Experiment 10
Fault Injection Attacks

We describe an experiment to implement an Fault Injection Attack in order to compromise the security of the system.

Theory Background

Fault injection attacks intentionally cause errors in a system in order to compromise the security of the system.

1. Fault injection attack on cryptographic devices

Cryptographic algorithms are being employed in many consumer products, e.g., smart cards, cell phones, and set-top boxes, to meet their high-security requirements. Unfortunately, these hardware modules, unless carefully designed, may result in security vulnerabilities. The cryptographic algorithms that are being implemented, are designed so that they are difficult to be broken mathematically. To obtain the secret key, which allows the decryption of encrypted information, an attacker must perform a brute force analysis that requires a prohibitively large number of experiments. For the most commonly used cryptographic algorithms, there is no known methodology to significantly reduce the secret key search space.

However, it has been shown that secret information (such as the key of the encryption algorithm) can leak through side channels. Examples of such side channels are the time needed to perform the encryption or the power consumed by the device implementing the encryption algorithm. Timing and power side channel attacks are based on the fact that the individual computation steps that are needed during the encryption are dependent on the bits of the secret key and thus, the time needed for these steps and the power consumed by them is directly correlated to the secret key bits. These attacks have proven to be effective and incur a relatively low cost. Furthermore, once a side-channel attack technique has been developed and made public, high technical skills and/or expensive equipment are not required to apply it in practice.

A different type of side-channel attack that proved to be very effective is realized through the injection of deliberate faults into a cryptographic device and the observation of the corresponding erroneous outputs. Using this type of attack and analyzing the outputs of the cryptographic device, called differential fault analysis, the number of experiments needed to obtain the bits of the secret key can be drastically reduced. This kind of active side-channel attacks has been in the last decade the subject of intense and expanding research, as it has been demonstrated to be highly effective.

2. Fault injection techniques

The fault injection techniques that have been developed in order to alter maliciously the correct functioning of a computing device currently include variations in the power supply voltage level, injection of irregularities in the clock signal, radiation or electromagnetic (EM) disturbances, overheating the device or exposing it to intense light.

2.1 Voltage glitching

The first fault injection technique we describe is the underpowering of the device. Through running the chip with a depleted power supply, the attacker can insert transient faults starting from single bit errors and become more invasive as the supply voltage gets lower. Since this technique does not require precise timing, the faults tend to occur uniformly throughout the computation, thus requiring the attacker to be able to discard results that are not fit to lead an attack. This methodology reported being effective on large integrated circuits such as the ARM9 processor, as well as on small application-specific integrated circuit (ASIC) implementations of the ciphers, results in delaying the correct setup for the logic gates of the circuit. The voltage underfeeding, achieved by employing a precise power supply unit, requires the attacker to be able to tap into the power supply line of the device and connect his power supply unit. This requires only basic skills and can be easily achieved in practice without leaving evidence of tampering. Moreover, no knowledge of the implementation details of the device is needed.

2.2 Clock glitching

Another viable option for an attacker is to tamper with the clock signal. For example, it is possible to shorten the length of a single cycle through forcing a premature toggling of the clock signal. Such shortening causes multiple errors corrupting a stored byte or multiple bytes. These errors are transient and thus it is possible to induce such faults without leaving any tamper evidence. To alter the length of the clock cycle, the attacker needs to have direct control over the clock line, which is the typical case when smart cards are targeted. It is not possible to attack chips that generate their own clock signal since disconnecting the clock line from the circuit is difficult.

2.3 EM disturbances

A practical way to induce faults without having to tap into the device is to cause strong EM disturbances near it. The eddy currents induced in the circuit by strong EM pulses cause temporary alterations of the level of a signal, which may be recorded by a latch. Since the EM pulse is affecting uniformly the entire attacked device, it is necessary to shield the components which should not be subject to faults using a properly grounded metal plate or mesh.

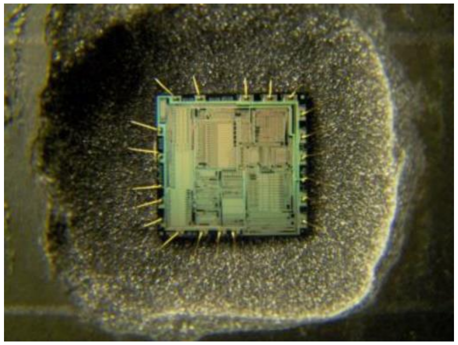

Figure 1 Decapsulated IC

2.4 Flash glitching

Assuming the attacker can successfully decapsulate a chip, he can perform fault injection attacks by illuminating the die with a high energy light source such as an ultraviolet (UV) lamp or a camera flash. The strong radiation directed at the silicon surface can cause the blanking of erasable EPROM and FLASH memory cells where constants needed for an algorithm execution are kept (e.g., the AES S-Boxes). This technique is applicable only if the memory cells have not been covered by a metallic layer. For example, metal wires placed above the memory cells may provide a shield against radiation.

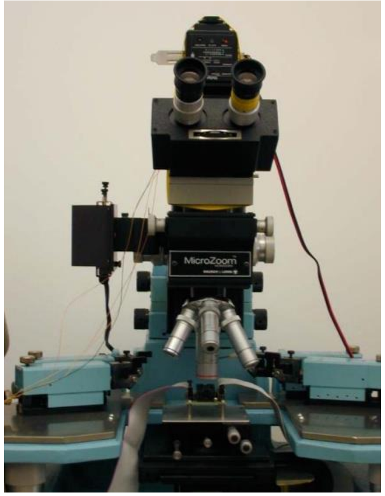

Figure 2 Sample Probing station

2.5 Laser glitching

The most straightforward refinement of the previous technique is to employ a laser beam instead of a camera flash. The injected fault model is similar to that obtained when using a concentrated light beam, except for the fact that the laser beam is capable of always inducing faults. Near-infrared lasers can also radiate the silicon die from the back allowing the attacker to hit circuits which are in the bottom layers of the chip although with a lower precision since the silicon substrate scatters the beam. It is worth noting that the inability to hit only a single-bit memory cell (due to the size of the concentrated beam) does not necessarily imply an inability to inject a single-bit fault.

3. Differential Fault Analysis (DFA) on AES

In October 1996, Biham and Shamir published an attack on secret key cryptosystems entitled Differential Fault Analysis (DFA). It is a type of side channel attack in the field of cryptography, specifically cryptanalysis. The principle is to induce faults (unexpected environmental conditions) into cryptographic implementations, to reveal their internal states. DFA is frequently used nowadays to test the security of cryptographic smartcards applications.

By using a DFA attack where a fault occurs on only one bit of the temporary cipher result at the beginning of the ***Final Round***, the entire last round key for an AES-128 can be obtained.

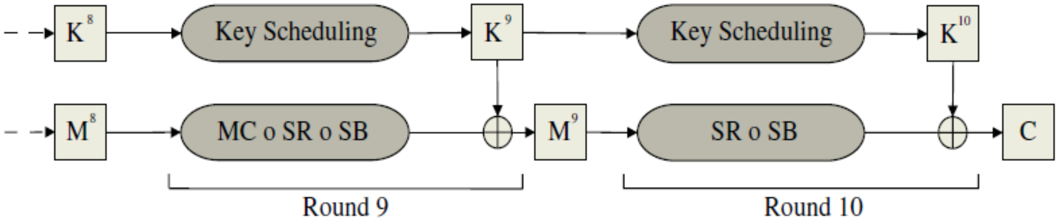

Figure 3 The last rounds of an AES-128

The Correct ciphertext (*C*) can be calculated from the 9th temporary cipher result (M^9) and the 10th round key (K^{10}) by

$$C = ShiftRows(SubBytes(M^9)) \oplus K^{10} \text{……(1)}$$

For the calculation of one byte, if the *i*th byte becomes the *j*th byte after ShiftRows, then the *j*th byte of the correct cipher is

$$C_j = SubByte(M_i^9) \oplus K_j^{10} \text{……(2)}$$

If a fault e_i is injected on the *i*th byte of M^9 just before the final round, a faulty ciphertext on the *j*th byte (D_j) will be

$$D_j = SubByte(M_i^9 \oplus e_i) \oplus K_j^{10} \text{……(3)}$$

From the above equations, we have

$$C_j \oplus D_j = SubByte(M_i^9) \oplus SubByte(M_i^9 \oplus e_i) \text{……(4)}$$

C_j is easy to get when there is now fault. D_j can be obtained when a one-bit fault is injected. Therefore, M_i^9 can be calculated. From M_i^9, K_j^{10} can be deduced. Then the key *K* can be known.

4. Clock glitching attack on AES

The bit fault mentioned in the above part can be injected by clock glitching attack described below.

Most ICs execute calculations by processing data by combinatorial logic blocks separated by D flip-flop register banks sharing the same clock. Data is usually latched by registers at rising clock edges. Between two such edges, the computed data travels between registers and gets modified by the intermediate combinatorial logic blocks. The time needed to propagate data through combinatorial logic is called propagation delay. The propagation delay and a second delay element, inherent to the use of D flip-flop, called set-up time, define the circuit's maximal operating frequency (nominal circuit period). Indeed, to ensure proper circuit operation, the clock period must be strictly greater than the maximal propagation delay in the concerned circuit (this maximal propagation delay is called critical path) plus the registers' set-up time. In other words:

$$T_{clock} > t_{critical} + t_{set-up} \text{……(5)}$$

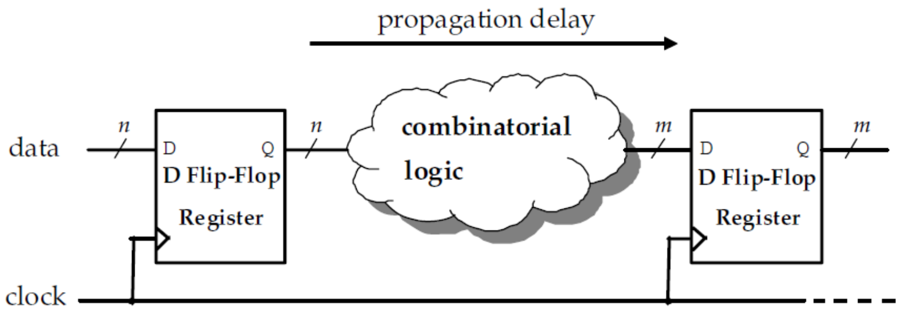

Figure 4 Synchronous Representation of Digital ICs

As a matter of fact, any data bit entering a register is the result of a combinatorial calculation involving several previous register output bits. The transformation of the previous registers' output into the next register's input bit takes a determined delay. This delay depends on the logic performed as well as on the data transiting through the logic. In addition, propagation time varies with circuit temperature and power supply voltage.

An adversary can inject clock fault by controlling two parameters: the precise moment at which the fault occurs and the clock anomaly's duration. Figure 5 gives an example of how a faulty clock can be generated. Two delayed versions of CLK with programmable skews are generated from CLK. FAULTY_CLK can be assembled by combining CLK_DELAYED_2's rising edge and CLK_DELAYED_1's falling edge. This is controlled by the signal TRIGGER that positions the perturbation in time. We can see that when the TRIGGER signal is triggered, the FAULTY_CLK will have a narrowed cycle whose period (faulty period) is shorter than the normal period.

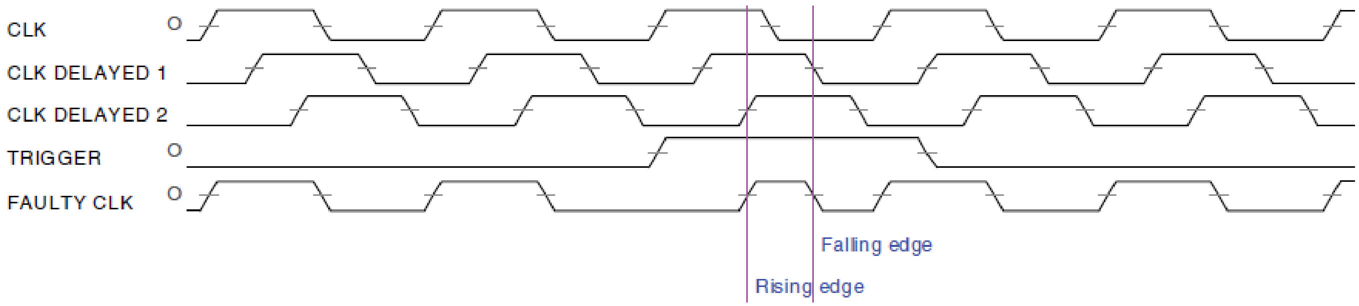

Figure 5 Faulty Clock Signal Generation

Clock an AES with this FUALTY_CLK. When not triggered, the AES will work normally. When triggered, but with a faulty period that is not so small, it is still possible for the AES to work well. As we control the faulty period to be shorter and shorter, more and more faults will be injected and the encryption result will be more and more different from the correct one. It is reported and has been practically proved that when the faulty period is controlled properly (neither too narrow nor too wide), an only one-bit fault on a byte of a temporary cipher result can be injected.

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. Digital channels are not recommended as you cannot see any slope on those channels, they are either 1 or 0. Therefore, important information may be lost. The original clock frequency of the clock source on the HAHA Board is 50MHz. The smallest time base the Analog Discovery 2 can reach is 10ns/div. The smallest three bases are recommended when you are configuring in the WaveForms software.

Instructions and Questions

In this experiment, you will implement an AES and attack AES to get the key by injecting a fault into it.

PART I Implement an AES

In this part, you are going to implement an AES into the FPGA. Verilog source code can be downloaded from the Canvas. You need to create a top module to instantiate the module aes_128 and a RAM. You will feed ‘clk’ (50MHz) from the board to the AES through an IO pin (pin 88). Provide ‘state’ and ‘key’ to the aes_128 in the top module by assigning values to wires. The encryption results should be stored into a RAM that you instantiated whose word width should be declared as 128 bits. (This procedure is very similar to what we have done on DES in the previous lab.) You can examine if your AES works fine by trying the keys and states below and see if the outputs are right.

state	Key	result
00000000000000000000000000000000	00000000000000000000000000000000	66e94bd4ef8a2c3b884cfa59ca342b2e
00112233445566778899aabbccddeeff	000102030405060708090a0b0c0d0e0f	69c4e0d86a7b0430d8cdb78070b4c55a
3243f6a8885a308d313198a2e0370734	2b7e151628aed2a6abf7158809cf4f3c	3925841d02dc09fbdc118597196a0b32

1. Each group uses its own state and key in the top module according to the table ***below***. Turn in your code for the top module.
2. Store the ciphertext in the In-System Memory Content Editor and take a screenshot of it. Turn in the picture. The encryption result should be correct according to your group’s state and key.

	state	Key
Group 1	d9da7bea1a31d8abe2a27b4e855c5c5c	50ed00c48388ea9b0fb7c204c2c12d39
Group 2	97157a6fc8e4bbe432c40d35f2716092	eba02e379817d636a144551df49ade37
Group 3	f01f2e724ac0ab35be3a20ff7a7d7fca	d005a3321bbf085c2bc611ae8820839d
Group 4	27ecb2e3a5ee3894885b5289307400e3	98546b83039e89eed41dc9e5f9ac1751
Group 5	2af70d6bf067d37ed41dcca704c38b4f	5dba9b93c355a576600a3e1ede5c396f
Group 6	f377a75c7d3b1679d10cf404fe12f3f1	de99eac143c1cfd0a59cfa5880592f5c
Group 7	7b4bf239cea318c9768d6a65cfa3f8f6	84116d9de060505c66d371312cf47ae7
Group 8	f49042f2e8f5f21deba6c156075be33b	1b3ceeda90cbdf65e7eda08bb1721e31
Group 9	8afa2f771dce63d87a1774897558d887	d1967592fae7810435d368979bf2a590
Group 10	451be11315accb73debf0193646502f3	97864a796b6f2e55cda680285fe680e7
Group 11	fe458cf6c49a4e25f8c80985fc5f23b5	94f7b931e1fa6604cb9aea5c2de4f7ba
Group 12	962f329d772795332149386ae179aa26	68064a6dc2289b33b8dccb1d9c45efc1

PART II DFA on AES

In this part, you will implement a DFA on AES to find the key of the AES. Download a sof file from the Canvas. There are twelve sof files. Each group should download and use the one named with your group number. The keys stored in the sof files are ***different*** from one file to another. You should report the correct key after analyzing differential fault.

The sof file you downloaded for this part is actually very similar to the one you created in PART I, except that there is another function which is to inject a bit fault into the AES.

For the new HAHA (HAHA SEP 1, used by students in the lab), turning on S9 will inject a bit fault. If it is OFF, ***no*** fault is injected, and you can observe the correct ciphertext in the In-System Memory Content Editor. However, when the switch is ON, there is a fault injected into one bit of temporary ciphertext of the 9th round (M^9). M^9 contains 128 bits and switches 0 to 6 can in total generate 128 (2^7) combinations. Each state of the seven switches is corresponding to one certain bit. S0 to S6 determines the location of the bit fault. For example, if S9 is ON and S6 to S0 shows 0001111, that means there is a fault injected on the 15th bit of M^9.

One bit error in M^9 will become a one-byte error in the cipher result. You can analyze the difference between the jth byte of the C and the D in equation (4). To find M^9, you can do this. In equation (6), try all the possibilities of x (which has 256 combinations) and e_i (which has 8 possibilities), and find when the right-hand side equals to the left. There will be several x's that satisfies the condition. Try all the 8 different D_j's for one byte and limit the number of x to be one. That x will be the byte you are looking for in M^9. After you get M^9, it will be easy for you to get K^{10} and K. You will need to use Matlab to make the guessing procedure faster. You need to record the C and all the 128 D's from the board and analysis the data.

$$C_j \oplus D_j = SubByte(x) \oplus SubByte(x \oplus e_i) \text{……(6)}$$

1. What is your group's C?
2. Why do we choose to inject a fault before the last encryption round? What's the difference between the last round and other rounds?
3. AES will do ShiftRows in every round. Which byte will the result of the encryption of the ith byte go to? (i=0,1, 2, …, 15)
4. For the jth byte of the encryption result, if you have C_j and all the eight D_j's, it is enough for you to calculate M_i^9.Turn in your Matlab script to calculate M_i^9. (If you can get one byte of M^9, you can get all bytes of M^9.)
5. What is your group's M^9?
6. What is your group's K^{10}?
7. What is your group's K?

PART III Clock glitching attack on AES (Optional)

The bit fault we injected in PART II can be realized in many ways, such as flash glitching and clock glitching. In this part, we will implement a clock glitching attack to produce a one-bit fault. All the groups will use the same key and same state shown in the table below.

State	00112233445566778899aabbccddeeff
Key	000102030405060708090a0b0c0d0e0f
Out	69c4e0d86a7b0430d8cdb78070b4c55a
M^8	fde3bad205e5d0d73547964ef1fe37f1
K^9	549932d1f08557681093ed9cbe2c974e
M^9	bd6e7c3df2b5779e0b61216e8b10b689
K^{10}	13111d7fe3944a17f307a78b4d2b30c5

Don't implement the whole AES. Implement only the 9th round and the last round. The inputs for the 9th round are M^8 and K^9 which are shown in the table above. The round key for the last round should be K^{10} as given in the table above. The text input for the last round is the encryption output of the 9th round which should equal to M^9 in the table above. However, when a fault is injected, M^9 will no longer stay the same. It is supposed to have bits flipped. Store M^9 and C (or D) in two RAMs.

Use the method mentioned in the background section to generate a clock glitch. You will be using the Phase-locked loops (PLL) provided in Quartus and you can instantiate it from the Installed IP by choosing 'ALTPLL'. When you configure the PLL, choose the speed grade to be 8 and the frequency input to be 50 MHz. Use two outputs and configure them to have different phase shifts (multiplication and division factors remain 1).

Generate CLK_DELAYED_1 and CLK_DELAYED_2 (which are just the outputs of the PLL) and generate FAULT_CLK as shown in Figure 5. The FAULT_CLK will be similar to the blue one in Figure 6.

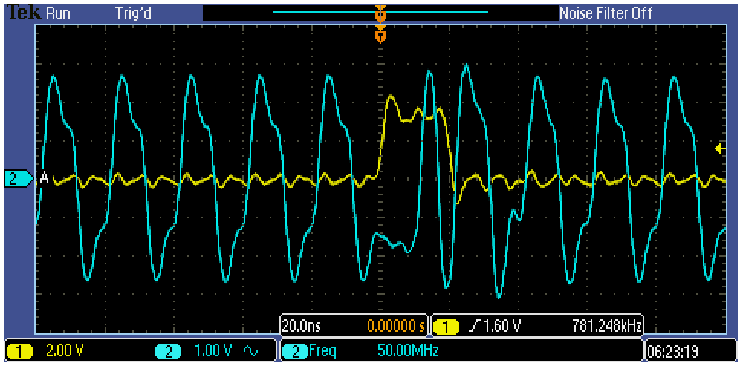

Figure 6 Sample waveform of the faulty clock.

Feed the FAULT_CLK into the last two rounds you just implemented and you may (or may not) find that the M^9 become different. If there is no fault or there are too many bit faults, adjust the phase shifts of CLK_DELAYED_1 and CLK_DELAYED_2. You should at least get one one-bit fault on one of the bytes of M^9. Ideally, you can get several one-bit faults on several bytes at the same time.

1. Turn in your Verilog code for the last two rounds of AES. Turn in your Verilog code for the FAULTY_CLK generator.
2. Turn in the oscilloscope waveform of the FAULTY_CLK when the one-bit fault happens.
3. Turn in a screenshot of the In-System Memory Content Editor showing the one-bit faulty M^9 and D.

☞ Lab Report Guidelines

Deliverables:

1. In your report, answer ALL the questions.
2. Prove your AES works well by giving your code and screenshot.
3. In part II, submit all the 128 wrong results of the encryption and 1 correct result. They will be used as the input to your Matlab (or C or others) code. Include your code; it should be able to (compiled and) run and get the key.
4. For part III, there will be a procedure to get the final result: you will have to try different parameters. Show your procedure by including screenshots for different parameters. Explain how do you approach the right answer.

Demonstration:

1. No demo and video is required.

References and Further Reading

[1] Barenghi, Alessandro, et al. "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures." Proceedings of the IEEE 100.11 (2012): 3056-3076.

[2] Giraud, Christophe. "DFA on AES." International Conference on Advanced Encryption Standard. Springer Berlin Heidelberg, 2004.

[3] Skorobogatov, Sergei P., and Ross J. Anderson. "Optical fault induction attacks." International Workshop on Cryptographic Hardware and Embedded Systems. Springer Berlin Heidelberg, 2002.

[4] Barenghi, Alessandro, et al. "Fault attack on AES with single-bit induced faults." *Information Assurance and Security (IAS), 2010 Sixth International Conference on*. IEEE, 2010.

[5] Agoyan, Michel, et al. "When clocks fail: On critical paths and clock faults." *International Conference on Smart Card Research and Advanced Applications*. Springer Berlin Heidelberg, 2010.