Experiment 9
Side Channel Attacks

We will implement two side channel attacks to get the keys: Simple Power Analysis (SPA) and Differential Power Analysis (DPA) on DES.

Theory Background

1. Side channel attack techniques

In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (compare cryptanalysis). For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information, which can be exploited to break the system. Some side-channel attacks require technical knowledge of the internal operation of the system on which the cryptography is implemented, although others such as differential power analysis are effective as black-box attacks. Many powerful side-channel attacks are based on statistical methods pioneered by Paul Kocher.

1.1 Ways of accessing the module

When analyzing the security of a cryptographic hardware module, it can be useful to perform a systematic review of the attack surface — the set of physical, electrical and logical interfaces that are exposed to a potential opponent. According to this observation, side channel attacks can be divided into the following classes: invasive attacks, semi-invasive attacks, and non-invasive attacks.

An Invasive attack involves de-packaging to get direct access to the internal components of cryptographic modules or devices. A typical example of this is that the attackers may open a hole in the passivation layer of a cryptographic module and place a probing needle on a data bus to see the data transfer. The concept of semi-invasive attack is first developed by Skorobogatov and Anderson. This kind of attack involves access to the device, but without damaging the passivation layer or making electrical contact other than with the authorized surface. For example, in a fault-induced attack, the attacker may use a laser beam to ionize a device to change some of its memories and thus change the output of this device. A non-invasive attack involves close observation or manipulation of the device's operation. This attack only exploits externally available information that is often unintentionally leaked. A typical example of such an attack is timing analysis: measuring the time consumed by a device to execute an operation and correlating this with the computation performed by the device in order to deduce the value of the secret keys.

1.2 Known side channel attacks

1.2.1 Power attacks

In this experiment, we will focus on power attacks. Power attacks measure the circuit's processing time and current consumption to infer what is going on inside it. Of all types of SCA attacks known today, the number of literature on power analysis attacks and the relevant countermeasures is the biggest. Power analysis attack is the current research focus of side-channel attacks. Power analysis attacks have been demonstrated to be very powerful attacks for most straightforward implementations of symmetric and public key ciphers.

Basically, power analysis attack can be divided into SPA and DPA. In SPA attacks, the aim is essential to guess from the power trace which particular instruction is being executed at a certain time and what values the input and output have. Therefore, the adversary needs an exact knowledge of the implementation to mount such an attack. On the other hand, DPA attack does not need the knowledge of the implementation details and alternatively exploiting statistical methods in the analysis process. DPA is one of the most powerful SCA attacks, yet it can be mounted using very little resources.

1.2.2 Timing attacks

A timing attack is, essentially, a way of obtaining some user's private information by carefully measuring the time it takes the user to carry out cryptographic operations. The principle of this attack is very simple: to exploit the timing variance in the operation.

Timing attacks were introduced in 1996 by Kocher, where RSA modular exponentiation was being attacked. The basic assumptions of timing analysis are 1. The runtime of a cryptographic operation depends to some extent on the key. With present hardware, this is likely to be the case, but note that there are various efficient hardware-based proposals to make the timing attack less feasible through 'noise injection'. Software approaches to make the timing attack infeasible are based on the idea that the computations in two branches of a conditional should take the same amount of time ('branch equalization'). 2.A sufficiently large number of encryptions can be carried out, during which time the key does not change. A challenge-response protocol is ideal for timing attacks. 3.Time can be measured with known error. The smaller the error, the fewer time measurements are required.

1.2.3 EM attacks

As electrical devices, the components of a computer often generate electromagnetic radiation as part of their operation. An adversary that can observe these emanations and can understand their causal relationship to the underlying computation and data may be able to infer a surprising amount of information about this computation and data. This ability can be devastating, should the computer be a trusted computing platform intended to keep this information from the adversary. Similar to the power analysis attacks, Electromagnetic Analysis attacks can also be divided into two main categories: Simple Electromagnetic Analysis and Differential Electromagnetic Analysis.

2. SPA

SPA is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material.

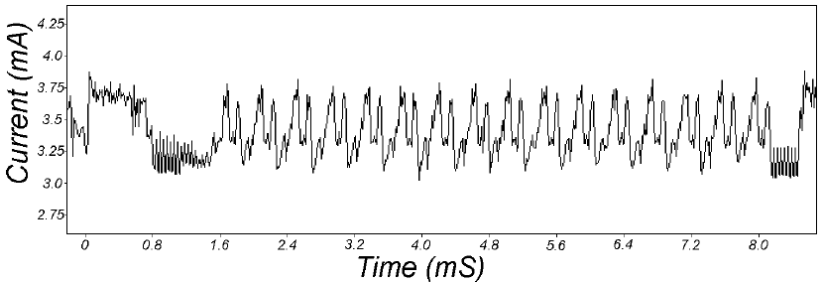

Figure 1 SPA trace showing an entire DES operation.

A trace refers to a set of power consumption measurements taken across a cryptographic operation. For example, a 1-millisecond operation sampled at 5 MHz yields a trace containing 5000 points. Figure 1 shows a SPA trace from a typical smart card as it performs a DES operation. Note that the 16 DES rounds are clearly visible.

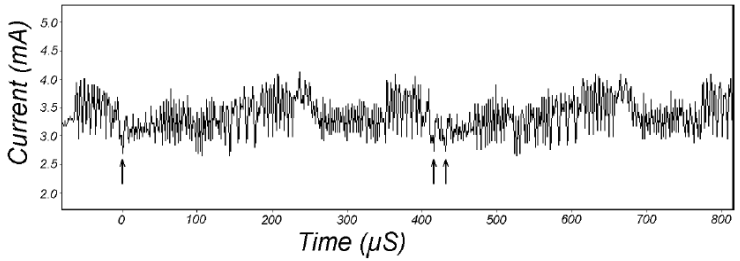

Figure 2 SPA trace showing DES rounds 2 and 3.

Figure 2 is a more detailed view of the same trace showing the second and third rounds of a DES encryption operation. Many details of the DES operation are now visible. For example, the 28-bit DES key registers C and D are rotated once in round 2 (left arrow) and twice in round 3 (right arrows). In Figure 2, small variations between the rounds just can be perceived. Many of these discernable features are SPA weaknesses caused by conditional jumps based on key bits and computational intermediates.

3. DPA

In addition to large-scale power variations due to the instruction sequence, there are effects correlated to data values being manipulated. These variations tend to be smaller and are sometimes overshadowed by measurement errors and other noise. In such cases, it is still often possible to break the system using statistical functions tailored to the target algorithm.

In an acquisition campaign, *m* power curves $\mathbf{T}_i$ (also casually called "traces") are garnered. Each trace $\mathbf{T}_i$ corresponds to a given leakage. For the sake of simplicity, we focus on a part of the leakage, related to a very restricted set of gates in the netlist. We assume this leakage can be summarized into two typical behaviors: high dissipation to charge a net versus low dissipation to discharge it, and activity versus no-activity.

These two behaviors are related to an inner Boolean variable, called a decision (selection) function D_i. If the secret key is unknown, this variable is also unknown; however, it depends in practice on only a couple of bits from the key, which can be tested exhaustively. The decision function $D_i \in \{0,1\}$ is an oracle for an attacker: if it is the correct decision function, it can allow for an exhibition of the two behaviors, otherwise it is simply irrelevant. The idea behind DPA is to exhibit an asymptotic difference between the behaviors. The ad hoc criterion is introduced as:

$$\mathrm{DPA} = \frac{1}{m_0}\sum_{i/D_i=0} \mathrm{T}_i - \frac{1}{m_1}\sum_{i/D_i=1} \mathrm{T}_i \ldots\ldots(1)$$

Where m_0 and m_1 denote the number of traces for each decision.

In [3], Paul Kocher defined the selection function $D(C, b, K_s)$ as computing the value of bit $0 \leq b \leq 32$ of the DES intermediate L at the beginning of the 16th round for ciphertext C, where the 6 key bits entering the S box corresponding to bit b are represented by $0 \leq K_s \leq 2^6$. Note that if K_s is incorrect, evaluating $D(C, b, K_s)$ will yield the correct value for bit b with probability $P \approx \frac{1}{2}$ for each ciphertext.

To implement the DPA attack, an attacker first observes *m* encryption operations and captures power traces $\mathbf{T}_{1..m}[1..k]$ containing *k* samples each. In addition, the attacker records the ciphertexts $C_{1..m}$. No knowledge of the plaintext is required.

DPA analysis uses power consumption measurements to determine whether a key block guess K_s is correct. The attacker computes a *k*-sample differential trace $\Delta_D[1..k]$ by finding the difference between the average of the traces for which $D(C_i, b, K_s)$ is one and the average of the traces for which $D(C, b, K_s)$ is zero. Thus $\Delta_D[j]$ is the average over $C_{1..m}$ of the effect due to the value represented by the selection function *D* on the power consumption measurements at point *j*.

If K_s is incorrect, the bit computed using *D* will differ from the actual target bit for about half of the ciphertexts C_i. The selection function $D(C_i, b, K_s)$ is thus effectively uncorrelated to what was actually computed by the target device. If a random function is used to divide a set into two subsets, the difference in the averages of the subsets should approach zero as the subset sizes approach infinity.

If K_s is correct, however, the computed value for $D(C_i, b, K_s)$ will equal the actual value of target bit *b* with probability 1. The selection function is thus correlated to the value of the bit manipulated in the 16th round. As a result, the $\Delta_D[j]$ approaches the effect of the target bit on the power consumption as $m \to \infty$. Other data values, measurement errors, etc. that are not correlated to *D* approach zero. Because power consumption is correlated to data bit values, the plot of Δ_D will be at with spikes in regions where *D* is correlated to the values being processed.

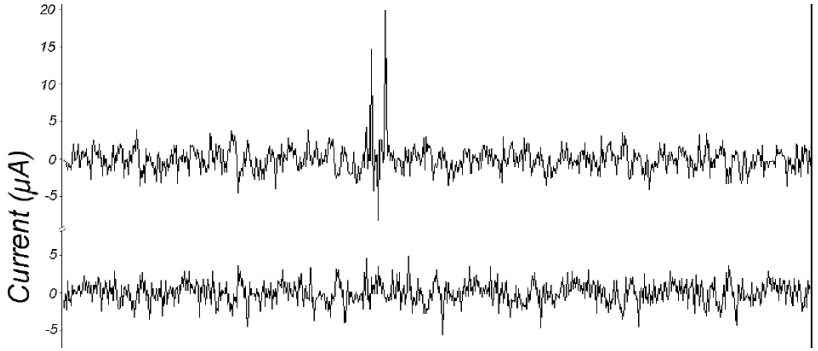

Figure 3 DPA traces: correct guess (top) and incorrect guess (bottom).

Experiment Set-up: Configuration

The hardware and software needed for this experiment include:

1. The HAHA Board and USB Blaster.
2. An Oscilloscope (Analog Discovery 2 for Edge students).
3. A computer.
4. Quartus to program the FPGA.
5. Matlab to process data.

For Edge students, use the Analog Discovery 2 instead of oscilloscope so that you can do the measurements at home. Analog Discovery has 2 analog channels and 16 digital channels. For this experiment, use the analog channels when you are measuring the generated clock waveform. As your 'oscilloscope" does not have probes but jump wires, you should connect the wires correctly on the HAHA Board. The only place for your connection is the jumper "JP1" on the board. Refer to the instructions on the next page and connect the pins in the same way. Analog Discovery 2 can reach the range 100uV/div, which is more than enough for this experiment, so choose the suitable range in the WaveForms software.

Instructions and Questions

```
SquareMult(x, e, N):
  let e_n, ..., e_1 be the bits of e
  y ← 1
  for i = n down to 1 {
    y ← Square(y)                (S)
    y ← ModReduce(y, N)          (R)
    if e_i = 1 then {
      y ← Mult(y, x)             (M)
      y ← ModReduce(y, N)        (R)
    }
  }
  return y
```

Figure 4 Square and multiply algorithm

PART I Implement a SPA

In this part, we will implement a SPA on square and multiply algorithm, which is often used in DES to implement modular exponentiation functions. The algorithm to calculate "$x^e \bmod N$" is shown in Figure 4. We can see that there is an if-statement controlled by the value of *e*. If e_i (i^{th} bit of e) is zero, only two steps will be needed: Square and ModReduce. Otherwise, if e_i is one, four steps will be needed: Square, ModReduce, Mult and another ModReduce. Therefore, in the case when e_i is one, the algorithm tends to consume more power (or time, in some cases).

We can implement the algorithm into the FPGA and measure how much power it consumes. Figure 5 shows the sub-circuitry for the HAHA board. You can find **CS1**, which is a very precise ***small*** resistor on the board. The current that consumed by the core of the FPGA will all go through the resistor. Therefore, the more power the FPGA consumes, the voltage drop across the resistor will be bigger.

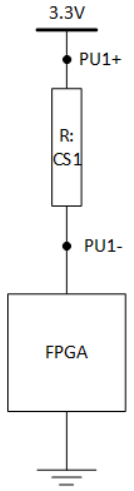

Figure 5 Circuitry

Download the code for square and multiply algorithm from the Canvas. *mul.v* is the top module. *other.v* contains all the sub-modules. Instantiate a RAM by yourself as you did before. Change the key (value of *e*) to see how the power consumption waveform changes. Measuring the voltage difference as shown in Figure 7 and never do what is shown in Figure 6! An example: when the key value is 8'b00001111, the waveform will look like Figure 8.

Figure 6 Wrong way of measuring

Figure 7 Right way of measuring

1.Why should we never do what is shown in Figure 6?
2.Turn in the waveform when e is 8'b01110001.
3.Try the bitstream files 1-A.sof, 1-B.sof, and 1-c.sof. Turn in the waveform and guess what value *e* has for each of them.

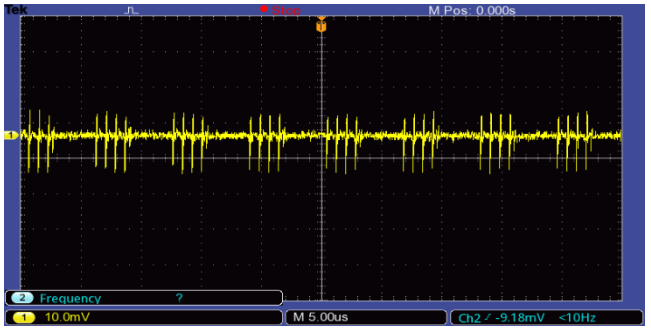

Figure 8 Waveform when e is 8'b00001111 .

PART II Implement a DPA on DES

In this part, you are required to do some measurement, but you will not process the data you measured. To do a DPA, from hundreds to tens of thousands of measurements are required. We don't have time to do that many measurements in the lab. Therefore, we will process the data provided on canvas. There are in total 80000 traces provided. Don't be frightened by the number. To get the key through DPA, you need far less than that. (Actually, there is little difference between 100 and 80000: you can calculate neither of them *by hand*.)

The provided traces are very similar to what is shown in Figure 1. They are the power consumption traces that include an entire DES operation of 16 rounds. The files are named with the plaintext (the first part) and the ciphertext (the second part). If you load (in Matlab) one of the files and plot its data, it will look like the waveform in Figure 9. As these 80000 traces are obtained when the DES are encrypting different plaintext (with the same key), the waveforms are different from each other. However, you can hardly tell the difference by looking with your eyes. If you do subtraction between two traces, you can see they can almost cancel out each other, as shown in Figure 10.

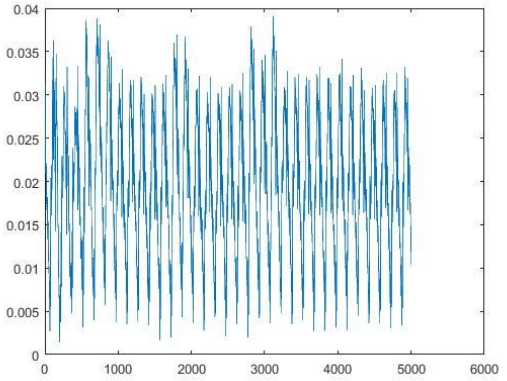

Figure 9 Trace example.

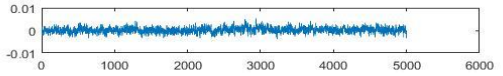

Figure 10 The difference of two traces.

The first step of DPA is to divide these traces into two groups using your selection function. Then, you calculate the average of the traces in both groups. These two average traces may still look very similar to the trace in Figure 9. However, when you calculate the difference between these two average traces and plot out the difference, an exciting thing happen. If you divided the traces using the right way in the first step, you can see that these two average traces ***cannot*** cancel out each other anymore. There will be a peak (Figure 11).

The key point is how you divide these traces into two groups. If you divide them randomly and calculate the difference of the average trace for the two groups, they cancel out each out. If you divide them using the wrong way, there will be no difference with the random way: still, cancel out each out. You should design a good selection function which, if the key guess is right, can divide the traces in a correct and efficient way, so that the two average traces cannot cancel each other.

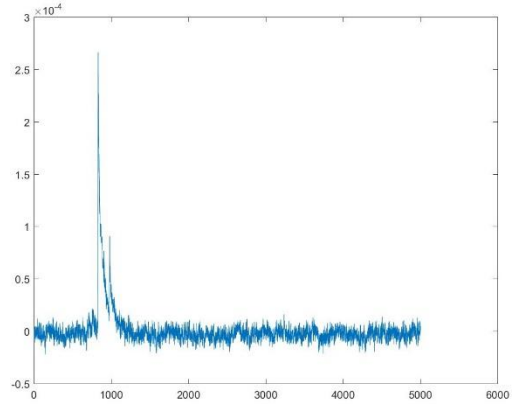

Figure 11 Difference of two average traces when the key guess is correct.

How do we design the selection function? We know that the DES does the same operation for 16 rounds and the weakness of DES is always in the first round or the last round. (Why?) In this experiment, we can try to get the sub-key for the first round. Once we know a sub-key for one round, we can calculate out the original key. (Why?)

We also know that semiconductors use current while switching. We can concentrate on one bit of one register (register R shown in Figure 12 is recommended). If it flips during the first round, we put the trace into one group. If it doesn't flip during the first round, we put it into the other group. Therefore, after we put all the traces into these two groups correctly, the traces in one group all tend to consume more power (at an unknown but certain time), and there will be a peak (at that certain time) in the difference trace.

Now we can design the selection function. The function will take 3 input: what the plaintext (*M*) is, which bit ($1 \leq b \leq 32$) of register R you choose, and what the guessed sub key ($0 \leq K \leq 63$) is for one sbox of the *f* function in DES. The *M* is given in the name of the files. The *b* can be chosen by you. The *K* is your guess. You should try all the 64 possible sub keys and get all the 64 difference traces. The one with the biggest peak will be the right guess. There are in total 8 different sboxes. We should repeat the whole process 8 times to get the whole 48-bit sub key.

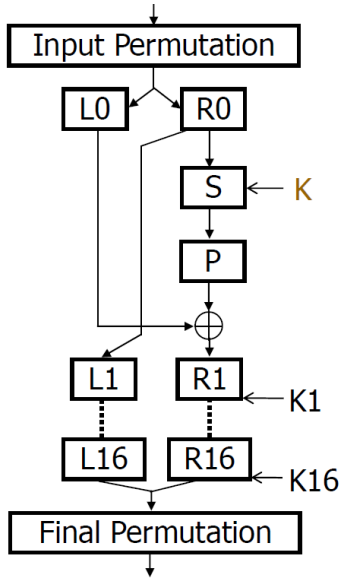

Figure 12 DES operation.

1. (Only this question requires you to do in the lab.) Implement the DES again using the code provided in experiment 5. Measure the power consumption trace containing 16 rounds with an oscilloscope. Turn in the waveform. (This may not look like the provided traces.)
2. First round and last round of encryption algorithms are often targeted by hackers. Why are these rounds often the weakness?
3. Explain how to get the original 56-bit key from a 48-bit sub key.
4. Note that after the 8 sboxes finish their substitution, there will be a permutation (the P box in Figure 12). That means the 4-bit output of the first sbox will not go to the most significant 4 bits of the result, etc. Which bits will the first sbox's output go to? Refer to the DES code to answer this question. (Here 'first' means this sbox takes the 6 most significant bits from the xor result of the sub key. Make sure to answer it correctly. We will need these bits in question 6.)
5. Create a selection function in Matlab. The output of the function is either 0 or 1. If bit *b* stays the same during the first round, the output is 0. If bit *b* flips, the output is 1. Turn in your Matlab script containing the function.
6. Choose one bit from the answer of question 4 to be your *b*. For *K*=0 to 63: use the selection function you created to divide all the traces into two groups. For each *K*, generate the difference trace for the average of the two groups. Which *K* produces the biggest peak? Turn in all the 64 trace pictures for all *K*'s. (If you can get the 6 bit key for one sbox, it's easy for you to get all the 48 bits of the sub key.)
7. (You can keep going to get extra credits.) Get all the 48 bits of the sub key for the first round. You should turn in all the trace pictures to prove you really finish this. 8×64 pictures are needed, name them in a good way.(Optional)
8. From the first-round sub key you get in 7, calculate out the original key. Include the process (or code) to do it. (Optional and extra credit)

☞ Lab Report Guidelines

1. In your report, answer ALL the questions.
2. Give a photo of your experiment set-up.
3. Attach all your code. Describe how do you come up with your code according to the algorithm of DES. Give a screenshot of the result of running the code.
4. Outputs for your code will be a big set of pictures. Include all the pictures, and explain how do you find the key by comparing these pictures.

References and Further Reading

[1] https://en.wikipedia.org/wiki/Exponentiation_by_squaring

[2] https://en.wikipedia.org/wiki/Data_Encryption_Standard

[3] Kocher, Paul, Joshua Jaffe, and Benjamin Jun. "Differential power analysis." Annual International Cryptology Conference. Springer Berlin Heidelberg, 1999.

[4] Zhou, YongBin, and DengGuo Feng. "Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing." IACR Cryptology ePrint Archive 2005 (2005): 388.

[5] Prouff, Emmanuel. "DPA attacks and S-boxes." International Workshop on Fast Software Encryption. Springer Berlin Heidelberg, 2005.

[6] Guilley, Sylvain, et al. "Silicon-level solutions to counteract passive and active attacks." Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on. IEEE, 2008.

[7] Guilley, Sylvain, et al. "Improving side-channel attacks by exploiting substitution boxes properties." International Conference on Boolean Functions: Cryptography and Applications (BFCA). 2007.

[8] Hnath, William. Differential power analysis side-channel attacks in cryptography. Diss. Worcester Polytechnic Institute, 2010.