

Lab 1: Cryptography

Christian O'meara

Willow Fussell

Elhacen Elmoustapha

Noureldin Youssef

CSE 566

Abstract

This lab introduces students to various cryptographic concepts and tools using the Kali Linux environment provided by UofL CSE's CyberPVE. Students will explore symmetric encryption, comparing different ciphers and modes (ECB vs. CBC) using the openssl software, and examine the effects of corrupted ciphertext. Additionally, the lab covers generating message digests and HMACs, with tasks to understand key size requirements and hash function properties. Students will also act as a Certificate Authority (CA), creating and issuing digital certificates. Finally, the lab includes password-based authentication exercises, involving hashing, salting, and cracking password hashes, as well as an offline analysis using password cracking tools. The goal is to deepen understanding of cryptographic methods and their practical applications in information security.

Part 1: Symmetric encryption using different ciphers and modes

In Lab 1, we experimented with several encryption and decryption symmetric ciphers using different cipher modes. We specifically worked with examples of AES(Advanced Encryption Standard) and DES(Data Encryption Standard) ciphers. Some of these ciphers require keys and initialization vectors (IVs) of specific lengths, while others do not require IVs, depending on their mode of operation. For instance, in the following encryption and decryption commands, one does not use an IV because it employs the ECB (Electronic Codebook) mode. We encrypted and decrypted multiples file. Each text file is named name_plain.txt where the name is the cipher type. Similarly, the decrypted file are named name_decrypted.bin where the name is the cipher type used to decrypt the txt file.

Each txt file contains the following text.

“encrypting and decrypting using [name of the cipher type]”

For example, “encrypting and decrypting using aes 128 cbc.”

AES

-aes-128-cbc,

-aes-128-cfb,

aes-128-ecb,

Aes-192-cbc

DES

Des-edc3-ecb

Des-ofb

```
openssl enc -aes-128-cbc -e -in aes_128_cbc_plain.txt -out aes_128_cbc.bin -K  
00112233445566778889aabbccddeeff -iv 01020304050607080000000000000000
```

```
$ openssl enc -aes-128-cbc -d -in aes_128_cbc.bin -out aes_128_cbc_decrypted.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080000000000000000
```

-aes-128-cfb,

```
openssl enc -aes-128-cfb -e -in aes_128_cfb_plain.txt -out aes_128_cfb.bin -K  
00112233445566778889aabbccddeeff -iv 01020304050607080000000000000000
```

```
openssl enc -aes-128-cfb -d -in aes_128_cfb.bin -out aes_128_cfb_decrypted.txt -K  
00112233445566778889aabbccddeeff -iv 01020304050607080000000000000000
```

aes-128-ecb,

Does not require IV

```
openssl enc -aes-128-ecb -e -in aes_128_ecb_plain.txt -out aes_128_ecb.bin -K  
00112233445566778889aabbccddeeff
```

```
openssl enc -aes-128-ecb -d -in aes_128_ecb.bin -out aes_128_ecb_decrypted.txt -K  
00112233445566778889aabbccddeeff
```

Aes-192-cbc

```
openssl enc -aes-192-cbc -e -in aes_192_cbc_plain.txt -out aes_192_cbc.bin -K  
00112233445566778899aabbccddeeff0011223344556677 -iv  
01020304050607080000000000000000
```

```
openssl enc -aes-192-cbc -d -in aes_192_cbc.bin -out aes_192_cbc_decrypted.txt -K  
00112233445566778899aabbccddeeff0011223344556677 -iv  
01020304050607080000000000000000
```

Des-ede3-ecb

Does not require IV

```
openssl enc -des-ede3-ecb -e -in des_ede3_ecb_plain.txt -out des_ede3_ecb.bin -K  
00112233445566778899aabbccddeeff0011223344556677
```

```
openssl enc -des-ede3-ecb -d -in des_ede3_ecb.bin -out des_ede3_ecb_decrypted.txt -K  
00112233445566778899aabbccddeeff0011223344556677
```

```
openssl enc -des-ofb -e -in des_ofb_plain.txt -out des_ofb.bin -K 0011223344556677 -iv  
0102030405060708
```

```
openssl enc -des-ofb -d -in des_ofb.bin -out des_ofb_decrypted.txt -K 0011223344556677  
-iv 0102030405060708
```

Openssl speed

The speed of computation is an important factor for any cryptographic operation. OpenSSL includes a built-in benchmarking tool that can help in assessing cryptographic algorithms, including both encryption and decryption operations. This benchmark can be invoked using the openssl speed command. In this lab, we analyzed the performance of symmetric encryption and decryption ciphers of type DES and AES. It's worth mentioning that `openssl speed` command traditionally defaults to testing cryptographic algorithms in modes like CBC because these modes are commonly used and standardized.

The following table indicates the performance of different ciphers using openssl speed command. We executed commands to measure speed of different AES/DES ciphers. The result actual output can be views in the following text files, AES_performance.txt and DES_performance.txt

AES ciphers: performance

Command: openssl speed aes

```

1|version: 3.0.11
2|built on: Tue Sep 19 16:58:30 2023 UTC
3|options: bn(64,64)
4|compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -fzero-call-used-regs=used-gpr -
  DOPENSSL_TLS_SECURITY_LEVEL=2 -Wa,--noexecstack -g -O2 -ffile-prefix-map=/build/reproducible-
  path/openssl-3.0.11=. -fstack-protector-strong -fstack-clash-protection -Wformat -Werror=format-
  security -fcf-protection -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -
  DOPENSSL_BUILDING_OPENSSL -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
5|CPUINFO: OPENSSL_ia32cap=0x80202001479bffff:0x0
6|The 'numbers' are in 1000s of bytes per second processed.
7|type          16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes   16384 bytes
8|aes-128-cbc    65881.55k   98181.78k   111500.63k  254759.59k  262561.79k  259276.80k
9|aes-192-cbc    59289.37k   83033.79k   91535.87k  211962.88k  218606.25k  211779.58k
10|aes-256-cbc    53442.17k   73111.95k   80284.07k  187151.36k  191466.15k  191457.96k

```

DES ciphers: performance

Command: `openssl speed des`

```

1|version: 3.0.11
2|built on: Tue Sep 19 16:58:30 2023 UTC
3|options: bn(64,64)
4|compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -fzero-call-used-regs=used-gpr -
  DOPENSSL_TLS_SECURITY_LEVEL=2 -Wa,--noexecstack -g -O2 -ffile-prefix-map=/build/reproducible-
  path/openssl-3.0.11=. -fstack-protector-strong -fstack-clash-protection -Wformat -
  Werror=format-security -fcf-protection -DOPENSSL_USE_NODELETE -DL_ENDIAN -DOPENSSL_PIC -
  DOPENSSL_BUILDING_OPENSSL -DNDEBUG -Wdate-time -D_FORTIFY_SOURCE=2
5|CPUINFO: OPENSSL_ia32cap=0x80202001479bffff:0x0
6|The 'numbers' are in 1000s of bytes per second processed.
7|type          16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes   16384 bytes
8|des-cbc        42604.34k   51741.14k   54792.36k   55648.26k   55683.75k   56164.35k
9|des-ede3       17914.75k   19509.29k   19956.57k   20188.84k   20231.51k   20288.85k

```

Types: Indicates the type of AES/DES operation/cipher and mode being tested. For example, aes-128 cbc represents AES/DES encryption and decryption using a 128-bit key in CBC (Cipher Block Chaining) mode.

Block Sizes: The columns represent different block sizes, ranging from 16 bytes to 8192 bytes. Block size indicates the size of data blocks that AES/DES processes in a one encryption/ decryption operation.

Throughput: The numbers in the table represent the throughput of AES/DES operations in thousands of bytes per second (k). This indicates how many bytes of data AES/DES can process per second for the given key size, block size, and mode of operation. Higher numbers indicate faster throughput.

Implications:

AES significantly outperforms DES in both speed and security. AES processes data much faster than DES, especially for larger block sizes, and offers stronger security with key sizes of 128, 192, and 256 bits, compared to DES's 56-bit key, which is now considered insecure. Additionally, 3DES, while more secure than DES, is still slower and less efficient than AES.

Therefore, AES is the preferred choice for modern encryption needs due to its superior efficiency and security.

Part 2: Encryption Mode – ECB vs. CBC

ECB mode encrypts each block of data independently using the same key. Consequently, if two blocks of plaintext are identical, they will produce identical ciphertext blocks. This can create patterns in the ciphertext, which attackers can exploit. Additionally, ECB mode lacks diffusion, so small changes in the plaintext lead to predictable changes in the ciphertext.

In contrast, CBC mode XORs each plaintext block with the previous ciphertext block before encryption. This ensures that even if plaintext blocks are identical, the resulting ciphertext blocks will differ due to the chaining effect. CBC mode enhances security by eliminating patterns in the ciphertext and providing diffusion, making it more difficult for attackers to analyze and exploit.

When choosing between ECB and CBC mode, always opt for CBC. ECB mode reveals information about the plaintext because identical plaintext blocks result in identical ciphertext blocks. CBC mode prevents this, making ECB mode insecure and unsuitable for use.

```
openssl enc -aes-128-cbc -e -in pic_original.bmp -out Lab_1_picture_CBC.bmp -K  
00112233445566778889aabbccddeeff -iv 01020304050607080000000000000000
```

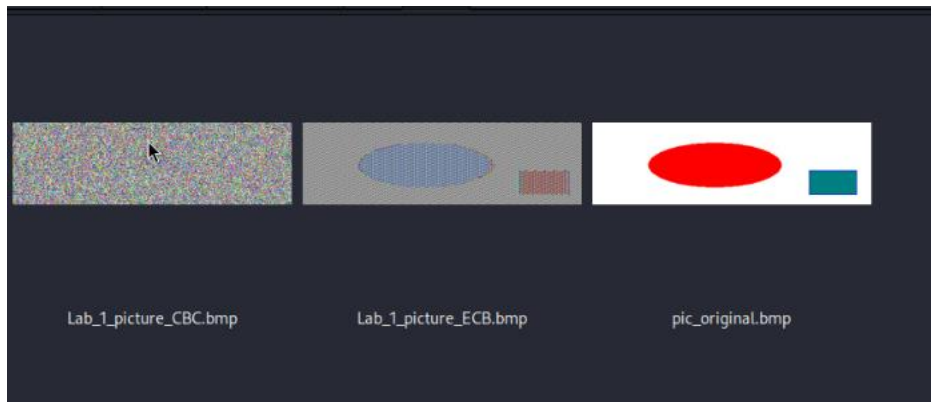
```
openssl enc -aes-128-ecb -e -in pic_original.bmp -out aes_128_ECB.bmp -K  
00112233445566778889aabbccddeeff
```

```
uofspeed@kali2023: ~/Desktop/CSE566Group4/Lab1/Part2
File Actions Edit View Help
00000000 42 4D 8E D2 02 00 00 00 00 00 36 00 BM.....6.
0000000C 00 00 28 00 00 00 CC 01 00 00 86 00 ..(.....
00000018 00 00 01 00 18 00 00 00 00 00 58 D2 .....X.
00000024 02 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 FF FF FF FF FF FF .....
0000003C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000048 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000054 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000060 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000006C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000078 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000084 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000090 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000009C FF FF FF FF FF FF FF FF FF FF FF FF .....
000000A8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000B4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000C0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000CC FF FF FF FF FF FF FF FF FF FF FF FF .....
000000D8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000E4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000F0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000FC FF FF FF FF FF FF FF FF FF FF FF FF .....
00000108 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000114 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000120 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000012C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000138 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000144 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000150 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000015C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000168 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000174 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000180 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000018C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000198 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001A4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001B0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001BC FF FF FF FF FF FF FF FF FF FF FF FF .....
000001C8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001D4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001E0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001EC FF FF FF FF FF FF FF FF FF FF FF FF .....
-- pic_original.bmp --0x0/0x2D28E--0%

uofspeed@kali2023: ~/Desktop/CSE566Group4/Lab1/Part2
File Actions Edit View Help
00000000 42 4D 8E D2 02 00 00 00 00 00 36 00 BM.....6.
0000000C 00 00 28 00 00 00 CC 01 00 00 86 00 ..(.....
00000018 00 00 01 00 18 00 00 00 00 00 58 D2 .....X.
00000024 02 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000003C 87 79 07 2F A8 76 DE 98 C3 E6 2C BF ..y./..v.....
00000048 B8 89 E5 76 01 2F 71 11 2D A3 07 4A .../.q.-..J
00000054 FF EC 12 43 9E 71 CF C7 FE C2 76 A4 ...C.q...v.
00000060 5E 64 01 E0 0B 65 65 CD 18 B3 9D A3 ^d...ee...
0000006C 06 DF 3A CE DD 14 6C E6 23 19 13 6D ..:...l.#..m
00000078 27 D2 2D 9F 6A 6E C1 B2 7D F3 13 28 ',-..jn..}...+
00000084 DE A4 46 A8 94 FC 32 38 B0 18 1B 9F ...F...2;...
00000090 A9 06 E7 7F B8 E1 BC 21 7D B7 68 38 .....1}.h8
0000009C 66 56 57 91 F8 3A 87 93 63 B1 69 AF fVW...:..c.i.
000000A8 72 EB E4 99 17 A9 9A 21 07 D9 41 8B r.....1..A.
000000B4 50 CC 73 61 81 A6 46 57 8E 8B 0E B3 P.s...FW...
000000C0 73 4F 90 9C CE E6 02 57 ED 65 D7 75 s0...W.e.u
000000CC AE 72 5E 24 0B 73 C3 63 D8 03 01 8E ..'$.s.....
000000D8 2D 22 95 D1 35 EC 06 89 5C 56 F0 78 -...5...V/x
000000E4 E1 1D BC DF 74 EA A6 99 EB 66 B0 53 ...t...f.S
000000F0 AD 01 73 F8 1C EC A3 9C 72 D7 37 EC ...S...E.T.7.
000000FC 4B 05 C8 C4 36 D7 8B 9E D9 55 02 78 K...6...U/x
00000108 FC A2 75 74 E2 5C 2A 31 8C 63 AF 7F ...ut..+1.c..
00000114 40 AC F4 09 6D EB D8 74 D6 69 14 60 @...m...i.i.'
00000120 58 DF 6C D0 59 C8 02 E4 FF 82 9E 67 X.l.Y...g
0000012C FD 1B CC 93 CF A1 B3 70 97 AE FE 8B .....p...
00000138 2C 43 C8 82 0C CF A3 EA CF 0C 93 8C ...C.....
00000144 9F 34 41 5B 2A 0A 77 15 F3 7A 9C E5 ..4A(*.w.z..
00000150 1A 22 68 92 37 C0 B9 52 56 20 D2 30 ..h.7..RV..0
0000015C 0C 5A C0 4D 08 0F 5C 14 40 66 F0 4E ..Z.M..._@f.N
00000168 4B E6 09 3D D6 A0 EA 31 F5 D9 08 64 K...~1...d
00000174 66 32 92 57 51 4F 7C A2 79 D9 75 17 f2.WQ0|.y.u.
00000180 E6 83 11 C5 27 3C F8 08 C8 A1 E7 EA ....^<.....
0000018C CA 0A 16 1A 42 3D 83 28 22 75 1C 43 ...B-('u.C
00000198 C9 89 31 A0 C9 18 0F A3 12 14 97 48 ...1.....H
000001A4 3A F9 79 F5 9D 9A 46 8F EA C4 4E C2 ..y...F...N.
000001B0 FB 74 1E 2A A5 B6 71 DE 23 DF 7B 5B ..t...q.#.{{
000001BC 06 78 FA 35 5B 13 78 89 FA 2B 40 87 ..x.5|.x...+0.
000001C8 54 A4 1F 7E 06 3F 61 A7 7F 1E 35 A4 T...~?a...5.
000001D4 23 D4 40 38 FD 5B 5B 8E AD 50 9F 16 #.00.[[...P..
000001E0 B3 7E EA 7E 06 6E 8D A3 EF DA 7E ...-..nn.....
-- Lab_1_picture_C8C.bmp --0x36/0x2D290--0%
```

```
uofspeed@kali2023: ~/Desktop/CSE566Group4/Lab1/Part2
File Actions Edit View Help
00000000 42 4D 8E D2 02 00 00 00 00 00 36 00 BM.....6.
0000000C 00 00 28 00 00 00 CC 01 00 00 86 00 ..(.....
00000018 00 00 01 00 18 00 00 00 00 00 58 D2 .....X.
00000024 02 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 FF FF FF FF FF FF .....
0000003C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000048 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000054 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000060 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000006C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000078 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000084 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000090 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000009C FF FF FF FF FF FF FF FF FF FF FF FF .....
000000A8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000B4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000C0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000CC FF FF FF FF FF FF FF FF FF FF FF FF .....
000000D8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000E4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000F0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000000FC FF FF FF FF FF FF FF FF FF FF FF FF .....
00000108 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000114 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000120 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000012C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000138 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000144 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000150 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000015C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000168 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000174 FF FF FF FF FF FF FF FF FF FF FF FF .....
00000180 FF FF FF FF FF FF FF FF FF FF FF FF .....
0000018C FF FF FF FF FF FF FF FF FF FF FF FF .....
00000198 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001A4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001B0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001BC FF FF FF FF FF FF FF FF FF FF FF FF .....
000001C8 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001D4 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001E0 FF FF FF FF FF FF FF FF FF FF FF FF .....
000001EC FF FF FF FF FF FF FF FF FF FF FF FF .....
-- pic_original.bmp --0x0/0x2D28E--0%

uofspeed@kali2023: ~/Desktop/CSE566Group4/Lab1/Part2
File Actions Edit View Help
00000000 42 4D 8E D2 02 00 00 00 00 00 36 00 BM.....6.
0000000C 00 00 28 00 00 00 CC 01 00 00 86 00 ..(.....
00000018 00 00 01 00 18 00 00 00 00 00 58 D2 .....X.
00000024 02 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000003C FB 5E F8 E6 47 C9 96 BB 3C 11 C7 0A ...^..G...<...
00000048 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
00000054 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
00000060 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
0000006C F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
00000078 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
00000084 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
00000090 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
0000009C F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
000000A8 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
000000B4 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
000000C0 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
000000CC F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
000000D8 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
000000E4 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
000000F0 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
000000FC F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
00000108 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
00000114 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
00000120 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
0000012C F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
00000138 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
00000144 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
00000150 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
0000015C F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
00000168 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
00000174 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
00000180 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
0000018C F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
00000198 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
000001A4 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
000001B0 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
000001BC F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
000001C8 BC F0 81 BF F0 99 9A B8 47 C9 96 BB <.....G...
000001D4 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 <.....G...
000001E0 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF G...<.....G...
000001EC F0 99 9A B8 47 C9 96 BB 3C 11 C7 0A <.....G...<...
-- Lab_1_picture_ECB.bmp --0x0/0x2D290--0%
```



The ECB encrypted image reveals patterns that can give away some information about the original image, making it less secure. This is because ECB mode encrypts each block of data independently without any interdependency. Therefore, identical plaintext/image blocks result in identical ciphertext/image blocks, leading to the visibility of repeating patterns.

In contrast, the CBC encrypted image appears completely random and does not reveal any useful information about the original image, making it more secure. This is because CBC mode uses an initialization vector (IV) and each block of the image data is combined with the previous encrypted block before being encrypted itself.

Part 3: Encryption Mode – Corrupted Cipher Text

How much information can be recovered by decrypting a file where one of the bytes in the encrypted file has been corrupted?

The amount of information that can be recovered from a decrypted file with one byte corrupted will depend on the encryption method used

- **Block Cipher in Electronic Codebook (ECB) Mode:** In ECB mode, each block of data is encrypted independently. If a byte in an encrypted block is corrupted, it typically affects only its output during decryption.
- **Block Cipher in Cipher Block Chaining (CBC) Mode:** In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. Therefore, a single byte corruption in one block affects its decryption and corrupts the beginning of the next block (due to the chaining).
- **Stream Ciphers:** For stream ciphers, the plaintext is combined with a pseudorandom cipher digit stream (keystream) using bitwise operations such as XOR. Corruption in one byte of the ciphertext will corrupt exactly one byte of the plaintext upon decryption. Thus, all information except the corrupted byte can be perfectly recovered.
- **Authenticated Encryption Modes (e.g., GCM, CCM):** These modes provide both encryption and integrity checking. If a byte is corrupted, the integrity check (authentication tag) will typically fail during decryption. This means that the decryption process will either refuse to output any plaintext or will signal that the data could not be authenticated, depending on the specific mode and implementation. In practice, this means no information is reliably recovered without additional error-correcting measures.

```
(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ openssl enc -aes-128-cbc -in plaintext.txt -out encrypted.dat -K $(cat key.hex) -iv $(cat iv.hex)

(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ openssl enc -aes-128-cbc -d -in encrypted.dat -out decrypted.txt -K $(cat key.hex) -iv $(cat iv.hex)

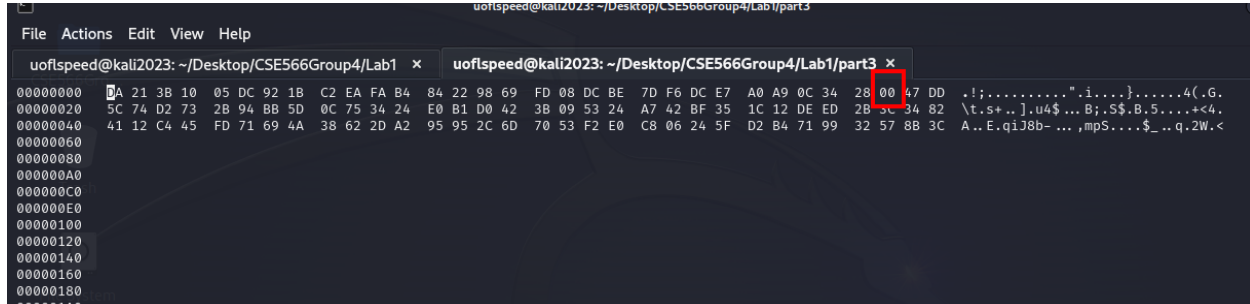
(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ cat decrypted.txt
This is a sample file that will use for the encryption. It has more than 64 bytes

(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ hexedit encrypted.dat

(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ cat decrypted.txt
This is a sample file that will use for the encryption. It has more than 64 bytes

(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ openssl enc -aes-128-cbc -d -in encrypted.dat -out decrypted.txt -K $(cat key.hex) -iv $(cat iv.hex)

(uoflspeed@ kali2023) - [~/Desktop/CSE566Group4/Lab1/part3]
$ cat decrypted.txt
This is a sample file that will use for the encryption. It has more than 64 bytes
```



In your lab write up reflect on the answer you gave before conducting the exercise, was your thinking correct? Why or why not? What are some implications of what you learned as a result of doing the activity?

I hypothesized that a corrupted byte in the encrypted file would primarily affect only the block containing that byte.

Results

Decrypting the corrupted AES-128-CBC encrypted file showed that not only the block with the corrupted byte was affected but also the subsequent block. This occurs because, in CBC mode, each block is XORed with the previous ciphertext block, so corruption propagates to at least one additional block.

Analysis of Results

My initial prediction underestimated the impact of corruption in CBC mode. The results revealed significant error propagation affecting multiple blocks, which was not initially anticipated.

Part 4: Generating Message Digest and MAC

We used the following three one-way:

SHA-256, MD5, SHA-1

```
(uoflspeed@kali2023) - [~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -md5 testfile.txt
MD5(testfile.txt)= e9fe0163fd69b80b738391145a817305

(uoflspeed@kali2023) - [~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -sha1 testfile.txt
SHA1(testfile.txt)= 6d6d917e54e3390a680129c8eae372db6440711

(uoflspeed@kali2023) - [~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -sha256 testfile.txt
SHA2-256(testfile.txt)= ae6ce50d69fee27fe70ba5f040a392f16b89e928a6a1cfc387dd2fdd2e96609b
```

```
(uoflspeed@kali2023)-[~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -md5 -hmac "simplekey" testfile.txt
HMAC-MD5(testfile.txt)= 45faa8bb425199742a375b9124aafb97

(uoflspeed@kali2023)-[~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -sha256 -hmac "simplekey" testfile.txt
HMAC-SHA2-256(testfile.txt)= 4953846c760130e3a0b7142400dadbbfc61909d0af027b4e0b5142b1949e0b08

(uoflspeed@kali2023)-[~/Desktop/CSE566Group4/Lab1/part4]
$ openssl dgst -sha1 -hmac "longerkeyexample" testfile.txt
HMAC-SHA1(testfile.txt)= 0ccb50c00c7278acd40ffe8bb516888ad10ffc79
```

Answer the following questions in the lab report:

Do we have to use a key with a fixed size in HMAC? If so, what is the key size? If not, why?

- No, the key in HMAC does not need to be of a fixed size. HMAC processes keys internally to fit the block size of the hash function. If the key is longer than the hash block size, it is hashed to shorten it. If it's shorter, it is padded.
- Experiment with keys of varying lengths to see the effect on the HMAC output.

How dissimilar are they? Write a short program (any language) to count how many bits are different between H1 and H2. In your lab report include H1, H2, your code (or at least the main logic part of it), and the output of the program.

```
def count_diff_bits(hex1, hex2):
    # Convert hex to binary
    bin1 = bin(int(hex1, 16))[2:].zfill(128) # Ensuring full length
    bin2 = bin(int(hex2, 16))[2:].zfill(128)

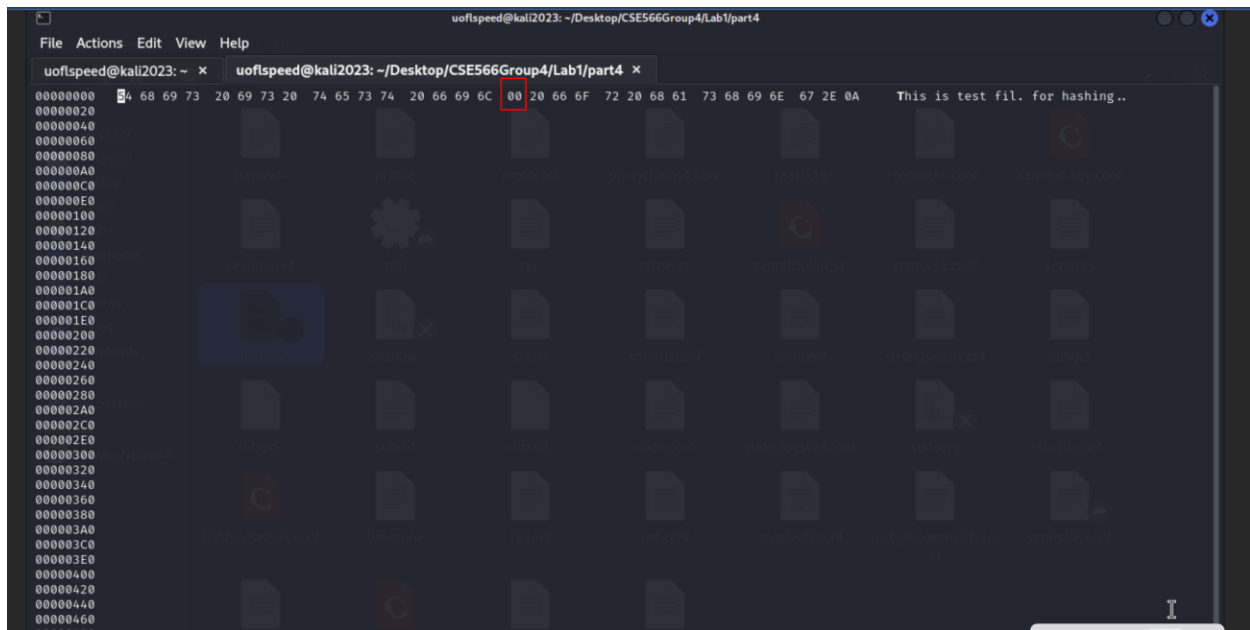
    # Count differences
    diff = sum(b1 != b2 for b1, b2 in zip(bin1, bin2))
    return diff

# Read hash values from files
with open('hash1.txt', 'r') as file:
    hash1 = file.read().strip()

with open('hash2.txt', 'r') as file:
    hash2 = file.read().strip()

# Calculate and print the number of differing bits
difference = count_diff_bits(hash1, hash2)
print(f"Differences in bits: {difference}")
```

Editing with hex editor



Flipping one bit in the input file resulted in 59 bits being different between the two MD5 hash outputs, H1 and H2. This demonstrates the huge effect, where a minimal change in the input causes changes in the output hash, ensuring that even small changes are detectable.

Part 5: Become a Certificate Authority (CA)

Using the command:

```
cp /usr/lib/ssl/openssl.cnf ./openssl.cnf
```

We were able to copy the default OpenSSL configuration file to our working directory. The following directories were then created: demoCA/certs, demoCA/crl, demoCA/newcerts, and demoCA/private, along with the files demoCA/index.txt and demoCA/serial with an initialized value of 1000.

The openssl.cnf file was then modified to ensure the required fields were included. After the preparation, the next step was to generate a self-signed certificate for the CA. We began with the command:

```
openssl req -new -x509 -days 3650 -key demoCA/private/ca.key -out  
demoCA/certs/ca.crt -config openssl.cnf -subj  
'/CN=www.pkilabserver.com/O=pkilabserver/C=US/ST=Kentucky'
```

And the certificate was verified using:

```
openssl x509 -noout -text -in demoCA/certs/ca.crt
```

Following this, we created a certificate for the server “pkilabserver.com” by generating the public/private key pair for the server using:

```
openssl genpkey -algorithm RSA -out server.key -aes256
```

And creating a CSR using:

```
openssl req -new -key server.key -out server.csr -config openssl.cnf -subj  
'/CN=www.pkilabserver.com/O=pkilabserver/OU=IT  
Department/C=US/ST=Kentucky/L=Louisville'
```

The CSR was then signed with the CA’s certificate using:

```
openssl ca -in server.csr -out server.crt -cert demoCA/certs/ca.crt -keyfile  
demoCA/private/ca.key -config openssl.cnf
```

Public Key of CA:

```
File Actions Edit View Help

(uoflspeed@ kali2023) ~
$ cat demoCA/certs/ca.crt
-----BEGIN CERTIFICATE-----
MIIDizCCAnOgAwIBAgIU0/D2WCUKAo2x9jnLOU8AeIgNPtcwDQYJKoZIhvcNAQEL
BQAwVTEcMBoGA1UEAwwTd3d3LnRlbGFiY2VydmVybWVudGVudHVja3kwHhcNMjQw
bGFiY2VydmVybWVudGVudGVudGVudGVudGVudGVudGVudGVudGVudGVudGVudGVudGV
NjA5MDQ1NTAwWhcNMzQwNjA3MDQ1NTAwWjBVMRwwGgYDVQQDDBN3d3cucGtsYWJz
ZXJ2ZXIuY29tMRUwEwYDVQQKDAxwa2lsYWJzZXJ2ZXIuY29tMRUwEwYDVQQIDAh
LW50dWwreTCCASiWdQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
A3mUd44/35Y2OvIT0p0fzpas449ncZER0zmLIhSn4+w191gCKpuM6Nq0rGGAfqWT
aj6FhtgLIh8DlcRxxkugFx04tH3jsMeyeY2rFkPmrNFrRtS+IHD1/F4fM5VJBrIME
tHqf7Rie9BLBM0wGic/D/Low0VFSF+06KUE8UGom59WwTTXehsZQ08Mx/V4EKp3L
CruCJ0E6ZovDG1SKJ0qEfrrvvji0E4znechtZCihN7fffarUfF8bcVy8IbLwpu5V
1ksXNHx1puAyjsX+rScvpWbRWzMI6cHgQH032U0xkv7Ub4mNmX47cEbQ4fUhlUY1
84y/C0NqfpZ0a4RV8cDgQksCAwEAANTMFwHQYDVIR00BBYEFKmqnALJLg6GKtFn
oVj0cAbDImo4MB8GA1UdIwQYMBaAFKmqnALJLg6GKtFnoVj0cAbDImo4MA8GA1Ud
EwEB/wQFMAMBAFBwDQYJKoZIhvcNAQELBQADggEBAQmEZ0NnUXcMhjZFszNCrvIT
SLMo/s0GfPC70/U10Jkqj6JDC5RYMw5xEqURfWB6k2LFpOAH/SeYf0Ww7BC5tcI4
czx3azb/Ynya/xeexjDyQV/csWmYjYBxG1eD3oraJ+zYgQIHLr3c0/ZyZcaCzP87
vb/fxNnFLBK9RKcXLJjxhbRAVf+xEBc+erEo7wTATkchSgJ5KJRULbzR8WgPm5WP
4yCEf0UN64TncBXLiNMh9LynTIRky8d9J0AR9vF90H7JAPOkE0/NRFQ2HUCnHIfb
EX8PbhY8or4soZzngCKH4PAXNf0kpE/DhXdF23ILxExo6cbfrzhbb7GM1kpgZ2g-
-----END CERTIFICATE-----

(uoflspeed@ kali2023) ~
```

Details of Signed Certificate:


```

uoflspeed@kali2023: ~
File Actions Edit View Help

(uoflspeed@kali2023) [~]
$ cat server.crt
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 4096 (0x1000)
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: CN=www.pkilabserver.com, O=pkilabserver, C=US, ST=Kentucky
  Validity
    Not Before: Jun  7 03:34:40 2024 GMT
    Not After : Jun  7 03:34:40 2025 GMT
  Subject: C=US, ST=Kentucky, O=pkilabserver, OU=IT Department, CN=www.pkilabserver.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:b7:72:33:4d:4d:34:7c:89:25:ad:7e:24:69:ce:
      e2:7c:96:9d:05:f2:e4:8f:3b:cb:f2:53:07:6d:
      0a:4b:54:ca:28:af:47:d9:d8:27:db:a6:ea:2e:68:
      07:85:f3:2c:c0:b6:b6:49:1a:3a:04:19:7d:2b:7e:
      34:85:9d:a9:77:c3:40:06:d1:d0:3a:2c:5f:bf:16:
      82:42:86:f7:f1:f4:06:56:34:fe:26:26:24:d4:6a:
      b2:c9:d3:4d:22:a0:27:f2:08:b4:37:5b:42:d0:6d:
      cb:3d:54:dc:75:f4:16:54:d3:1c:b3:d2:58:de:c1:
      7c:cc:d9:e8:10:09:da:78:23:76:e2:d1:b5:42:87:
      5b:a7:d1:ab:64:03:de:64:28:a0:33:e5:5a:14:31:
      7a:ef:40:b5:16:08:fc:b6:84:2c:0d:6e:97:71:0f:
      2d:7f:56:e1:1f:f2:a7:70:98:0f:58:60:04:c6:d7:
      a4:88:33:29:08:2d:ea:7a:4e:50:b2:c1:7d:34:54:
      42:33:63:8a:aa:a4:f8:fc:34:6c:ea:9f:34:ea:aa:
      f3:22:36:d4:80:75:df:f2:0a:75:d2:80:e0:e5:57:
      4b:8f:d2:d6:60:42:da:49:bb:89:9f:7b:81:80:4e:
      92:fa:52:6b:ec:a8:1e:67:55:5f:0b:28:64:9c:09:
      37:21
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
    X509v3 Subject Key Identifier:
      16:BC:8A:71:34:88:E1:4F:1D:08:AF:2B:8F:FC:67:6D:CA:38:38:82
    X509v3 Authority Key Identifier:
      A9:90:9D:A2:C9:2E:0E:06:2A:D1:67:A1:58:CE:70:06:C3:20:CA:38

```

Section 1: Password based authentication

The hash types in this table were found by using Hashcat without a `-m` specifier, which defaulted Hashcat to attempting to find out the most likely types of hashes for each hash. This was then verified by creating five text files, each of the first four had a separate hash in it, and the fifth had the given password, “password” in it. A dictionary attack was then performed on each hash with the specified `-m` type to be the top value in the Hashcat assumptions, and when the hash was cracked with the appropriate output, the hash type was then verified and logged in the table.

Hash	Password	Hash Type
5f4dcc3b5aa765d61d8327deb882cf99	password	MD5
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8	password	SHA1
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8	password	SHA2-256

b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffa0cacbc86	password	SHA2-512
---	----------	----------

Openssl screenshot:

```
(uoflspeed@kali2023)-[~]
$ echo -n password | openssl md5
MD5(stdin)= 5f4dcc3b5aa765d61d8327deb882cf99

(uoflspeed@kali2023)-[~]
$ echo -n password | openssl sha1
SHA1(stdin)= 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8

(uoflspeed@kali2023)-[~]
$ echo -n password | openssl sha256
SHA2-256(stdin)= 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

(uoflspeed@kali2023)-[~]
$ echo -n password | openssl sha512
SHA2-512(stdin)= b109f3bbbc244eb82441917ed06d618b9008dd09b3befd1b5e07394c706a8bb980b1d7785e5976ec049b46df5f1326af5a2ea6d103fd07c95385ffa0cacbc86

(uoflspeed@kali2023)-[~]
$
```

To crack the four hashes given under a. through d., a Hashcat MD5 dictionary attack using rockyou.txt was initially used. This resulted in hashes 5f4dcc3b5aa765d61d8327deb882cf99, 819b0643d6b89dc9b579fdcf9094f28e, 1c625cc86f824660a320d185916e3c55, and cf7a5b3016903a29bc0c17cfca1e584f being cracked and revealed as password, password3, russia, and may2011, respectively. MD5 was assumed due to hash a. being the same as hash 1. given in the prompt.

The Crypt Linux function was used to hash this password. The structure of the hashed password is separated by \$ characters. Below is an image of the hashed passwords in the shadow file.

```
nm-openconnect:!:19648:::::  
uoflspeed:$y$j9T$2YrPfd3GgzZnIkONy84nb1$3.s/6Sjg0lXKNsdLzQRfBdoiABhLuUUu6c8WIMDUgn0:19648:0:99999:7:::  
_galera:!:19648:::::  
willow1:$y$j9T$1pHjZzGydgx0Ct789D6af1$eD6Zttf6PwbWYSA20MZTCGD.HjRtysp4RLN1GjCkWP6:19884:0:99999:7:::
```

According to the man page for crypt(5), the, \$y\$, means that the password is hashed using yescrypt. The second chunk, \$j9T\$, corresponds to the options used when generating the password hash. The third chunk, \$1pHjZzGydgx0Ct789D6af1\$, corresponds to the salt used for the hash. The fourth chunk is the hashed password. This information was all gained through the man crypt pages.

Salts are used in hashing to prevent the same two passwords from having the same hash value after being put through the hashing function. The shadow file is used to protect the various passwords of the users on the system. To access the shadow file, sudo permissions have to be given to open the file and read it.

Offline Analysis:

Attempting to crack the root with Hashcat resulted in an error of CL_OUT_OF_RESOURCES. A patch had to be applied to the Windows machine being used for the process. Attempting to crack the root using Hashcat with the rockyou.txt wordlist resulted in Hashcat reporting exhausted. Cracking the user joey with the previously mentioned settings resulted in the password hash being cracked, which showed the password to be jesus. Cracking the user alice resulted in the password hash being cracked, which showed the password to be password1. Cracking the user bob with the previously mentioned settings resulted in Hashcat reporting exhausted as well. Cracking the user tom with the previously mentioned settings resulted in the password hash being cracked, which showed the password to be mookie.

Conclusion

This lab provided hands-on experience with various cryptographic techniques and tools, enhancing our understanding of symmetric encryption, encryption modes, message digests, and the role of Certificate Authorities (CAs). We observed the performance and security differences among encryption algorithms, the importance of choosing appropriate encryption modes like CBC over ECB, and the critical role of hash functions in ensuring data integrity. Additionally, we gained practical knowledge in generating and managing digital certificates and the importance of salting and hashing passwords for security.