ECE/CS230 – Computer Systems Security @KAUST
Fall 2023

TA: Rana Alahmadi

Assignment 4

Submit your solutions as a .zip file at blackboard.kaust.edu.sa.
Include your **full name** in the submitted files.
Name the solutions folder as **x**-A4.zip, where **x** is your **last name**.

Preliminaries: Read about side channel attacks and the AES encryption algorithm. You can start with the following resources:

- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- https://en.wikipedia.org/wiki/Side-channel_attack
- Daniel J. Bernstein, "Cache-timing attacks on AES", 2005
- https://en.wikipedia.org/wiki/Finite_field_arithmetic
- https://privacycanada.net/aes-encryption-guide/

Lab Objectives: The goal of this lab is for the students to learn and understand symmetric data encryption with the AES algorithm, and how side-channel attacks can extract data from otherwise secure encryption mechanisms.

Lab deliverables: The student must submit a zip file containing a text file (in .pdf format) with the answers to problem 1 and problem 2.1- 2.4, as well as images (.png or .pdf) with the answers to problem 2.5.

Problem 1 (60 points): Below is provided an *incomplete* version of the AES algorithm used in Bernstein's cache-based side channel attack (**three** steps have been removed, 20 points each). The student must:
1) Identify where the instructions have been removed (write the step number, e.g., 8.5 if an instruction is missing between 8 and 9) (5 points)
2) Provide the correct instructions to fill the missing steps in order to correctly implement AES (10 points)
3) For each removed step, explain why it is necessary, and how the output would be affected if that step remains skipped (5 points)

The provided algorithm is the following:
1) Take 128-bit input n for AES scrambling
2) Generate a key, say 128-bit, k
3) Define a constant 2048-bit (256-bytes) table S
4) Define a constant 2048-bit table S'

5) S and S′ are expanded into four 8192 bits (1024 bytes) tables: T0, T1, T2, T3. These tables are defined by the figure below:

$$T_0[b] = (S'[b], S[b], S[b], S[b] \oplus S'[b]),$$
$$T_1[b] = (S[b] \oplus S'[b], S'[b], S[b], S[b]),$$
$$T_2[b] = (S[b], S[b] \oplus S'[b], S'[b], S[b]),$$
$$T_3[b] = (S[b], S[b], S[b] \oplus S'[b], S'[b]).$$

6) AES uses two 128-bit (16-byte) auxiliary arrays, x and y

7) Array x is initialized to k

8) Array x is chunked into 4 4-byte arrays: x0, x1, x2, x3

9) Compute Array e as = (S[x3[1]]⊕1, S[x3[2]], S[x3[3]], S[x3[0]])

10) Array y = (y0, y1, y2, y3) is then modified as follows:

$$(T_0[y_0[0]] \oplus T_1[y_1[1]] \oplus T_2[y_2[2]] \oplus T_3[y_3[3]] \oplus x_0,$$
$$T_0[y_1[0]] \oplus T_1[y_2[1]] \oplus T_2[y_3[2]] \oplus T_3[y_0[3]] \oplus x_1,$$
$$T_0[y_2[0]] \oplus T_1[y_3[1]] \oplus T_2[y_0[2]] \oplus T_3[y_1[3]] \oplus x_2,$$
$$T_0[y_3[0]] \oplus T_1[y_0[1]] \oplus T_2[y_1[2]] \oplus T_3[y_2[3]] \oplus x_3).$$

11) AES again modifies Array x with ⊕2 this time

12) On the 10th round, because we have a 128-bit key, Array y is modified with (S[], S[], S[], S[]) because there is no mixing.

Problem 2 (40 points): We provide the files server.c, study.c and correlate.c which show an example of how to create a server running an encryption service, how to run a time-based side-channel attack against it, and how to analyze the results of the attacks.

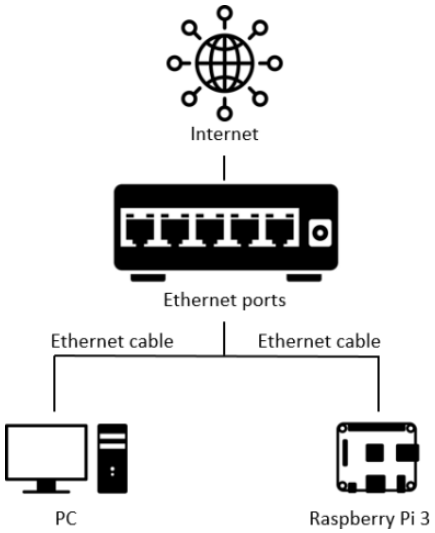
We also provide the files study.400 and attack.400 reporting the results of running the attack on a dedicated machine, as students are not required to run the attack themselves. The student must:

1) Read and understand the attack explained below (no grade)
2) Extract the results using correlate.c and paste, in your answers, the *last three lines* (5 points)
3) Find for which bytes the range of possibilities for each k[i] is lower than 256 (5 points)
4) Explain, in their own words, how a particular k[i] was selected as having fewer than 256 possible values during the analysis. This must be a high-level explanation reflecting your understanding of the attack and how to interpret the correlated data (just copying the relevant line of code will yield no point) (10 points)
5) For one specific k[i] index which had fewer than 256 possibilities (if there are multiple options, pick one), draw 2 graphs:
    a. How n[i] affects OpenSSL AES timings for k = i inside the targeted server (10 points)
    b. How n[i] affects OpenSSL AES timings for random selection of k inside the targeted server (10 points)

The attack results were collected in the following environment:

Test 1 – Intel®i7-8700
Server (VM Ubuntu –Bridged adapter)
Intel® i7-8700, 16GB RAM
Ubuntu 18.04.2 LTS
L1d cache: 192K
L1C cache: 192K
L2 cache: 51536K
L3 cache 12,288K
gcc (Ubuntu 7.4.0-1ubuntu) 7.4.0
OpenSSL 1.1.1, 11 Jan 2018

Attacker (Raspberry Pi 3 Model B)
Broadcom BCM2837RIFBG, 1GB RAM
L1 cache: 32K
L2 cache: 512K
gcc (Raspbian 8.3.0-6+rpi1) 8.3.0
OpenSSL 1.0.1c, 28 May 2019



Preparation of the attack: In the attacker's role, the server.c program is compiled and run on the first VM: Ubuntu (server). The program allows the targeted server to listen for UDP packets on port 1000 of the specified server's IP address (146.201.69.200) using a known AES key (all zeros, as indicated by /dev/zero), as follows:

```
$ gcc version
gcc (Ubuntu 7.4.0-1 ubuntu1~18.04.1) 7.4.0 $ openssl version
OpenSSL 1.1.1 11 Sep 2018
$ gcc -O3 -o server server.c -lcrypto
$ ./server 146.201.69.200 < /dev/zero
```

From the raspberry pi 3 (attacker), then, random 400-byte packets are sent to the IP of the server using the study.c program, which is first compiled and then run with the server's IP address and the number of bytes for each packet as parameters, saving the output in a file called study.400, as follows:

```
$ gcc -O3 -o study study.c -lm
$ ./study 146.201.69.200 400 > study.400
```

A number of packets are sent to the server at this point until the program is stopped. The time of the transmission can vary from one processor to the other, as well as depending on the connection being used.

In the example experiment, 216 packets required 36 hours to complete by using Intel 7 processor 8700 When the program is stopped (Ctrl + C), the study.400 file contains many lines with information about the attack, such as the following:

```
13 400 8 262471 1224.266 42.036 -0.018 0.082
```

This line conveys the following information: The server was sent 2624711 400-byte packets having n[13] = 8. It handled those packets in 1224.266 cycles on average, with a deviation of 42.036 cycles. Compared to the average over all choices of n[13], the average for n[13] = 8 was 0.018 cycles lower. The number 0.082 is an estimate of the deviation in this difference: if the distribution of timings is close to normal, with a deviation around 42.036 cycles, then an average of 262471 such timings has a deviation around 42.036/262471 ≈ 0.082 cycles.

(The deviation in the average over all choices of n[13] is, presumably, 1/16th as large, and is ignored). Other lines show similar information for other choices of n[13].

Carrying out the attack: In the victim's role, the server (VM: Ubuntu in Test 1), 16 bytes are taken from /dev/random, the operating system's pseudorandom number generator, and stored in a variable named secretkey (a secret 16-byte key used as the AES key that the attacker will try to extract), as follows:

```
$ dd if=/dev/urandom of=secretkey bs=16 count=1 1+0 records in
1+0 records out
16 bytes copied, 1.35026 e-04s, 118 kB/s
```

The server.c program is compiled and run again in the server computer, this time using a secret key (the variable secretkey) instead of the known key (all zeros) used in the preparation, as follows:

```
$ gcc -O3 -o server server.c -lcrypto $ ./server 146.201.69.200 <
secretkey
```

Then, in the attacker's role, random 400-byte packets are sent to the victim's server, as follows as in the preparation, saving the output in a file called attack.400, as follows:

```
$ ./study 146.201.69.200 400 > attack.400
```

After a considerably amount of time (similarly to the preparation step, 36h) to send 216 packets, the program is stopped (Ctrl+C). The attack.400 file contains many lines with information about the attack, just like study.400 did in the preparation.

Analysis of the attack: After 216 random packets, we have to analyze the attack by correlating the lines obtained from the preparation (study.400) and from the attack (attack.400). We can produce 16 correlations, one per byte, which show if the secret key could be easily guessed: if the produced number of possibilities is lower than 256 for a given key byte, then the range of possibilities to guess that secret byte is narrowed.

For this test, the range of possibilities that was produced by correlating the preparation and the attack was narrower than 256 for some k[i], which indicates that the correlation was extracting valuable information, although limited, about the secret key; showing that the attacks to the Intel 7 processor 8700 were somewhat successful.

As mentioned above, the resulting timings from the attack are correlated with the known timings collected in the preparation, using the correlate.c, which correlates the two files study.400 and attack.400, saving the output in a file called attack, as follows:

```
$ gcc -O3 -o correlate correlate.c -lm

$ (tail -4096 study.400 ; tail -4096 attack.400) | ./correlate >>
attack
```

The first column of the resulting attack file shows the range of possibilities for each k[i] (second column). The rest columns show the corresponding values. For example, one line could be:

```
10 5 c6 cc c3 c8 ce c4 cd c9 c2 c0
```

We can see that the secret k[5] can be any of the presented 10 values (e.g 0xc6, 0xcc etc.) instead of all the possible 256 values.