When Algorithm Meets Reality

Homework Assignment 2 - Correlation Power Analysis

DESCRIPTION

A simulated AES implementation is waiting for you at the address: http://aoi-assignment2.oy.ne.ro:8080/
Your assignment is to discover the AES key using Correlation Power Analysis. You can get simulated side-channel traces for your attack from this URL, for the difficulty you choose:
http://aoi-assignment2.oy.ne.ro:8080/encrypt?user=yos&difficulty=1
You can verify your guess with using this address, which returns 1 if the password is correct:
http://aoi-assignment2.oy.ne.ro:8080/verify?user=yos&difficulty=1&key=4b165f8317b1a0a0a623a09df78ceaaa

You should try to crack using no more than 10,000 traces with the highest difficulty possible.

Running the Assignment:

To offload the server and make sure the running time is not affected by other users, you are provided with a docker image to run the assignment locally. when you are finished you can test yourselves on the course official server.
To run the docker run the following commands in your terminal:

```
docker pull annakul1/attacks_on_implementations:Assignment2
docker run -p 80:8080 annakul1/attacks_on_implementations:Assignment2
```

SUBMISSION GUIDELINES

1. Submission is through the Moodle site.

   NOTE: Late submissions are always allowed, with the price of 1 point deducted per day. Please notify the grader of any late submission.

2. Submission is allowed in groups of two students.
3. For the first milestone, you do not need to submit a pdf report, for the second milestone you do.
4. Submission is only in **Python** (version3+).
5. For questions, please use the Piazza hw3.
6. For problems regarding submission, contact the grader Anna Kulakovsky at annakul@post.bgu.ac.il

HINTS AND TIPS

- The key and plaintext are 16 bytes in hex format (32 characters, no spaces or delimiters).
- The web server returns outputs in JSON format. You probably do not have to write a parser, since there are JSON parsers for almost all programming languages.
- The JSON output will include a random plaintext and leaks.
- There are 32 interesting positions in the power trace: 16 contain $HW(P^{\mathrm{k}})$, and 16 contain $HW(S[P^K])$. Just like real power traces the simulated power traces include other “garbage” values which are not relevant to the attack.
- Please specify your username and difficulty in your requests. If the difficulty is not specified – the default will be 1
- The attack works under the known plaintext model, you will get the plaintext together with the side channel.
- You do not need the web server for the attack! Just download 10,000 traces and save them to disk.
- Do not attempt to “hack” the web server or any other service running on the host, don’t try file traversal attack – you will probably succeed. This will be considered an ethics violation and will result in disciplinary action.

SUBMISSION MILESTONES

Milestone 1:

1. Submission date:**03/06/2025**
2. Input: a filename to save the traces to.
3. Output: mean and variance of each one of the traces.
4. Functionality: Downloads the traces and saves them in filename. Then, find the mean and variance of each one of the traces it downloaded, and prints the following:

```
print(f'Mean\tVariance') #Print once at the start of your program and after that:
print(f'{mean}\t{var}) #For each one of the traces
```

5. Implementation Suggestion (optional, you can implement however you want, this is just a suggestion): begin with implementing the following functions:
   - The method `download_power_traces (filename, serverURL, number_of_power_traces)`, which will download `number_of_power_traces` power traces from the server `serverURL` and save the results to the file filename in a usable format (.txt), in which each row represents a downloaded power trace.
   - The method `getMeansVariances(filename).` The method should read the file `filename,` containing the saved power traces, and calculates the mean and variance of each position in the trace (i.e., if you have 10,000 traces, each with 200 measurements, you should output 200 means and 200 variances)

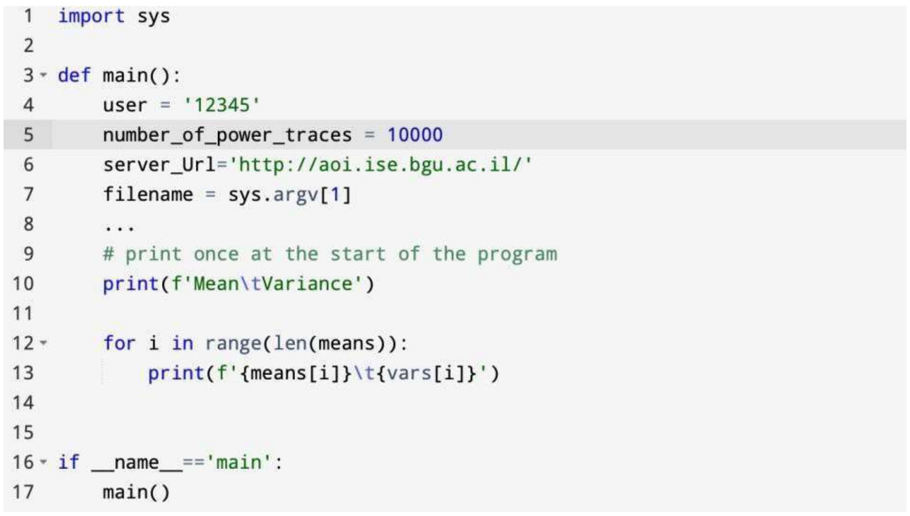
6. <u>The submission should include:</u>

   a. A link to Google Colab with the name "ex02_M1_ID1_ID2.zip" (where ID1, ID2 are replaced with the students' ID's) with the complete source code, in high quality (with comments, indentation, good function and variable names, return code checking etc.)

   b. The output file with the downloaded traces (i.e., the output of `download_power_traces` method).

   c. Output of a sample execution of the `main` program.

   d. Following is a template for you. Make sure to define the class ex02_M1 as your primary class with the main that receives a single string, and at the end prints the time as follows.



```
import sys

def main():
    user = '12345'
    number_of_power_traces = 10000
    server_Url='http://aoi.ise.bgu.ac.il/'
    filename = sys.argv[1]
    ...
    # print once at the start of the program
    print(f'Mean\tVariance')

    for i in range(len(means)):
        print(f'{means[i]}\t{vars[i]}')


if __name__=='main':
    main()
```

   e. The graders test program will read the traces file you saved and calculate the mean and variance and compare to your results.

Milestone 2:

1. Submission date: **24/06/2025**
2. Input: a filename to save the traces to.
3. Output: username, key and difficulty as shown in “Functionality”.
4. Functionality: Downloads the traces and saves the in filename. Finds the correct key, in hexadecimal, without ‘0x’ (for example: 4b165f8317b1a0a0a623a09df78ceaaa), and prints the user and the key using the following:

   ```
   print(f‘{user},{key},{difficulty}’)
   ```

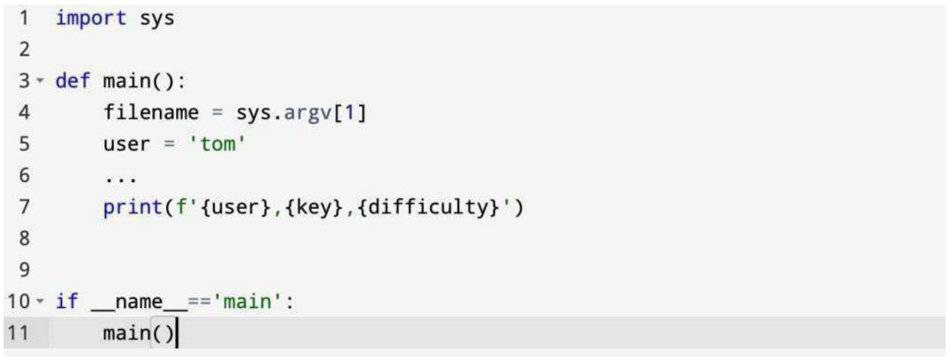
   You must print all passwords and difficulties from 1 up to at least difficulty 10 within 10 minutes.

   Additionally, print the time it took to find each key.

   Example output:

   ```
   alice,4b165f8317b1a0a0a623a09df78ceaaa,1
   alice,83a1e7a9c2f05d9b92db5a894ded0ef3,2
   ...
   alice,dcb0e7e924f87dfaf21e4030b7d3772a,10
   Total runtime: 100.43 seconds
   ```
5. Note:
   a. To print the key in hexadecimal you can use:

      ```
      print(format(ord(key), "x")) #For each char
      ```
   b. The program does not have to connect to the server to verify the key, you can do this manually.
6. The submission should include:
   a. A link to Google Colab with the name "ex02_M2_ID1_ID2.zip" (where ID1, ID2 are replaced with the students’ ID’s) with the complete source code in high quality (with comments, indentation, good function and variable names, return code checking etc.)
   b. The output file with the downloaded traces (i.e., the output of `download_power_traces` method from ex02_M1).
   c. Following is a template for you. Make sure to define the class ex02_M2 as your primary class with the main that receives a single string, and at the end prints the time as follows:



```python
import sys

def main():
    filename = sys.argv[1]
    user = 'tom'
    ...
    print(f'{user},{key},{difficulty}')


if __name__=='main':
    main()
```

d. A PDF report with the name: "ex02_M2_ID1_ID2.pdf", inside the submitted zip. Should include:

1. The highest difficulty you achieved.
2. Output of a sample execution of the program, showing how it recovers the keys for each member of the team.
3. Analysis:
   i. What positions in the power trace were the correct ones to use? How did you find them?
   ii. How many traces did you need for the attack? To answer this question, please provide a graph with the number of traces on the x scale and the count of correct bytes in the output on the y scale (minimum 0, maximum 16).
4. NOTE: you can submit the report only in English.

Good luck!

APPENDIX:

Here is the AES SubBytes table, for your convenience:

```
AesSbox = [
        0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
        0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
        0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
        0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
        0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
        0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
        0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
        0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
        0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
        0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
        0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
        0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
        0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
        0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
        0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
        0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
]
```