CT437 Assignment 2

Using and Benchmarking Blockciphers using OpenSSL

CANNY, LUKE (19339166)

Introduction

In this assignment, OpenSSL is used to encrypt and decrypt data using different blockciphers. The CPU time of each algorithm is measured and compared in this report. This report is broken down into two primary sections covering problem 1 and problem 2 as outlined in the assignment brief.

Problem 1: Blockcipher Benchmarking

In this section, the following encryption settings are first measured (CPU time) and then contrast:

- AES, ARIA and Camellia Algorithm
- 128- and 256-bit key length
- ECB, CBC and GCM mode
- 10 MB and 100 MB of data
- encoding and decoding

In the following plot, the average time taken to complete encryption or decryption is measured for different block sizes. From the graph is it evident that AES (advanced encryption standard) algorithm is significantly faster than ARIA or Camellia regardless of the size of the plaintext.

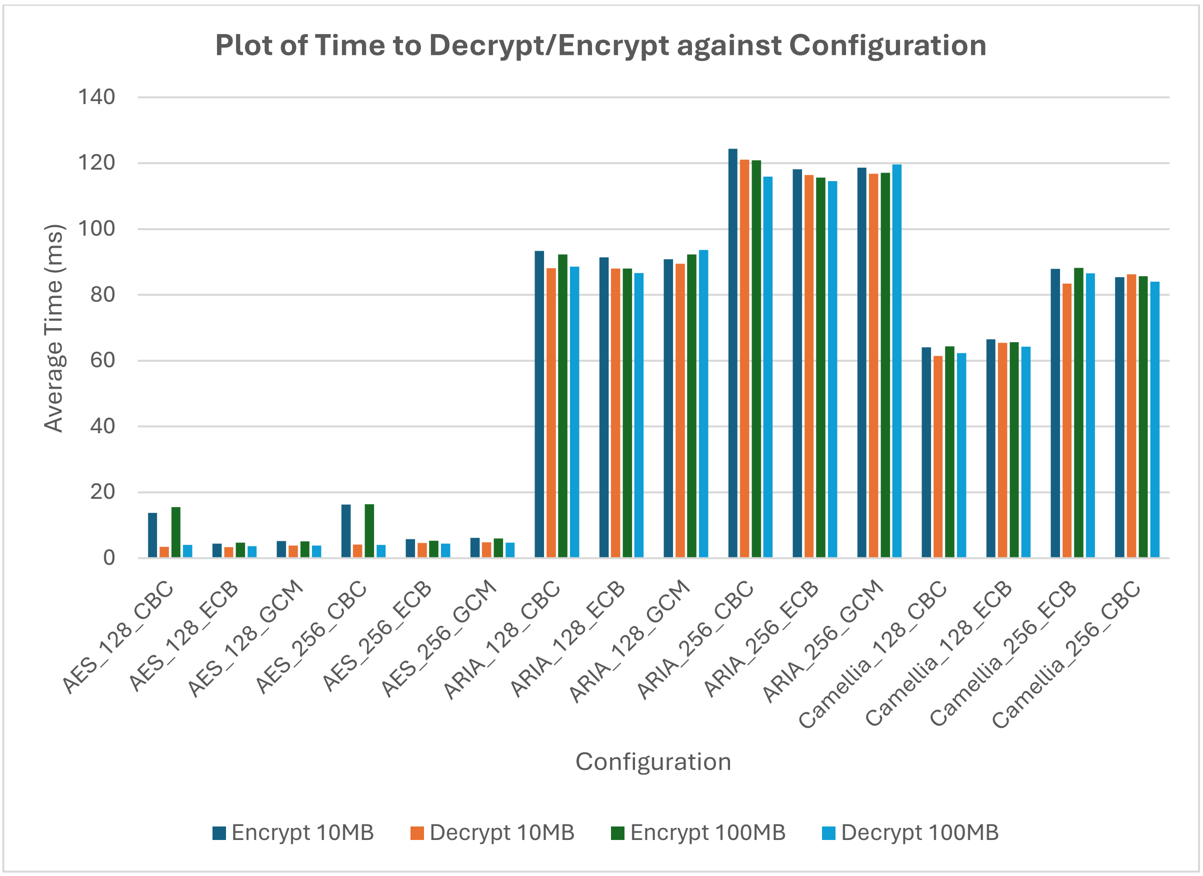


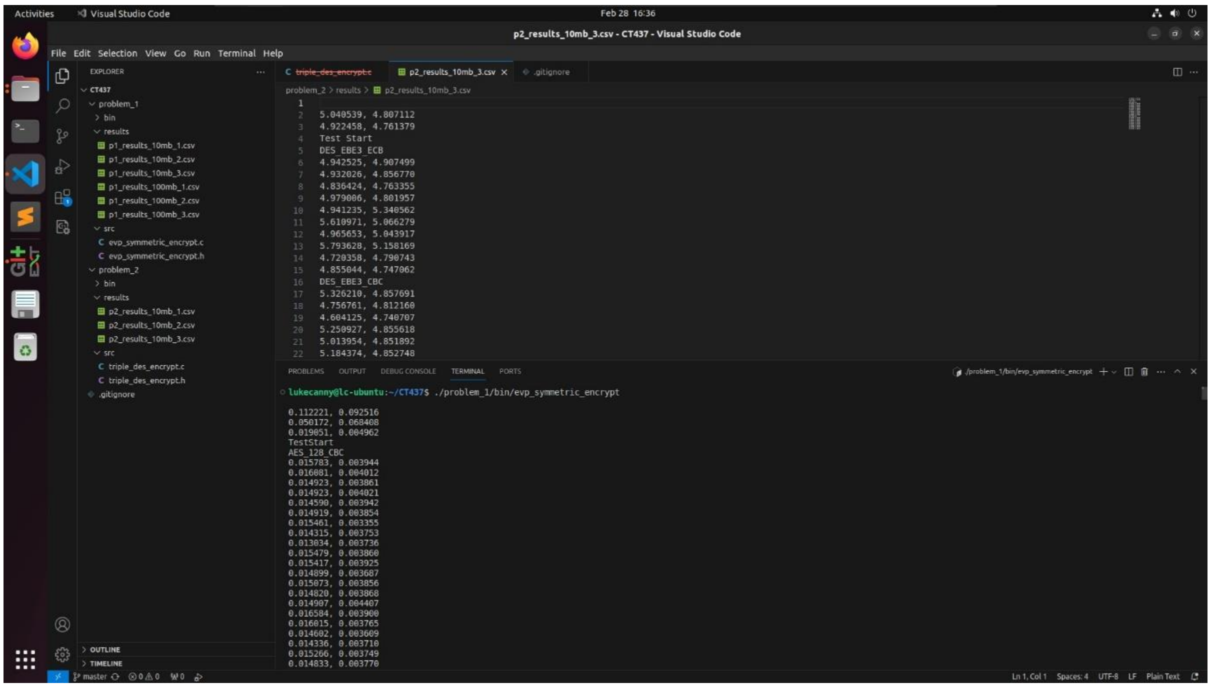
Figure 1 Plot of average time to complete operation against configuration used.

From the graph, it may also be observed that the size of the plaintext (10 or 100 megabytes) has little impact on the performance of each algorithm in most configurations with the exception of the AES algorithm, in CBC mode. In this configuration, the algorithm is significantly slower when encrypting the data. The decryption speed of AES in CBC mode is comparable to the other AES algorithm configurations.

The graph also presents another characteristic which impacts the performance of each algorithm, the size of the key. It is clear from the figure that configurations with a 128-bit key can encrypt and decrypt faster than their 256-bit counterparts. When looking at CPU time alone, this would make 128-bit keys seem more advantageous, however it must be noted that a smaller key reduces the level of protection each algorithm provides. On average, 128-bit key configurations were 20.9% lower than its 256-bit counterpart. In my opinion, the performance gain is not significant enough to warrant the loss in resilience of the ciphertext produced.

Running the Benchmarks

In figure 2, the script is running on a virtual machine running Ubuntu 22.04. The script is running in a terminal instance in VS Code. To gather the benchmarks into a file, the following bash command was used: *./problem_1/bin/evp_symmetric_encrypt > output.csv*


Feb 28 16:36
p2_results_10mb_3.csv - CT437 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
~~triple_des_encrypt.c~~
p2_results_10mb_3.csv
.gitignore
problem_2 > results > p2_results_10mb_3.csv
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

5.040539, 4.807112
4.922458, 4.761379
Test Start
DES_EBE3_ECB
4.942525, 4.907499
4.932026, 4.856770
4.836424, 4.763355
4.979006, 4.801957
4.941235, 5.340562
5.610971, 5.066279
4.965653, 5.043917
5.793628, 5.158169
4.720358, 4.790743
4.855044, 4.747062
DES_EBE3_CBC
5.326210, 4.857691
4.756761, 4.812160
4.604125, 4.740707
5.250927, 4.855618
5.013954, 4.851892
5.184374, 4.852748
```

```
lukecanny@lc-ubuntu:~/CT437$ ./problem_1/bin/evp_symmetric_encrypt

0.112221, 0.092516
0.050172, 0.068408
0.019051, 0.004962
TestStart
AES_128_CBC
0.015783, 0.003944
0.016081, 0.004012
0.014923, 0.003861
0.014923, 0.004021
0.014590, 0.003942
0.014919, 0.003854
0.015461, 0.003355
0.014315, 0.003753
0.013034, 0.003736
0.015479, 0.003860
0.015417, 0.003925
0.014899, 0.003687
0.015073, 0.003856
0.014820, 0.003868
0.014907, 0.004407
0.016584, 0.003900
0.016015, 0.003765
0.014602, 0.003609
0.014336, 0.003710
0.015266, 0.003749
0.014833, 0.003770
```

Figure 2 Script running on an Ubuntu Virtual Machine

Problem 2: Implementing and Benchmarking Triple-DES

Triple-DES was implemented using the OpenSSL library in C as seen in the source code. The Triple-DES implementation performed extremely poorly relative to the configurations tested in problem 1 of this report. Thus, in table 1, the performance of Triple DES in ECB and CBC mode is only compared with the worst case observed in the previous section.

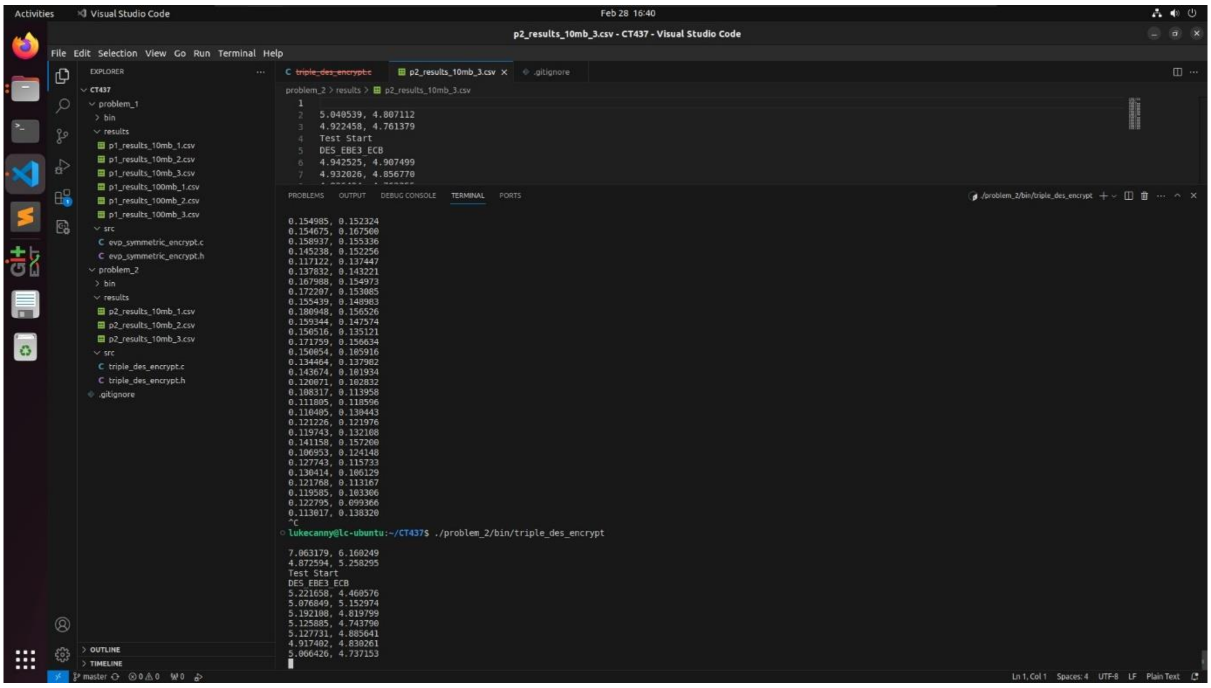
	Worst Case in P1 100MB ARIA 128bit CBC	Triple DES 100MB ECB	Triple DES 100MB CBC
Encryption	120.83 ms	5057.68 ms	4993.18 ms
Decryption	115.90 ms	4945.63 ms	4805.77 ms

Table 1 Worst Case Observed in Problem 1 vs Triple DES Results

The poor performance of Triple-DES can be attributed to many factors in its design. In short, Triple-DES is slower due to smaller block size, algorithm complexity and its lack of support for hardware acceleration when compared to AES, ARIA and Camellia. Specifically, 3DES is slower as it processes each block three times rather than only once.

Running the Benchmarks

Similarly to problem 1, the script is run on the same virtual machine (Ubuntu 22.04) in VS Code. To gather the benchmarks into a file, the following bash command was used: *./problem_2/bin/triple_des_encrypt > output.csv*


p2_results_10mb_3.csv - CT437 - Visual Studio Code
problem_2 > results > p2_results_10mb_3.csv
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS

```

5.040539, 4.807112
4.922458, 4.761379
Test Start
DES_EBE3_ECB
4.942525, 4.907499
4.932026, 4.856770
```

```
0.154985, 0.152324
0.154675, 0.167500
0.158937, 0.155336
0.145238, 0.152256
0.117122, 0.137447
0.137832, 0.143221
0.167988, 0.154973
0.172207, 0.153085
0.155439, 0.148983
0.180948, 0.156526
0.159344, 0.147574
0.150516, 0.135121
0.171759, 0.156634
0.150054, 0.105916
0.134464, 0.137982
0.143674, 0.101934
0.120071, 0.102832
0.108317, 0.113958
0.111805, 0.118596
0.110405, 0.130443
0.121226, 0.121976
0.119743, 0.132108
0.141158, 0.157200
0.106953, 0.124148
0.127743, 0.115733
0.130414, 0.106129
0.121768, 0.113167
0.119585, 0.103306
0.122795, 0.099366
0.113017, 0.138320
^C
lukecanny@lc-ubuntu:~/CT437$ ./problem_2/bin/triple_des_encrypt

7.063179, 6.160249
4.872594, 5.258295
Test Start
DES_EBE3_ECB
5.221658, 4.460576
5.076849, 5.152974
5.192108, 4.819799
5.125885, 4.743790
5.127731, 4.885641
4.917402, 4.830261
5.066426, 4.737153
```

Figure 3 Script running on an Ubuntu Virtual Machine

Appendix

GitHub Repo: https://github.com/lukecanny/CT437_A2_OpenSSL