

## **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**.
  - 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.
-

## **Differential Power Analysis**

- Power Analysis is a relatively new concept but has proven to be quite effective in attacking smartcards and similar devices.
  - The smartcard is very susceptible to this form of attack mainly because it applies little or no power filtering due to its small size.
  - 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 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 3.
-

- For smartcards and similar devices, the power can be measured across a  $10 - 50\Omega$  resistor in series with the power or ground line of the specific device.
  - 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. It is better to put the resistor in series with the ground of the device.
  - If the power line is used then two scope probes would be needed and the resultant waveforms subtracted.
-

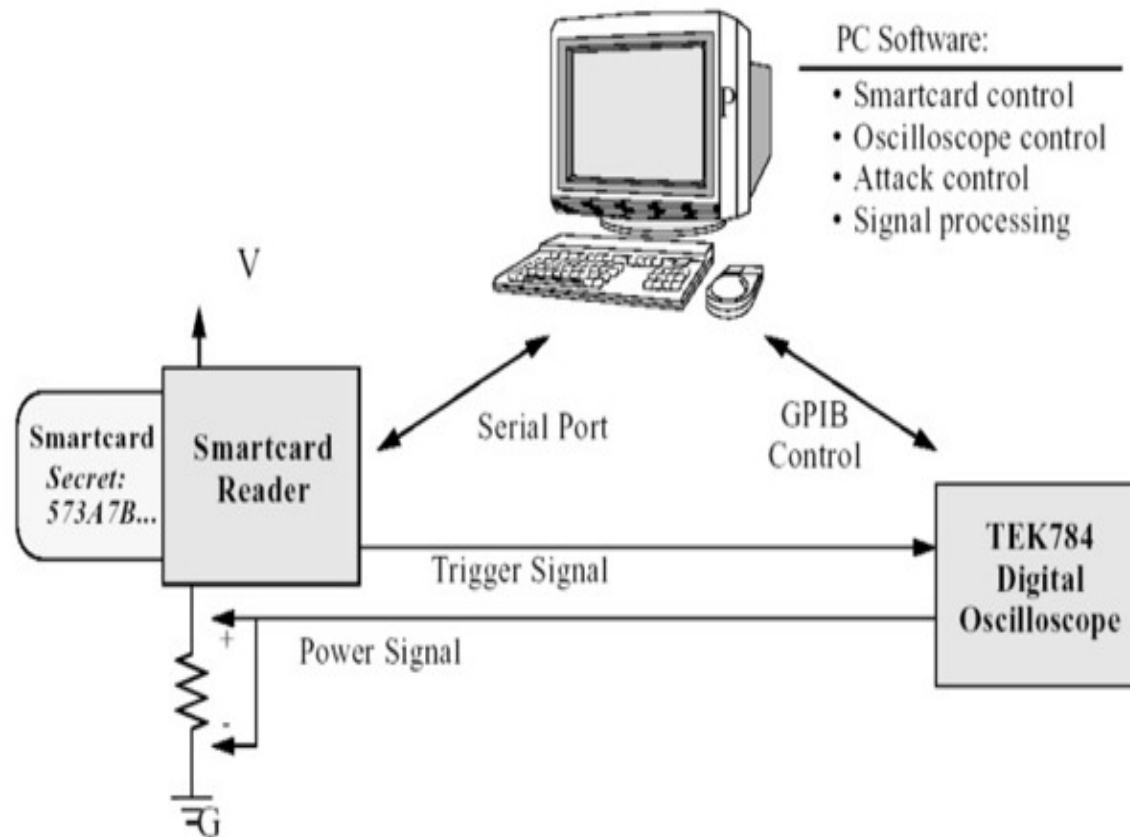


Figure 3: An example setup for a Differential Power Analysis attack on a smartcard.

---

- Although the setup in figure 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 tamper resistant devices would not permit this from happening.
  - An example of a possible setup is shown in figure 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.
-



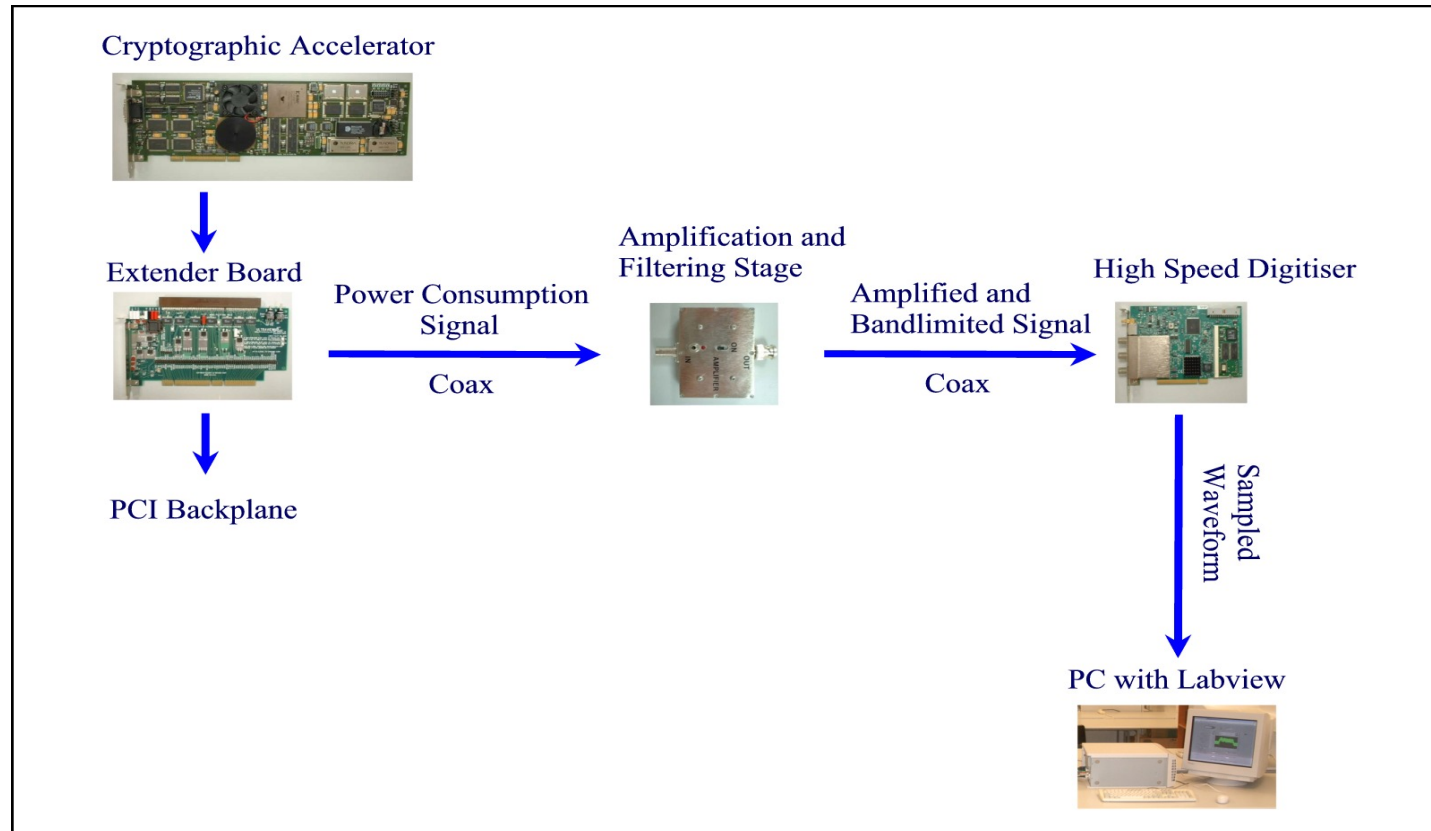


Figure 4: An example setup for a Differential Power Analysis attack on a high speed cryptographic accelerator.

- Assuming a setup such as those in figures 3 and 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.
  - 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.
  - 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. Possible that we might have multiple operations occurring in each clock cycle. Also, operation of interest might only last a small fraction of the clock cycle.
    2. The more samples you have per cycle the less chance of noise caused by a misalignment of samples.
  - The waveforms observed for each  $pi_j$  can be represented as a matrix  $wf_{jk}$ , where  $1 \leq k \leq K$ .
  - 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.

- 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**,  $D(\cdot)$ , 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:

$$D(C_1, C_6, K_{16}) = C_1 \oplus SBOX1(C_6 \oplus K_{16}) \quad (8)$$

- 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  $wf1_{qk}$  will contain a randomly
-

distributed collection of waveforms which will average out to the same result (Provided the plaintext inputs are randomly chosen).

- 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

$$wf0_{pk} = \{wf_{jk} | D(.) = 0\} \quad (9)$$

and

$$wf1_{qk} = \{wf_{jk} | D(.) = 1\} \quad (10)$$

---



- 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**,  $\Delta D_k$ .
- Again, in mathematical terms, the above can be stated as

$$awf0_k = \frac{1}{P} \sum_{wf_{jk} \in wf1} wf_{jk} = \frac{1}{P} \sum_{p=1}^P wf0_{pk} \quad (11)$$

and

$$awf1_k = \frac{1}{Q} \sum_{wf_{jk} \in wf0} wf_{jk} = \frac{1}{Q} \sum_{q=1}^Q wf1_{qk} \quad (12)$$

---

- The differential trace  $\Delta D_k$  is then obtained as

$$\Delta D_k = awf1_k - awf0_k \quad (13)$$

- The last five equations can now be condensed into one:

$$\Delta D_k = \frac{\sum_{j=1}^J D(.)wf_{jk}}{\sum_{j=1}^J D(.)} - \frac{\sum_{j=1}^J (1 - D(.))wf_{jk}}{\sum_{j=1}^J (1 - D(.))} \quad (14)$$

- As  $J \rightarrow \infty$ , the power biases will average out to a value  $\epsilon$  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 and 14 will
-

converge to

$$E\{wf1_k\} - E\{wf0_k\} = \epsilon, \quad \text{at times } k = k_D \quad (15)$$

and

$$E\{wf1_k\} - E\{wf0_k\} = 0, \quad \text{at times } k \neq k_D \quad (16)$$

- Therefore, at times  $k = k_D$ , there will be a power bias  $\epsilon$  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 15 and 16 will condense to

$$E\{wf1_k\} - E\{wf0_k\} = 0, \quad \forall k \quad (17)$$

---

- 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.
-

- A brute force search can then be used to obtain the remaining 8 bits of the 56 bit key.
  - NOTE: The same  $J$  power signals can be used for each S-box as the different D functions re-order them accordingly.
-

## **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 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.
-

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

---



- 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 much 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.
  - Operation of subsequent steps depends on the unknown exponent bit.
-

- If the bit is set  $d = (d \times b) \bmod m$  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 analysis essential to the timing
-

attack.

- RSA Data Security incorporates 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. Compute  $M' = (C')^d \bmod n$  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.
-