Chapter 13

Attacks on Cryptosystems

Up to this point, we have mainly seen how ciphers are implemented. We have seen how symmetric ciphers such as DES and AES use the idea of substitution and permutation to provide security and also how asymmetric systems such as RSA and Diffie Hellman use other methods. What we haven't really looked at are attacks on cryptographic systems. An understanding of certain attacks will help you to understand the reasons behind the structure of certain algorithms (such as Rijndael) as they are designed to thwart known attacks. Although we are not going to exhaust all possible avenues of attack, we will get an idea of how cryptanalysts go about attacking ciphers.

This section is really split up into two classes of attack[1]: **Cryptanalytic attacks** and **Implementation attacks**. The former tries to attack mathematical weaknesses in the algorithms whereas the latter tries to attack the specific implementation of the cipher (such as a smartcard system).

The following attacks can refer to either of the two classes (all forms of attack assume the attacker knows the encryption algorithm):

- **Ciphertext-only attack**: In this attack the attacker knows only the ciphertext to be decoded. The attacker will try to find the key or decrypt one or more pieces of ciphertext (only relatively weak algorithms fail to withstand a ciphertext-only attack).
- **Known plaintext attack**: The attacker has a collection of plaintext-ciphertext pairs and is trying to find the key or to decrypt some other ciphertext that has been encrypted with the same key.
- **Chosen Plaintext attack**: This is a known plaintext attack in which the attacker can choose the plaintext to be encrypted and read the corresponding ciphertext.
- **Chosen Ciphertext attack**: The attacker has the able to select any ciphertext and study the plaintext produced by decrypting them.
- **Chosen text attack**: The attacker has the abilities required in the previous two attacks.

The following terminology is also useful to know:

[1] Brute force attacks are also available to the attacker.

- An encryption scheme is **unconditionally secure** if the ciphertext generated does not contain enough information to determine uniquely the corresponding plaintext no matter how much ciphertext is available or how much computational power the attacker has. With the exception of the one time pad, no cipher is unconditionally secure.
- The security of a **conditionally secure** algorithm depends on the difficulty in reversing the underlying cryptographic problem such as how easy it is to factor large primes. All ciphers other than the one-time pad fall into this category.
- An encryption scheme is said to be **computationally secure** if:
    1. The cost of breaking the cipher exceeds the value of the encrypted information
    2. The time required to break the cipher exceeds the useful lifetime of the information.

Brute force attacks are also available to the attacker however we will not be concerned with them in this chapter although it is interesting to note the following. In 1977 Diffie and Hellman claimed that it would cost twenty million dollars to build a million chip machine that could find a DES key in twelve hours (given a plaintext-ciphertext pair). In 1995 it was estimated that advances in chip densities and speeds would permit a several thousand chip machine to do the same job at a cost of well under a million dollars. However in July 1998 a machine was build by EFF, cryptography research and Advanced wireless technologies that could search 90 billion keys per second which would take a little over 200 hours to search the entire key space. They managed however to find the key in 56 hours. The total budget remained under $250,000 which made the machine the fastest most economical key search device ever known to have been produced.

13.1 Cryptanalytic Attacks

All forms of cryptanalysis for symmetric encryption schemes are designed to exploit the fact that traces of structure or pattern in the plaintext may survive encryption and be discernible in the ciphertext. Cryptanalysis of public-key schemes proceeds from a fundamentally different premise, namely that the mathematical properties of the pair of keys may make it possible for one of the two keys to be deduced from the other. In this section we will only be concerned with three main attacks. Two of them (Differential and Linear cryptanalysis) are used to attack block ciphers whereas the third (birthday attack) is used to attack hash functions.

13.1.1 Differential cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. Although this appears to have been discovered at least 30 years ago it

was not reported in the open literature until 1990. The first published effort appears to have been the cryptanalysis of a block cipher called FEAL. This was followed by a number of papers by Biham and Shamir, who demonstrated this form of attack on a variety of encryption algorithms and hash functions.

The most publicised results for this approach have been those that have application to DES. Differential cryptanalysis is the first published attack that is capable of breaking DES in less than 2^{55} complexity. The scheme can successfully cryptanalyse DES with an effort of 2^{47}, requiring 2^{47} chosen plaintext (hence it is a chosen plaintext attack). Although 2^{47} is certainly significantly less than 2^{55}, the need to find 2^{47} chosen plaintexts makes this attack of only theoretical interest. Apparently this attack was known at the time DES was being designed and played a large part in the design of DES.

The attack can be summarised as follows (a more detailed explanation can be found on the website). Consider the original plaintext block for DES m to consist of two halves m_0, m_1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block m_i ($2 \leq i \leq 17$), then the intermediate message halves are related as follows:

$$m_{i+1} = m_{i-1} \oplus f(m_i, K_i), \qquad i = 1, 2, \ldots, 16$$

In differential cryptanalysis, one starts with two messages, m and m', with a known XOR difference $\Delta m = m \oplus m'$, and considers the difference between the intermediate message halves: $\Delta m_i = m_i \oplus m'_i$. Then we have:

$$\begin{aligned} \Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus f(m_i, K_i)] \oplus [m'_{i-1} \oplus f(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [f(m_i, K_i) \oplus f(m'_i, K_i)] \end{aligned}$$

Now, suppose that many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used. To put this more precisely, let us say that X may cause Y with probability p, if for a fraction p of the pairs in which the input XOR is X, the output XOR equals Y. We want to suppose that there are a number of values of X that have high probability of causing a particular output difference. Therefore, if we know Δm_{i-1} and Δm_i with high probability, then we know Δm_{i+1} with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f.

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages m and m' with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext. Actually, there are two probable differences for the two 32-bit halves: ($\Delta m_{17} || \Delta m_{16}$). Next, we submit m and m' for encryption to determine the actual difference under the unknown key and compare the

result to the probable difference. If there is a match,

$$E_k(m) \oplus E_k(m') = (\Delta m_{17} || \Delta m_{16})$$

then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

13.2 Linear Cryptanalysis

A more recent development is linear cryptanalysis that was presented by Mitsuru Matsui at Eurocrypt '93. This attack is based on finding linear approximations to describe the transformations performed in DES (and other block ciphers). This method can find a DES key given 2^{47} *known* plaintexts, as compared to 2^{47} *chosen* plaintexts for differential cryptanalysis (it is therefore a known plaintext attack although it can also work as a ciphertext only attack). Although this is a minor improvement (because it may be easier to acquire known plaintext rather than chosen plaintext) it still leaves linear cryptanalysis infeasible as an attack on DES. However it is useful for an understanding of other similar attacks and gives an insight into why the S-boxes are constructed the way they are.

To understand the attack we will define a few terms:

A Boolean function $h : Z_2^n \rightarrow Z_2$ in n variables $s_1, \ldots, s_n$ is **linear** if it can be represented as $h(s) = a_1 s_1 \oplus \ldots \oplus a_n s_n$ for some $a_i \in Z_2 = \{0, 1\}, i = 1, \ldots, n$. The set of all linear Boolean functions in n variables is denoted by

$$L_n = \{h : Z_2^n \rightarrow Z_2 | h = a_1 s_1 \oplus \ldots \oplus a_n s_n\}$$

A Boolean function $f : Z_2^n \rightarrow Z_2$ is called **affine** if either $f(s) = h(s)$ or $f(s) = h(s) \oplus 1$, for some $h(s) \in L_n$. The set of all affine Boolean function in n variables is therefore:

$$A_n = L_n \cup \{h \oplus 1 | h \in L_n\} = L_n \cup \overline{L_n}$$

In other words, A_n consists of all linear functions and their negations.

A summary of Linear cryptanalysis is as follows. For a cipher with n-bit plaintext and ciphertext blocks and an m-bit key, let the plaintext block be labeled $P[1], \ldots P[N]$, the cipher text block $C[1], \ldots C[n]$ and the key $K[1], \ldots K[m]$. Then define

$$A[i, j, \ldots, k] = A[i] \oplus A[j] \oplus \ldots \oplus A[k]$$

The objective of linear cryptanalysis is to find an effective *linear* equation of the form

$$P[\alpha_1, \alpha_2, \ldots, \alpha_a] \oplus C[\beta_1, \beta_2, \ldots, \beta_b] = K[\gamma_1, \gamma_2, \ldots, \gamma_c] \tag{13.1}$$

that holds with probability $p \neq 0.5$. Here we have $x = 0, 1; 1 \leq a, b \leq n, 1 \leq c \leq m$, and where α, β and γ terms represent fixed, unique bit locations. The further p is from 0.5, the more effective the equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is 0 more than half the time, assume $K[\gamma_1, \gamma_2, \ldots \gamma_c] = 0$. If it is 1 most of the time, assume $K[\gamma_1, \gamma_2, \ldots, \gamma_c] = 1$. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.

The above explanation give us an overview of the whole attack. Let us expand a little on some of the details. The fact that we desire equation 13.1 to hold with a probability $p \neq 0.5$ implies that it can be $0 \geq p < 0.5$ or $0.5 < p \leq 1$. This leads us to the idea of a **linear probability bias** which is given by $\epsilon = |p - 0.5|$. The larger this bias is (in other words the closer p is to 0 or 1) the better the applicability of linear cryptanalysis with fewer known plaintext. If $p = 1$ this implies that equation 13.1 is a perfect representation of the cipher behaviour and the cipher has a catastrophic weakness. If $p = 0$ then equation 13.1 represents an affine relationship in the cipher which is also a catastrophic weakness. Both linear and affine approximations, indicated by $p > 0.5$ and $p < 0.5$ respectively, are equally susceptible to linear cryptanalysis.

For an ideal cipher what we would like is that the plaintext be mapped to the ciphertext in such a way that the mapping is random. In other words there is no correlation between the plaintext and the ciphertext. By choosing $a + b$ random values (number of plaintext plus ciphertext bits in equation 13.1) equation 13.1 should hold with a probability of exactly 0.5. This would be the case if it were a perfect cipher. However as ciphers are not perfect the probability contains a bias ϵ. This bias not only exists for the overall cipher but also for each of the individual non-linear element (S-boxes in the case of DES). We can therefore find linear approximations for certain S-boxes and use what's known as the **piling up lemma** to concatenate the results to gain an expression for the whole cipher. With this expression, it is possible to take some plaintext and ciphertext pairs and a **target partial subkey** (this is your guess at a part of the subkey) and deduce the target partial subkeys values.

More detailed information is given in the paper "A Tutorial on Linear and Differential Cryptanalysis" by Howard Heys. This paper is on the website.

13.2.1 Birthday attack

The Birthday attack makes use of what's known as the Birthday paradox to try to attack cryptographic hash functions. The birthday paradox can be stated as follows: What is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday? It turns out that the answer is 23 which is quite a surprising result. In other words if there are 23 people in a room, the probability that two of them have the same birthday is approximately 0.5. If there is 100 people (i.e. k=100) then the probability is $.9999997$, i.e. you are almost

guaranteed that there will be a duplicate. A graph of the probabilities against the value of k is shown in figure 13.1.

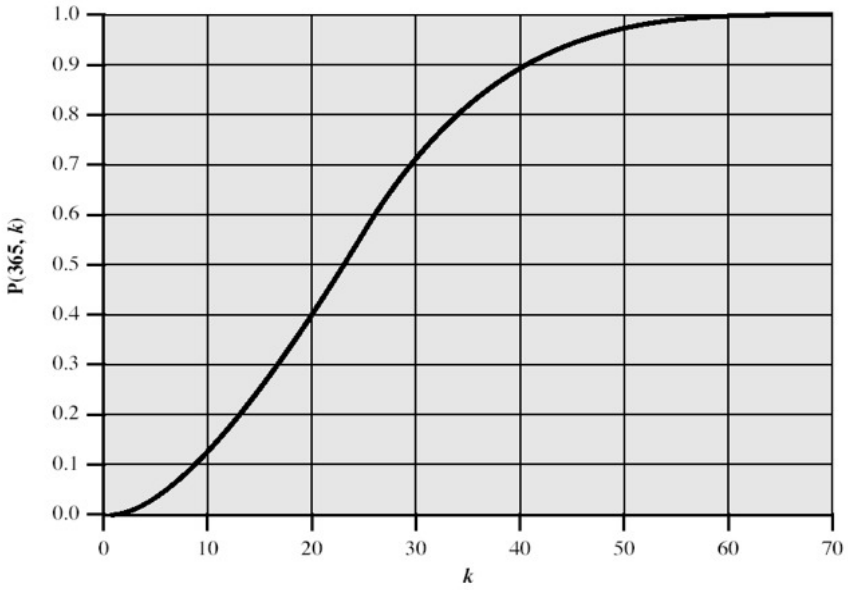

Figure 13.1: The Birthday Paradox.

Although this is the case for birthdays we can generalise it for n equally likely values (in the case of birthdays $n = 365$). So the problem can be stated like this: Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability $P(n, k)$, that there is at least one duplicate?

It turns out (see Stallings) that this value is

$$\begin{aligned} P(n,k) &= 1 - \frac{n!}{(n-k)!n^k} \\ &= 1 - [(1 - \frac{1}{n}) \times (1 - \frac{2}{n}) \times \ldots \times (1 - \frac{k-1}{n})] \end{aligned} \tag{13.2}$$

Take it that the following inequality holds (to see why see Stallings)

$$(1 - x) \leq e^{-x} \tag{13.3}$$

then we have

$$\begin{aligned} P(n,k) &> 1 - [(e^{-1/n}) \times (e^{-2/n}) \times \ldots \times (e^{-(k-1)/n})] \\ &> 1 - e^{-[(1/n)+(2/n)+((k-1)/n)]} \\ &> 1 - e^{-(k\times(k-1))/2n} \end{aligned} \tag{13.4}$$

We would like to know when $P(n,k) > 0.5$ so we set the right hand side of equation 13.4 to 0.5:

$$\begin{aligned} 0.5 &= 1 - e^{-(k\times(k-1))/2n} \\ \Rightarrow ln(2) &= \frac{k \times (k-1)}{2n} \end{aligned} \tag{13.5}$$

For large values of k we can replace $k \times (k-1)$ by k^2 giving

$$k = \sqrt{2(ln2)n} = 1.18\sqrt{n} \approx \sqrt{n} \tag{13.6}$$

which can be seen to be almost equal to 23 for $n = 365$.

Now lets look at this in terms of hash codes. Remember a hash code is a function that takes a variable length message M and produces a fixed length message digest. Assuming the length of the digest is m then there are 2^m possible message digests. Normally however, because the length of M will generally be greater than m this implies that more than one message will be mapped to the same digest. Of course the idea is to make it computationally infeasible to find two messages that map to the same digest. However if we apply k random messages to our hash code what must the value of k be so that there is the probability 0.5 that at least one duplicate (i.e. $H(x) = H(y)$ will occur for some inputs x, y)? This is the same as the question we asked about the birthday duplicates. Using equation 13.6 we have

$$k = \sqrt{2^m} = 2^{m/2} \tag{13.7}$$

Using this idea we can discuss the birthday attack as follows:

- The source, A is prepared to "sign" a message by appending the appropriate m-bit hash code and encrypting that hash code with A's private key.
- The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations of the fraudulent message to be substituted for the real one.
- The two sets of messages are compared to find a pair of messages that produce the same hash code. The probability of success is greater than 0.5. If no match

is found, additional valid and fraudulent messages are generated until a match is made.

- The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

If we use a 64-bit hash code then the level of effort required is only on the order of $2^{m/2} = 2^{64/2} = 2^{32}$ which is clearly not sufficient to withstand today's computational systems.

The generation of many variations that convey the same meaning is not that difficult as figure 13.2 shows.

Figure 13.2: A letter in 2^{37} variations.

13.3 Implementation Attacks

Implementation attacks take on a different approach to the above for discovering the secret key. Instead of attacking the mathematical properties of the algorithm these form of attacks (also known as side channel attacks) take advantage of the physical phenomena that occurs when a cryptographic algorithm is implemented in hardware. Four side channel attacks are listed in the FIPS standard 140-2 "Security Requirements for Cryptographic Modules", **Power Analysis**, **Timing Analysis**, **Fault Induction** and **TEMPEST**. The following is an excerpt from this document (which is available on the NIST website).

1. **Power Analysis:** Attacks based on the analysis of power consumption can be divided into two categories, Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA involves a direct (primarily visual) analysis of electrical power consumption patterns and timings derived from the execution of individual instructions carried out by a cryptographic module during a cryptographic process. The patterns are obtained through monitoring the variations in electrical power consumption of a cryptographic module for the purpose of revealing the features and implementations of cryptographic algorithms and subsequently values of cryptographic keys. DPA has the same goals but utilizes advanced statistical methods and/or other techniques to analyze the variations of the electrical power consumption of a cryptographic module. Cryptographic modules that utilize external power (direct current) sources appear to be at greatest risk. Methods that may reduce the overall risk of Power Analysis attacks include the use of capacitors to level the power consumption, the use of internal power sources, and the manipulation of the individual operations of the algorithms or processes to level the rate of power consumption during cryptographic processing.

2. **Timing Analysis:** Timing Analysis attacks rely on precisely measuring the time required by a cryptographic module to perform specific mathematical operations associated with a cryptographic algorithm or process. The timing information collected is analyzed to determine the relationship between the inputs to the module and the cryptographic keys used by the underlying algorithms or processes. The analysis of the relationship may be used to exploit the timing measurements to reveal the cryptographic key or CSPs (Cryptographic Security Parameters). Timing Analysis attacks assume that the attacker has knowledge of the design of the cryptographic module. Manipulation of the individual operations of the algorithms or processes to reduce timing fluctuations during processing is one method to reduce the risk of this attack.

3. **Fault Induction:** Fault Induction attacks utilize external forces such as microwaves, temperature extremes, and voltage manipulation to cause processing errors within the cryptographic module. An analysis of these errors and their

patterns can be used in an attempt to reverse engineer the cryptographic module, revealing certain features and implementations of cryptographic algorithms and subsequently revealing the values of cryptographic keys. Cryptographic modules with limited physical security appear to be at greatest risk. Proper selection of physical security features may be used to reduce the risk of this attack.

4. **TEMPEST:** TEMPEST attacks involve the remote or external detection and collection of the electromagnetic signals emitted from a cryptographic module and associated equipment during processing. Such an attack can be used to obtain keystroke information, messages displayed on a video screen, and other forms of critical security information (e.g., cryptographic keys). Special shielding of all components, including network cabling, is the mechanism used to reduce the risk of such an attack. Shielding reduces and, in some cases, prevents the emission of electromagnetic signals. If a cryptographic module is designed to mitigate one or more specific attacks, then the modules security policy shall specify the security mechanisms employed by the module to mitigate the attack(s). The existence and proper functioning of the security mechanisms will be validated when requirements and associated tests are developed.

Here we will be interested mainly in Differential Power Analysis (DPA) as it applies to DES however we will have a brief look at Timing attacks. Both attacks were developed by Paul Kocher of cryptographic research (www.cryptography.com) and you can find his papers on his website.

13.3.1 Differential Power Analysis

Power Analysis is a relatively new concept but has proven to be quite effective in attacking smartcards and similar devices[2]. It was first demonstrated by Ernst Bovelander in 1997 but a specific attack strategy was not given. A year later it was brought to the general public's attention by Paul Kocher and the Cryptographic Research team in San Francisco. Kocher et al. provided an attack strategy that would recover the secret key from cryptographic systems running the DES algorithm. This caused great concern amongst the smartcard community and a search for an effective countermeasure began. To date a limited number of countermeasures have been proposed and none are fully effective. The attacks work equally well on other cryptographic algorithms as shown by Thomas Messerges et al. who presented a great deal of supplementary research on the subject.

Power analysis involves an analysis of the pattern of power consumed by a cryptographic module as it performs its operations. The purpose of this pattern analysis is to acquire knowledge about causal operations that is not readily available through other sources. The power consumption will generally be different for each operation performed (and even for the same operations with different data values). One of the

[2]The smartcard is very susceptible to this form of attack mainly because it applies little or no power filtering due to its small size.

causes of these variations is the transistor technology used to implement the module. The transistors act as voltage controlled switches, and the power they consume varies with the type of instructions being processed. For example, a conditional branch instruction appears to cause a lot of noticeable fluctuations according to Kocher, and should therefore be avoided if possible where secret keys are concerned.

An example of a setup for a power analysis attack is shown in figure 13.3. For smartcards and similar devices, the power can be measured across a $10 - 50\Omega$ resistor[3] in series with the power or ground line of the specific device. It is better to put the resistor in series with the ground of the device as the oscilloscope measures voltages with reference to ground. Therefore the attacker only needs to measure one side of the resistor. If the power line is used then two scope probes would be needed and the resultant waveforms substracted.

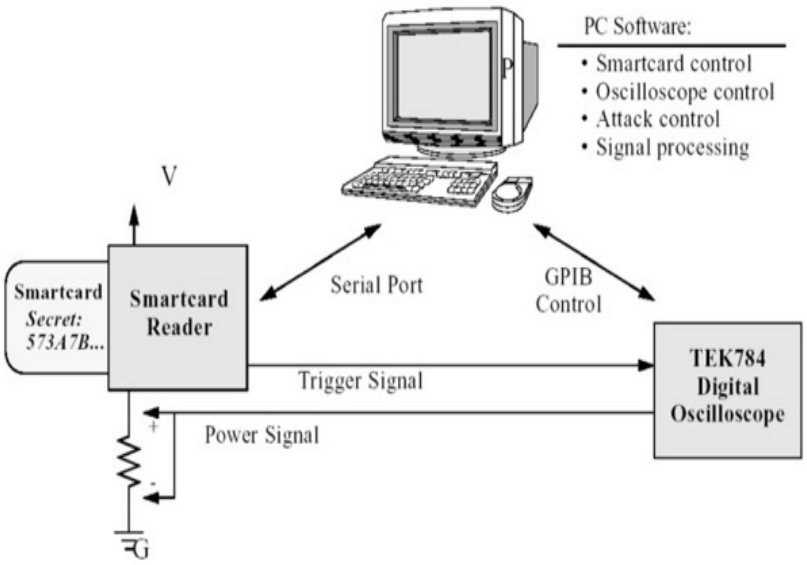

Figure 13.3: An example setup for a Differential Power Analysis attack on a smartcard.

Although the setup in figure 13.3 will suffice for a smartcard it will generally not be this simple for a complex cryptographic accelerator which probably draws its power from the peripheral component interconnect (PCI) backplane of a computer. Ideally, the attacker would wish to get as close as possible to the actual chip performing the operations if a high signal to noise ratio (SNR) is to be obtained. This might be more difficult than it first appears as information on which of the boards numerous chips is actually running the algorithm may not be readily available. Even if it were, the power pin of the chip would have to be physically separated from the board to perform the attack and then reattached once complete (if the attack were to go unnoticed). Most

[3]The resistor should be small enough so as not to interfere with the operation of the circuit itself, but large enough to give easily observable voltage fluctuations.

tamper resistant devices would not permit this from happening.

An example of a possible setup is shown in figure 13.4. In this case a PCI extender board is used to measure the power fluctuations. The actual cryptographic board slots into the extender board and therefore the power the cryptographic board draws from the PCI backplane has to flow through the extender board which can be fitted with some points that allow for measurement of the power. These can be home made or easily purchased.

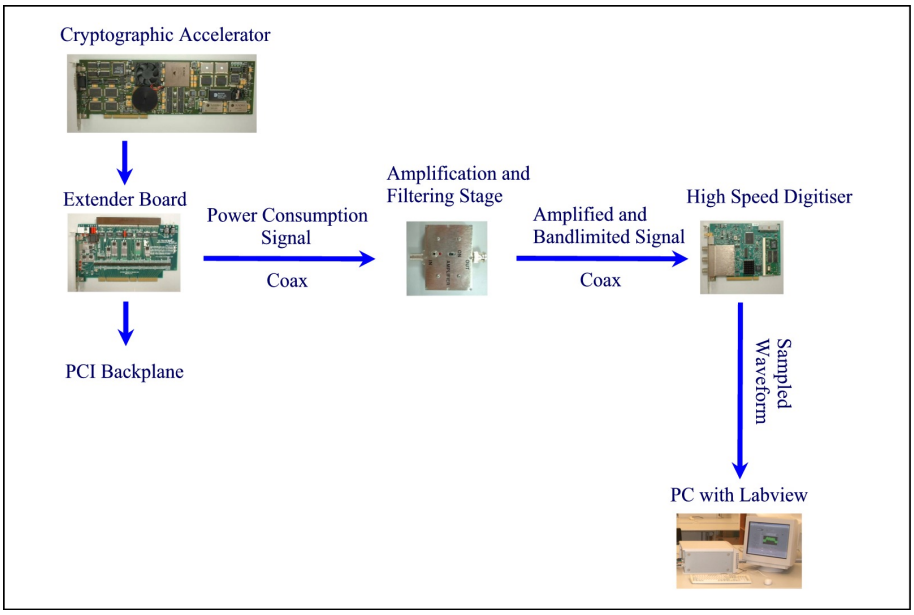

Figure 13.4: An example setup for a Differential Power Analysis attack on a high speed cryptographic accelerator.

Assuming a setup such as those in figures 13.3 and 13.4 in which the algorithm being executed is the Data Encryption Standard (DES) the attack can proceed as follows. A method must be devised to produce a random set of J plaintext inputs that can be sent to the cryptosystem for encryption[4]. On receiving these plaintext inputs, pi_j, $1 \leq j \leq J$, the board will begin to run its algorithm and draw varying amounts of power. These power fluctuations can be sampled using a digital sampling oscilloscope which should be capable of sampling at about 20-30 times the clock frequency being used. There are two main reasons for this:

1. Just because the clock frequency is a certain value, it is possible that we might have multiple operations occuring in each clock cycle. Also, the operation we are interested in might begin on the rising edge of a clock cycle but could only last a small fraction of the clock cycle itself.

[4]This method must be automated as the number of random plaintext inputs will be quite large. Generally this will be the job of the PC however on more complex cryptosystems it may be possible to upload new firmware that will do the trick.

2. The more samples you have per cycle the less chance of noise caused by a misalignment of samples. Ideally what is required is that the samples at $t = 0$ etc. (where t is the time of the sample) line up exactly with one another however, this may not be the case due to fluctuations in the triggering point of the waveform (when the samples are being acquired).

The waveforms observed for each pi_j can be represented as a matrix wf_{jk}[5], where $1 \leq k \leq K$. A second column matrix, co_j, can also be used to represent the ciphertext output. In practice, each row of wf_{jk} would probably be stored as a separate file for ease of processing. Having captured each power waveform and ciphertext output, a function known as a *partitioning function*, D(.), must now be defined. This function will allow division of the matrix wf_{jk} into two sub-matrices $wf0_{pk}$ and $wf1_{qk}$ containing P and Q rows respectively, with $1 \leq p \leq P$ and $1 \leq q \leq Q$ where $P + Q = J$. Provided that the inputs pi_j were randomly produced, then $P = Q = J/2$ as $J \rightarrow \infty$ (i.e. the waveforms will be divided equally between the two sets).

The partitioning function allows the division of wf_{jk} because it calculates the value of a particular bit, at particular times, during the operation of the algorithm. If the value of this bit is known, then it will also be known whether or not a power bias should have occurred in the captured waveform. For a 1, a bias should occur, and for a 0 it shouldn't. Separating the waveforms into two separate matrices (one in which the bias occurred and another in which it didn't) will allow averaging to reduce the noise and enhance the bias (if it occurred). For randomly chosen plaintexts, the output of the D(.) function will equal either a 1 or 0 with probability $\frac{1}{2}$ (this is just another statement of the fact that $P = Q = J/2$ as $J \rightarrow \infty$).

An example of a partitioning function is:

$$D(C_1, C_6, K_{16}) = C_1 \oplus SBOX1(C_6 \oplus K_{16}) \tag{13.8}$$

where SBOX1(.) is a function that outputs the target bit of S-box 1 in the last round of DES (in this case it's the first bit), C_1 is the one bit of co_j that is exclusive OR'ed with this bit, C_6 is the 6 bits of co_j that is exclusive OR'ed with the last rounds subkey and K_{16} is the 6 bits of the last round's subkey that is input into S-box 1.

The value of this partitioning function must be calculated at some point throughout the algorithm. So, if the values C_1, C_6 and K_{16} can be determined, it will be known whether or not a power bias occurred in each waveform. It is assumed that the values C_1 and C_6 can be determined and the value of the subkey K_{16} is the information sought. To find this, an exhaustive search needs to be carried out. As it is 6 bits long, a total of $2^6 = 64$ subkeys will need to be tested. The right one will produce the correct value of the partitioning bit for every plaintext input. However, the incorrect one will only produce the correct result with probability $\frac{1}{2}$. In this case, the two sets $wf0_{pk}$ and $wf1_{qk}$

[5]The subscripts j and k are used to identify the plaintext number causing the waveform and the time sample point within that particular waveform, respectively.

will contain a randomly[6] distributed collection of waveforms which will average out to the same result. The differential trace (discussed below) will thus show a power bias for the correct key only. Of course it means that 64 differential traces are needed but this is a vast improvement over a brute force search of the entire 56 bit key.

Mathematically, the partitioning of wf_{jk} can be represented as

$$wf0_{pk} = \{wf_{jk} | D(.) = 0\} \tag{13.9}$$

and

$$wf1_{qk} = \{wf_{jk} | D(.) = 1\} \tag{13.10}$$

Once the matrices $wf0_{pk}$ and $wf1_{qk}$ have been set up, the average of each is then taken producing two waveforms $awf0_k$ and $awf1_k$ both consisting of K samples. By taking the averages of each, the noise gets reduced to very small levels but the power spikes in $wf1_{pk}$ will be reinforced. However, averaging will not reduce any periodic noise contained within the power waveforms and inherent to the operations on the cryptographic board. This can largely be eliminated by subtracting $awf0_{pk}$ from $awf1_{qk}$ (this can be thought of as demodulating a modulated signal to reveal the "baseband", where the periodic noise is the "carrier"). The only waveform remaining will be the one with a number of bias points identifying the positions where the target bit was manipulated. This trace is known as a *differential trace*, ΔD_k.

Again, in mathematical terms, the above can be stated as

$$awf0_k = \frac{1}{P} \sum_{wf_{jk} \in wf1} wf_{jk} = \frac{1}{P} \sum_{p=1}^{P} wf0_{pk} \tag{13.11}$$

and

$$awf1_k = \frac{1}{Q} \sum_{wf_{jk} \in wf0} wf_{jk} = \frac{1}{Q} \sum_{q=1}^{Q} wf1_{qk} \tag{13.12}$$

The differential trace ΔD_k is then obtained as

$$\Delta D_k = awf1_k - awf0_k \tag{13.13}$$

The last five equations can now be condensed into one:

[6]Provided the plaintext inputs are randomly chosen.

$$\Delta D_k = \frac{\sum_{k=1}^{K} D(.)wf_{jk}}{\sum_{k=1}^{K} D(.)} - \frac{\sum_{k=1}^{K}(1 - D(.))wf_{jk}}{\sum_{k=1}^{K}(1 - D(.))} \tag{13.14}$$

As $J \rightarrow \infty$, the power biases will average out to a value ϵ which will occur at times k_D - each time the target bit D was manipulated. In this limit, the averages $awf0_k$ and $awf1_k$ will tend toward the expectation $E\{wf0_k\}$ and $E\{wf1_k\}$, and equations 13.13 and 13.14 will converge to

$$E\{wf1_k\} - E\{wf0_k\} = \epsilon, \qquad \text{at times} \quad k = k_D \tag{13.15}$$

and

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \text{at times} \quad k \neq k_D \tag{13.16}$$

Therefore, at times $k = k_D$, there will be a power bias ϵ visible in the differential trace. At all other times, the power will be independent of the target bit and the differential trace will tend towards 0.

The above will only work if the subkey guess was correct. For all other guesses the partitioning function will separate the waveforms randomly, and equations 13.15 and 13.16 will condense to

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \forall k \tag{13.17}$$

As mentioned above, 64 differential traces are needed to determine which key is the correct one. Theoretically, the one containing bias spikes will allow determination of the correct key however, in reality the other waveforms will contain small spikes due to factors such as non-random choices of plaintext inputs, statistical biases in the S-boxes and a non-infinite number of waveforms collected. Generally however, the correct key will show the largest bias spikes and can still be determined quite easily.

The other 42 bits from the last round's subkey can be determined by applying the same method to the other 7 S-boxes[7]. A brute force search can then be used to obtain the remaining 8 bits of the 56 bit key.

13.3.1.1 Mitigation Techniques

The following could be used as mitigation techniques for power attacks in general:

1. Timing Randomisation: This involves placing random time delays into the software so that a power analysis will not be possible. With random delays introduced, a steady trigger will not be sufficient to allow the averaging to work and

[7]The same J power signals can be used for each S-box as the different D functions re-order them accordingly.

will therefore act as a countermeasure.

2. Internal power supplies/power supply filtering: This would be another method that could be used to reduce the possibility of a power attack. For example, Adi Shamir proposes building a simple capacitance network into each smartcard to allow the fluctuations to be contained within the smartcard itself thereby preventing power attacks.

3. Data masking: One of the methods proposed consists of *masking* the intermediate data (i.e. mask the input data and key before executing the algorithm). This would make the power fluctuations independent of the actual data.

4. Tamper Resistance: This involves placing some detection/prevention system around the device to stop intruders gaining access to the power fluctuations.

5. Fail Counters: A differential power analysis attack requires the attacker to obtain a significant number of power waveforms. In order to do this the attacker must have the ability to run quite a few encryptions on the system under attack. If the number of encryptions were limited to a certain number then the attacks would become increasingly difficult.

6. Removal of conditional elements: One of the main features used to attack the square and multiply algorithm is the fact that it has a conditional multiplication that depends on the value of the exponent bit being operated upon. One suggested countermeasure is to implement this multiplication in every round (regardless of the value of the bit) and to only do a register update when the bit is a 1.

13.4 Timing Attacks

A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. We can explain the attack using the modular exponentiation algorithm shown in figure 13.5, but the attack can be adapted to work with any implementation that does not run in fixed time. In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

As Kocher points out in his paper, the attack is simplest to understand in an extreme case. Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes musch more time than an entire average modular exponentiation. The attack proceeds bit by bit starting with the leftmost bit $e[N-1]$. Suppose that the first j bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known). For a given ciphertext, the attacker can complete the first j iterations of the **for** loop. The operation of the subsequent steps depends on the unkown exponent bit. If the bit is set d = (d $\times$ b)

```
square_and_mul(b, e, m)
{
   d = 1;
   for (k = N-1 downto 0)
   {
      d = (d × d) mod m;
      if (e[k] == 1)
      {
       d = (d × b) mod m;
      }
   Return d;
}
```

Figure 13.5: Square and Multiply algorithm for Computing $b^e \bmod (m)$ where e is N bits long.

mod m will be executed. For a few values of b and d, the modular multiplication will be extremely slow, and the attacker knows which these are. Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is slow with a 1 bit, then this bit is assumed to be 1. If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.

In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm. Nevertheless, there is enough variation to make this attack practical.

Although the timing attack is a serious threat, there are simple counter measures that can be used including the following:

- **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
- **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher point out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.
- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analsys essential to the timing attack.

RSA Data Security incorportates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows:

1. Generate a secret random number r between 0 and $n - 1$.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.
3. Comput $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M = M'r^{-1} \bmod n$ (where r^{-1} is the multiplicative inverse of $r \bmod n$). It can be demonstrated that this is the correct result by observing that $r^{ed} \bmod n = r \bmod n$.

RSA Data Security reports a 2 to 10% performance penalty for blinding.