Assignment 2: Side-Channel Analysis and Fault Injection

Amar Lakshya (AXX945), Calvin Menezes (CXM1010)

School of Computer Science, University of Birmingham

---

1. Task1

The average number of cycles per execution of the exponentiation for $e = 17$ is: 9224544 cycles. **TODO**

2. Task2

2.1. Oscilloscope Capture

The image of the RSA operations can be seen below:

Figure 1: The picture of oscilloscope showing the first 6 operations of RSA

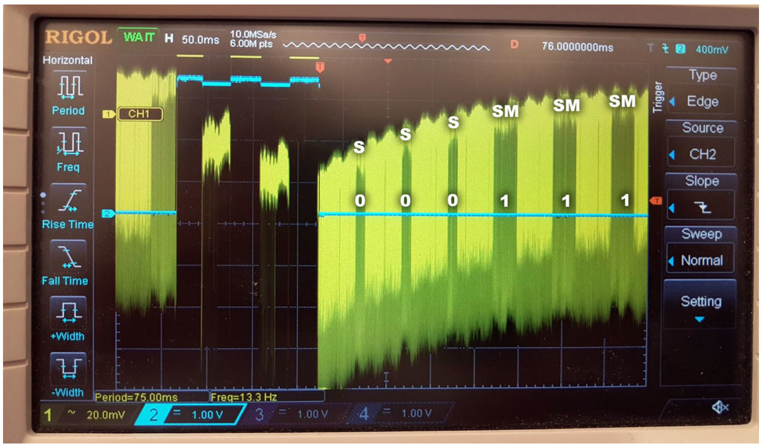

Figure 2: Superimposed picture showing operations as seen in the trace

- Based on the square-and-multiply (SaM) algorithm, the for-loop contains a square (S) operation that takes place in every iteration and a multiply (M) operation that is done only when the current exponent bit is 1.
- The initial bit of the exponent is always 1 and the loop starts from the 2nd bit onwards (Left to right).
- The screenshot above from an oscilloscope shows variation in power consumption as the algorithm runs.
- The thinner lines plotted indicate a square operation and the thicker lines indicate squaring followed with multiplication.
- The sequence of operations from the graph are S, S, S, SM, SM, SM that translates to the bits being 0,0,0, 1, 1, 1.
- Therefore, the upper six bits of the exponent e are 100011.

2.3. *Acquired Trace*

The graph of the acquired trace can be found in Fig 3 and **python** program file *load_trace* contains the program.

2.4. *SPA*

TODO

3. Task3

3.1. Overlapping Traces

The plot of the first 10 traces can be seen in Fig 4.

The corresponding output can be seen below:

```
byte value for trace 0: D8
byte value for trace 1: E8
byte value for trace 2: 42
byte value for trace 3: 3D
byte value for trace 4: 96
byte value for trace 5: E5
byte value for trace 6: CB
byte value for trace 7: 0B
byte value for trace 8: 1F
byte value for trace 9: 33
```

3.2. AES XOR

The following function implements the required functionality (the complete program can be found in the aforementioned source code repository and submitted zip):

```python
def AES_XOR(x, k_prime):
    return SBOX[x ^ k_prime]
```

3.3. Significant Bit

The method to extract the *MSB* involves two steps where we first convert an *integer* to a *bit-field* and then we get the first element of the *bit-field*. The following is an implementation of the function in **python**:

```python
def int_to_bits(n):
    return [n >> i & 1 for i in range(BIT_SIZE-1,-1,-1)]

def MSB(n):
    return int_to_bits(n)[0]
```

3.4. DPA

The submitted program file *dpa.py* contains the program to produce the overlapping graphs of all difference-of-means curves as seen in Fig 5 and the zoomed-in part showing the candidate key $K10 = 7$ as seen in Fig 6.

4. Task4

4.1. Fault Injection on RSA-CRT

From the given values $n = 91, e = 5, S = 48, \overline{S} = 35, x = 55$ we calculate the valid signature y by:

$$y = x^e \mod n \tag{1}$$

$$y = 55^5 \mod 91 \tag{2}$$

$$y = 48 \tag{3}$$

4.1.1. Using Bellcore Method

The bellcore method gives factor q by:

$$q = \gcd(S - \overline{S}, n) \tag{4}$$
$$q = \gcd(48 - 35, 91) \tag{5}$$
$$q = \gcd(13, 91) \tag{6}$$
$$q = 13 \tag{7}$$

we know $n = p * q$ therefore factor p is:

$$p = n/q \tag{8}$$
$$p = 91/13 \tag{9}$$
$$p = 7 \tag{10}$$

4.1.2. Using Lenstra Method

The Lenstra method gives factor q by:

$$q = \gcd(\overline{y}^e - x, n) \tag{11}$$
$$\tag{12}$$

From eq 1, we know that $y = 48$ and since $\overline{y} = y, \overline{y} = 48$, therefore

$$q = \gcd(48^5 - 55, 91) \tag{13}$$
$$q = \gcd(48 - 55, 91) \tag{14}$$
$$q = \gcd(7, 91) \tag{15}$$
$$q = 7 \tag{16}$$

we know $n = p * q$ therefore factor p is:

$$p = n/q \tag{17}$$
$$p = 91/7 \tag{18}$$
$$p = 13 \tag{19}$$

The major advantage of Lenstra method is that only the faulty value $\overline{S}$ needs to be known.

4.2. Counter Measures

4.2.1. Difference between Detection based and Algorithmic

- Detection-based countermeasures are based in detecting faults through hardware or software properties of faults while Algorithmic take help of specific algorithmic patterns and monitor changes in those patterns caused by faults.
- Since Algorithmic countermeasures can use techniques like time randomization, they are much harder to inject faults against.

4.2.2. RSA-CRT Countermeasures

- Detection-based countermeasures
  - **TODO**
- Algorithmic countermeasures
  - A simple countermeasure could be to do verification of the signature before and after RSA-CRT using Hashes.
  - **TODO**

4.2.3. SPA Prevention

There a number of countermeasures that can be taken to prevent the SPA attack:

- The Square and Always Multiply Algorithm can be used to conditionally throw away the result so that there's less contrast in the acquired power trace.
- Noise Addition or Signal Reduction can be implemented on the signal to reduce the SNR
- Balancing power consumption requires a lot of changes in hardware that create redundant circuits and is therefore not a preferred countermeasure.

4.2.4. Time Randomization

- In Both Fault Injections models namely *Flip-Bit* and *Stuck-at*, the faults are injected in order to alter the normal control flow of a program and since the instructions in a control flow are dependent on time, Time Randomization makes these injections to work.
- In Side-Channel attacks focus on recovering secrets through analysis of the Runtime. Since the Runtime of a program is based on time, Time randomization makes these attacks also harder to work.

4.2.5. DPA Effects

- Increasing the amplitude noise decreases the SNR and therefore each power trace will have lesser amount of signal and a greater number of traces will be required.
- If the attacker can acquire infinitely many traces then increasing noise below a certain point (after which the signal is no longer traceable) will not be a suitable countermeasure. However under a finite number of traces, this technique can function as an effective countermeasure.

5. Appendices

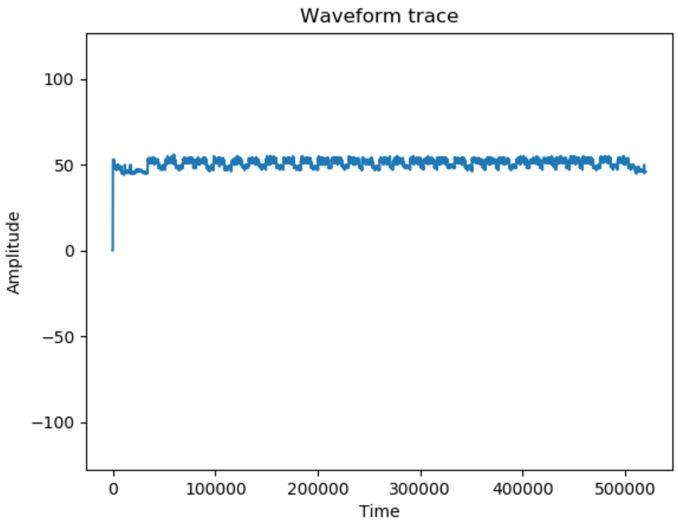

Figure 3: Graph of the acquire trace in full scale

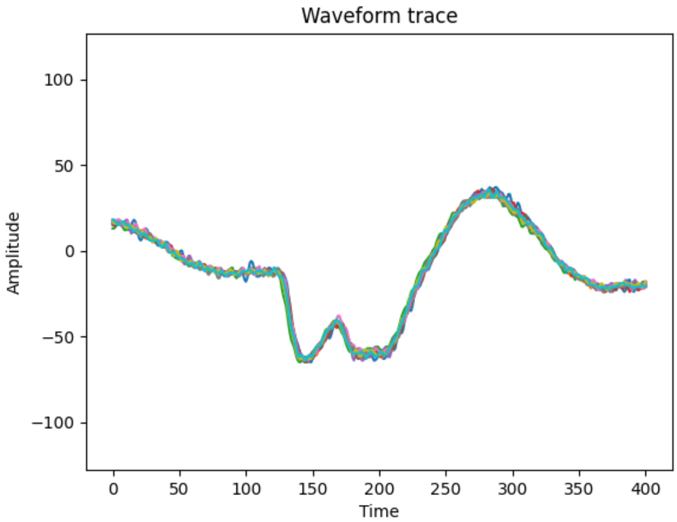

Figure 4: Graph of superimposed 10 traces

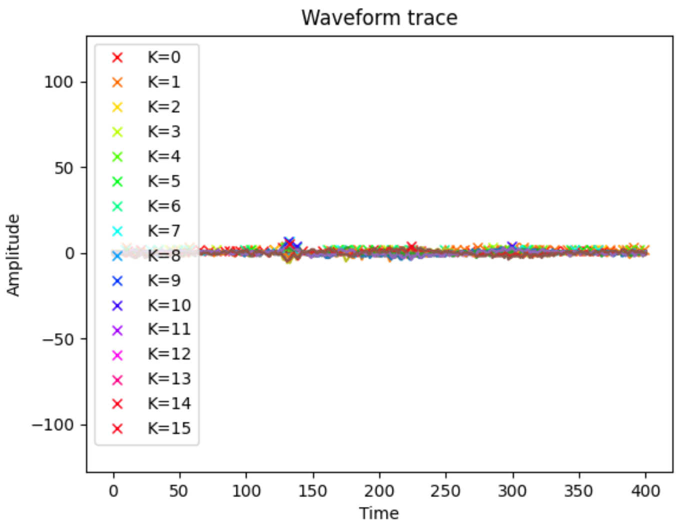

Figure 5: Graph of superimposed difference-of-means

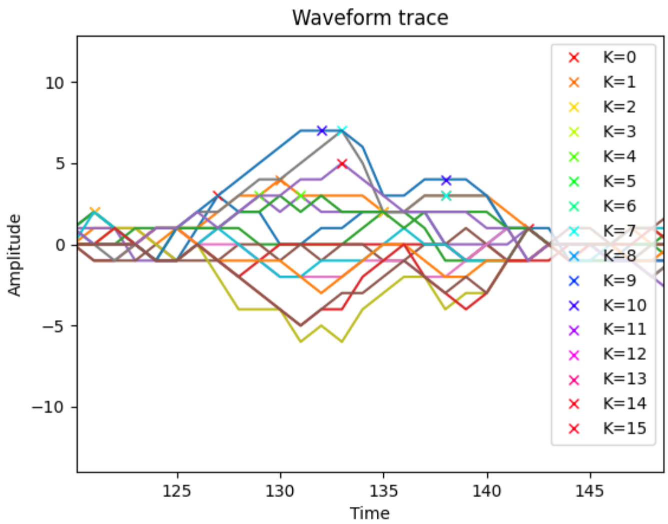

Figure 6: Graph of zoomed-in part of difference-of-means