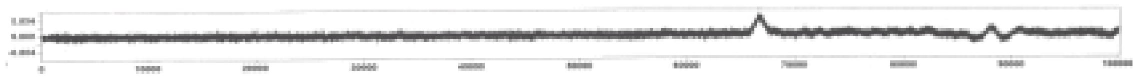

FIGURE 4.8: The differential trace for a correct key guess.

It is important that the execution of the instructions which manipulate the bit used to partition the traces are aligned in each of the traces (i.e., the j^*'s are constant across the different traces). If this is not the case, then the averaging step will degrade the power biases rather than reinforce them. In practice, aligning the power traces can be done by identifying characteristics common to each trace using SPA techniques.

DPA is generally considered to be a more powerful attack than SPA since the only implementation aspect it relies on is that the power consumed when a token processes a 0 is different from when it processes a 1 (i.e., $\epsilon > 0$). In devices which show Hamming weight correlation this is certainly true. With transition correlation, the DPA technique may still be applicable using a partition function based on two bits.

4.4.3 Multiple bit DPA

The number of traces, n, required for a successful DPA attack is related to the size of the power bias ϵ attributed to the value of the partitioning bit and the noise in the power consumption measurements, σ. To see this, note:

$$\mathrm{Var}(A_0[j]) = \mathrm{Var}\left(\frac{1}{|\mathcal{T}_0|}\sum_{T_i\in\mathcal{T}_0} T_i[j]\right) \approx \left(\frac{2}{n}\right)^2 \mathrm{Var}\left(\sum_{T_i\in\mathcal{T}_0} T_i[j]\right) = \frac{2\sigma^2}{n}$$

where $|\mathcal{T}_0|$ is approximated by $\frac{n}{2}$. Thus, $\mathrm{Var}(\Delta[j]) = \mathrm{Var}(A_1[j] - A_0[j]) \approx \frac{4\sigma^2}{n}$ and so the noise in the differential trace is roughly $\frac{2\sigma}{\sqrt{n}}$. To distinguish the biases in Δ we must have $\epsilon > \frac{2\sigma}{\sqrt{n}}$, thus an adversary should choose n larger than $\left(\frac{2\sigma}{\epsilon}\right)^2$. Multiple bit DPA attempts to increase the magnitude of the power bias ϵ so that DPA can be carried out using fewer traces (i.e., smaller value of n).

In devices which show Hamming weight correlation, if the power bias of different bit values is ϵ, then the power bias of different byte values can be as large as 8ϵ. Thus, sorting power traces according to the value of multiple bits can result in differential traces with large spikes, which may be distinguished with fewer traces.

With DES, a guess for the key input to an S-box allows all four output bits of an S-box to be predicted. In our previous description, we kept track of the value of only one output bit and ignored the others. Sorting the traces into the sets:

$$\mathcal{T}_0 = \{T_i : \text{S-box output is } 0000\}$$
$$\mathcal{T}_1 = \{T_i : \text{S-box output is } 1111\}$$

will produce a differential trace with spikes of height roughly 4ϵ. The disadvantage of this approach is that each of $\mathcal{T}_0$ and $\mathcal{T}_1$ contain fewer traces (roughly $\frac{n}{2^4}$ each) so the average traces A_0 and A_1 will contain higher levels of noise. A detailed discussion of the trade offs between spike height and the noise in Δ is given in [42].

Designing multiple bit sorting functions must be done with respect to the words that a device actually manipulates. Although a key guess may allow a few intermediate bits to be determined, multiple bit DPA is only applicable if these bits are manipulated together (e.g., in the same byte).

4.5 Countermeasures

Techniques for resisting power analysis can be implemented at both the hardware and software levels. Countermeasures at the software level seem to be more desirable, from a commercial standpoint at least, since they can be implemented on existing architectures. Hardware countermeasures are generally more costly to implement, but they may be necessary depending on the required level of security. We give examples of countermeasures at each of the two levels now.

Using secret values to perform conditional operations can cause SPA vulnerabilities in cryptographic algorithms. We saw this with RSA in Figure 4.4. Avoiding these types of conditional statements when implementing these algorithms can eliminate many SPA weaknesses. In algorithms which inherently assume this type of key dependent branching, it may not be possible to remove these statements completely. However, operations with large power characteristics (e.g., multiplications) can be moved outside of conditional branches to decrease the size of SPA characteristics. This strategy can be applied to the square-and-multiply algorithm as shown in Figure 4.9.

```
INPUT: $M, N, d = (d_{n-1}d_{n-2}\dots d_1d_0)_2$
OUTPUT: $S = M^d \mod N$
1 $S \leftarrow 1$
2 for $j = n-1 \dots 0$ do
3     $S_0 \leftarrow S^2 \mod N$
4     $S_1 \leftarrow S_0 \cdot M \mod N$
5     $S \leftarrow S_{d_j}$
6 return $S$
```

FIGURE 4.9: An SPA resistant version of the square-and-multiply algorithm.

The microcode run by some microprocessors can cause large operand dependent power consumption features, as noted in [32, 12, 38]. Even constant execution path code can demonstrate serious power analysis vulnerabilities when run on these components. One way to counteract this problem is to split operands into shares, using a threshold scheme (a technique of secret sharing), and then have the processor compute by manipulating shares of sensitive data rather than the data itself [12]. To deduce sensitive data, an adversary must now combine multiple power consumption measurements from various locations within a power trace. This effectively increases the amount of noise, σ, obscuring the value of the sensitive data. Recall that, for a successful DPA attack, $n > \left(\frac{2\sigma}{\epsilon}\right)^2$. So, increasing σ causes the number of required power traces to increase. The authors in [12] argue that the number of power traces required for a successful DPA attack increases exponentially as a function of the number of shares. Unfortunately, the performance penalty associated with this countermeasure limits its practicality. The technique of *random masking*, a similiar mode of defense introduced in [23], has better performance characteristics. However, implementating this countermeasure must be done carefully, as shown in [40] and [15].

Interleaving random computations into the execution of cryptographic operations is a common defense against DPA. If an encryption operation is interrupted at random times with dummy computations then the times at which, say, a particular key byte is manipulated will vary from encryption to encryption. Power traces collected from devices protected in this way will not be aligned with respect to the operations the device has performed. As a result, spikes which would normally appear thin and tall in a differential trace appear shorter and are smeared across an interval. Similar to the secret sharing countermeasure, this technique increases the amount of noise in the differential trace, which hopefully increases the number of traces necessary for a successful DPA attack to an unreasonable number. More details on this technique can be found in [14]. Microprocessors which are capable of randomized multithreading are especially suited to this countermeasure. Clocking devices using a randomized clock signal produces a similar effect [34].

Hardware components (e.g., capacitors and inductors) can be added to the power line of tokens to filter, or smooth out, power consumption characteristics [16, 45]. This approach attempts to decrease the size of the power bias, ϵ, thereby increasing the number of traces required for a successful DPA attack.

A hardware countermeasure, which has been developed with smartcard systems in mind, is to ensure that external power supplies are never connected directly to the internal chip [48, 45]. This approach attempts to decorrelate the flow of current on external power lines from internal computations. This is done by inserting a kind of buffer between external power lines and internal ones. Of course, the internal chip needs power, so the buffer must accommodate this. Systems of capacitors and transformers have been proposed which function in this way.

Unfortunately, given enough power consumption traces, adversaries can overcome most countermeasures. For this reason, system designers should adopt a *leak-tolerant* design methodology, as recommended in [32] and [16]. As a token consumes power, engineers should expect that some secret information will be leaked to observers. The rate at which information is leaked can be used to determine key lifespans. Keys can be refreshed using non-linear update functions (e.g., SHA-1) when they expire. Several tests to determine the leakage rate of devices are proposed in [16].

4.6 Remarks

Power analysis has been applied to many different ciphers, including several of the recent AES candidates [11]. Depending on the ways that an adversary can manipulate a token, it is possible to attack a cipher with different variations of power analysis. For example, in [43], it is explained how the ability to re-key a token, or a copy of a token, can be exploited with power analysis.

Some researchers have proposed attacks on tokens which make use of highly detailed power consumption profiles [20, 1, 7]. The work required to profile a device is substantial when compared with standard DPA, but the profile can be reused to extract keys from several tokens which presumably use different keys.

Details on the equipment necessary to perform power analysis attacks can be found in [11]. We only note that the cost of this equipment is low; an adversary should be able to purchase the required equipment for less than \$10,000.

Chapter 5

Conclusions

This thesis has presented several ways in which an adversary might use side channel information to cryptanalyze ciphers such as DES and RSA. The purpose of discussing these particular ciphers was to exploit the reader's likely familiarity with them — a precedent which was set in the first papers to deal with this type of cryptanalysis [30, 10, 32]. Many of the techniques which we applied to these ciphers can be readily applied to others. The timing attack, for example, was not so much an attack on RSA as it was an attack on modular exponentiation. Any cryptosystem which implements this operation (e.g., DSS or Diffie-Hellman) may be vulnerable.

Of the three sources of side channel information we considered, it seems that power consumption presents the most serious problem to cryptographic engineers. Recall that message blinding and computational checks were relatively effective software countermeasures against timing and fault analysis. Unfortunately, there has yet to appear a defense with similar qualities against attacks like differential power analysis. The secret sharing countermeasure presented in [12] is attractive because of the proof of security which comes with it. However, the performance penalty incurred in using this countermeasure is quite high in terms of memory and execution time. Contrary to what was initially suggested, secret sharing must be done extensively throughout a cipher's computation, rather than only in, say, the first three and last three rounds of DES. Profiling attacks can be used to conduct DPA on the inner rounds of DES, so secret sharing must be used there as well.

Much of the power analysis literature which appears now focuses on ways to defeat previously suggested countermeasures [14, 15]. This trend has caused any newly suggested countermeasures to be greeted with much scepticism, but this is an important part of developing a sound defense.

The reader who is interested in pursuing his or her own investigations into side channel cryptanalysis will hopefully find the collection of references in the bibliography useful. Since this is a relatively new field of study much of the relevant literature is not well known. So far, the majority of the papers on the subject have been presented at the Cryptographic Hardware and Embedded Systems (CHES) conferences.

Bibliography

[1] M.L. Akkar, R. Bevan, P. Dischamp, and D. Moyart. Power Analysis: What is now Possible. In T. Okamoto, editor, *Advances in Cryptology - Proceedings of ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 489–502. Springer-Verlag, 2000.

[2] R. Anderson and M. Kuhn. Tamper Resistance – a Cautionary Note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pages 1–11, November 1996. Available from `http://www.usenix.org`.

[3] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA. In U. Maurer, editor, *Advances in Cryptology - Proceedings of EUROCRYPT 96*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996. Available from `http://www-cse.ucsd.edu/users/mihir`.

[4] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *LNCS*, pages 131–146. Springer-Verlag, 2000.

[5] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer-Verlag, 1993.

[6] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In B. Kaliski, editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *LNCS*, pages 513–525. Springer-Verlag, 1997.

[7] E. Biham and A. Shamir. Power Analysis of the Key Scheduling of the AES Candidates. In *Second AES Candidates Conference*, March 1999. Available from `http://csrc.nist.gov/encryption/aes/round1/-conf2/aes2conf.htm`.

[8] G. Blom. *Probability and Statistics: Theory and Applications*. Springer-Verlag, 1989.

[9] D. Boneh. Twenty Years of Attacks on the RSA Cryptosystem. *Notices of the American Mathematical Society*, 46(2):203–213, 1999. Available from `http://crypto.stanford.edu/~dabo/pubs.html`.

[10] D. Boneh, R. DeMillo, and R. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *LNCS*, pages 37–51. Springer-Verlag, May 1997.

[11] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In *Second AES Can-*

didates Conference, March 1999. Available from `http://csrc.nist.gov/-encryption/aes/round1/conf2/aes2conf.htm`.

[12] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer-Verlag, August 1999.

[13] D. Chaum. Blind Signatures for Untraceable Payments. In R. Rivest and A. Sherman and D. Chaum, editor, *Advances in Cryptology - Proceedings of CRYPTO 82*, volume 0, pages 199–203. Plenum Press, 1983.

[14] 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 *LNCS*, pages 252–263. Springer-Verlag, August 2000.

[15] J.S. Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 231–237. Springer-Verlag, August 2000.

[16] J.S. Coron, P. Kocher, and D. Naccache. Statistics and Secret Leakage. In *Financial Cryptography '00*, 2000.

[17] J. Daemen and V. Rijmen. Resistance Against Implementation Attacks: A Comparitive Study of the AES Proposals. In *Second AES Candi-*

dates Conference, March 1999. Available from `http://csrc.nist.gov/-encryption/aes/round1/conf2/aes2conf.htm`.

[18] J.F. Dhem, F. Koeune, P.A. Leroux, P. Mestré, J.J. Quisquater, and J.L. Willems. A Practical Implementation of the Timing Attack. Technical Report CG-1998/1, Université catholique de Louvain, 1998. Available from `http://www.dice.ucl.ac.be/crypto`.

[19] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[20] P. Fahn and P. Pearson. IPA: A New Class of Power Attacks. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *LNCS*, pages 173–186. Springer-Verlag, August 1999.

[21] FIPS 140-1. Security Requirements for Cryptographic Modules. Federal Information Processing Standard, National Institute of Standards and Technology, January 1994. Available from `http://csrc.nist.gov/fips/fips1401.pdf`.

[22] FIPS 46-3. Data Encryption Standard. Federal Information Processing Standard, National Institute of Standards and Technology, 25 October 1999. Available from `http://www.itl.nist.gov/fipspubs/by-num.htm`.

[23] L. Goubin and J. Patarin. DES and Differential Power Analysis - The "Duplication" Method. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *LNCS*, pages 158–172. Springer-Verlag, August 1999.

[24] H. Handschuh and H. Heys. A Timing Attack on RC5. In *In Workshop Record of Selected Areas of Cryptography - SAC '98*, pages 318–329. Queen's University, 1998.

[25] H. Handschuh, P. Paillier, and J. Stern. Probing Attacks on Tamper-Resistant Devices. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *LNCS*, pages 303–315. Springer-Verlag, August 1999.

[26] M.A. Hasan. Countermeasures for Koblitz Curve Cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 93–108. Springer-Verlag, August 2000.

[27] K. Heidenstrom. FAQ / Application notes: Timing on the PC family under DOS, 1995. Available from `ftp://ftp.simtel.net/pub/simtelnet/-msdos/info/pctim003.zip`.

[28] A. Hevia and M. Kiwi. Strength of Two Data Encryption Standard Implementation Under Timing Attacks. *ACM Transactions on Information and System Security*, 2(4):416–437, November 1999.

[29] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side Channel Cryptanalysis of Product Ciphers. *Journal of Computer Security*, 8(2-3):141–158, 2000. Available from `http://www.counterpane.com/side_channel.html`.

[30] P. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In N. Koblitz, editor, *Advances in Cryptology -*

CRYPTO '96, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, August 1996. An alternate version is available from `http://www.cryptography.com/-timingattack/paper.html`.

[31] P. Kocher, J. Jaffe, and B. Jun. Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc., 1998. Available from `http://www.cryptography.com/dpa/technical/index.html`.

[32] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, August 1999. Available from `http://www.cryptography.com/dpa/Dpa.pdf`.

[33] F. Koeune and J.J. Quisquater. A Timing Attack Against Rijndael. Technical Report CG-1999/1, Université catholique de Louvain, 1999. Available from `http://www.dice.ucl.ac.be/crypto`.

[34] O. Kömmerling and M. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *USENIX Workshop on Smartcard Technology - Smartcard '99*, pages 9–20. USENIX Association, May 1999.

[35] J. Markoff. Secure Digital Transactions Just Got a Little Less Secure. *New York Times*, page A1, 11 December 1995.

[36] J. Markoff. Potential Flaw in Cash Card Security Seen. *New York Times*, page D1, 26 September 1996.

[37] M. Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In Y. G. Desmedt, editor, *Advances in Cryptology - CRYTPO '94*, volume 839 of *LNCS*, pages 1–11. Spring-Verlag, August 1994.

[38] R. Mayer-Sommer. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 78–92. Springer-Verlag, August 2000.

[39] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC, 1996.

[40] T. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption Workshop - FSE 2000*, volume 1978 of *LNCS*. Springer-Verlag, April 2000.

[41] T. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer-Verlag, August 2000.

[42] T. Messerges, E. Dabbish, and R. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *USENIX Workshop on Smartcard Technology*, pages 151–161, May 1999. Available from `http://www.eecs.uic.edu/~tmesserg/papers.html`.

[43] T. Messerges, E. Dabbish, and R. Sloan. Power Analysis Attacks of Modular

Exponentiation in Smart Cards. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES '99*, volume 1717 of *LNCS*, pages 144–157. Springer-Verlag, August 1999.

[44] PKCS #1 v2.0. RSA Cryptography Standard. Public Key Cryptography Standard, RSA Laboratories, September 1998. Available from `ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-1v2.asc`.

[45] P. Rakers, L. Connell, T. Colins, and D. Russell. Secure Contactless Smartcard ASIC with DPA Protection. In *IEEE 2000 Custom Integrated Circuits Conference*, pages 239–242, May 2000.

[46] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Technical Memo LCS/TM 82, MIT Laboratory for Computer Science, 4 April 1977. Revised 12 December 1977.

[47] W. Schindler. A Timing Attack against RSA with the Chinese Remainder Theorem. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 109–124. Springer-Verlag, August 2000.

[48] A. Shamir. Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *LNCS*, pages 71–77. Springer-Verlag, August 2000.

[49] D. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[50] P. Wayner. Code Breaker Cracks Smart Cards' Digital Safe. *New York Times*, page D1, 22 June 1998.

[51] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 2nd edition, 1994.

[52] P. Wright. *Spy Catcher: The Candid Autobiography of a Senior Intelligence Officer*. Viking, 1987.