EECS565 Intro to Computer and Information Security

# Mini Project 2

# **Secret-Key Encryption**

# Outline

- Task 1: Encryption using Different Ciphers and Modes
  - Overview the modes of operations
  - Visualize the difference
- Task 2: Padding
  - Who needs padding and how to add padding?
- Task 3: Error Propagation – Corrupted Cipher Text
  - Learn the difference in error propagation using different modes
- Task 4: IV and Common Mistakes
  - What will happen if the attacker knows the IV?

# Task 1: Ciphers

- OpenSSL Enc
  - The basic usage is to specify a ciphername and various options describing the actual task

    `$ openssl enc -ciphername [options]`

  - To get a list of available ciphers you can use the list -cipher-algorithms command

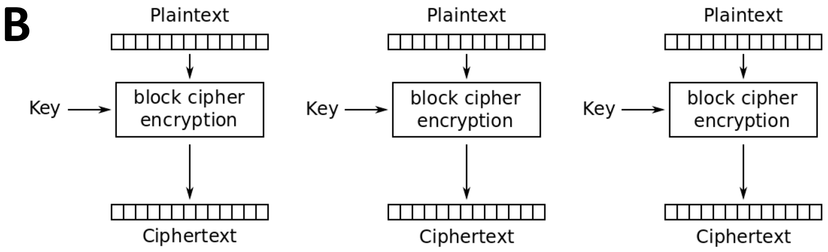
    `$ openssl list -cipher-algorithms`

```
AES-128-CBC
AES-128-CFB
AES-128-CFB1
AES-128-CFB8
AES-128-CTR
AES-128-ECB
AES-128-OFB
AES-128-XTS
```

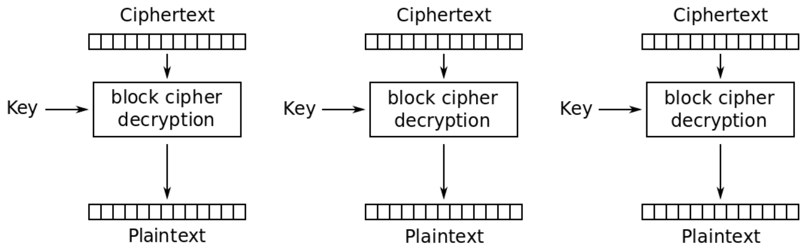
  - Options we will use in this mini project
    - **-in** filename -- This specifies the input file.
    - **-out** filename -- This specifies the output file. It will be created or overwritten if it already exists.
    - **-e** or **–d** -- This specifies whether to encrypt (-e) or to decrypt (-d). Encryption is the default.
    - **-iv** IV -- This specifies the initialization vector IV as hexadecimal number.
    - **-K** key -- This option allows you to set the key used for encryption or decryption.
    - **-nopad** -- This disables standard padding

# Ciphers

## ECB

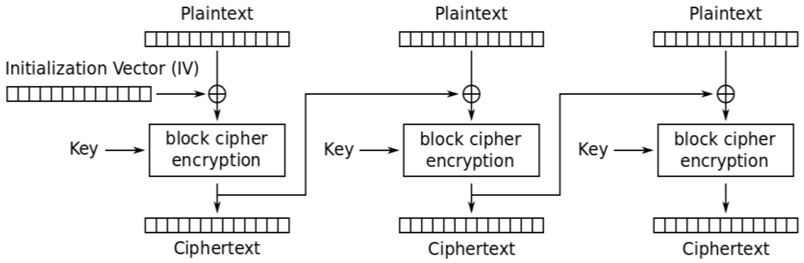



Electronic Codebook (ECB) mode encryption

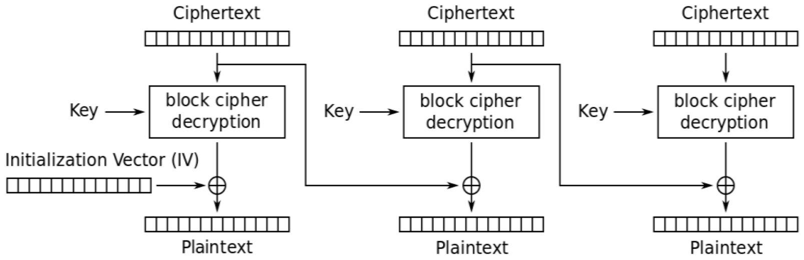



Electronic Codebook (ECB) mode decryption

## CBC




Cipher Block Chaining (CBC) mode encryption




Cipher Block Chaining (CBC) mode decryption

```
$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
         -K 00112233445566778899AABBCCDDEEFF
$ openssl enc -aes-128-ecb -d -in cipher.txt -out plain2.txt \
         -K 00112233445566778899AABBCCDDEEFF
```

```
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt  \
              -K  00112233445566778899AABBCCDDEEFF \
              -iv 000102030405060708090a0b0c0d0e0f
```

The length of IV equals the block size (AES is 16)

Note: Due to different inputs of different encryption mode , some modes need IV, the others do not. When you are trying different modes, make sure whether the mode needs IV or not.

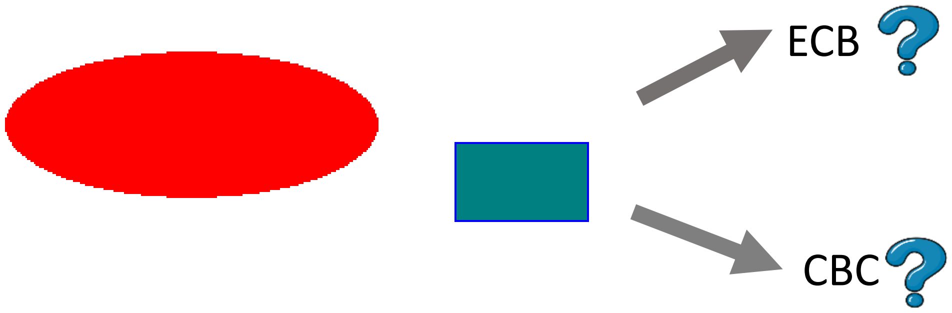
# Encryption Mode Differences

- Encrypt the picture in the same way as previous
- The first 54 bytes contain header information about the picture.
  - These 54 bytes are needed in plaintext, so that a picture viewer can recognize the file’s type.
- We replace the header of the encrypted picture with the header of the original picture.
  - Hint: we can use following commands:

```
$ head -c 54 file1 > header
$ tail -c +55 file2 > data
$ cat header data > new.bmp
```

– Use ECB to encrypt a picture:

plaintext ciphertext



# Task 2: Padding

- Block ciphers divide the plaintext into blocks.
  - So, the size of each block should match the cipher's block size.
  - But there is no guarantee that the size of the last block matches the cipher's block size.
  - So, the last block of the plaintext needs padding.

- How to pad?
  - Before encryption, we add extra data to the last block of the plaintext. So, its size equals to the cipher's block size.
  - The padding schemes need to clearly mark where the padding starts. So, decryption can remove the padded data.

- A commonly used padding scheme is PKCS#5.

# Padding Experiment

- You can simply use the command ls –ld filename to check the sizes of the original file and the encrypted file.

```
[02/06/23]seed@VM:~/MP2$ echo -n "1234567890" > plain.txt
[02/06/23]seed@VM:~/MP2$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin
-K 00112233445566778899aabbccddeeff -iv 0102030405060708091011121314151б
[02/06/23]seed@VM:~/MP2$ ls -ld cipher.bin
-rw-rw-r-- 1 seed seed 16 Feb  6 15:51 cipher.bin
[02/06/23]seed@VM:~/MP2$ openssl enc -aes-128-cbc -d -in cipher.bin -out plain2.txt
 -K 00112233445566778899aabbccddeeff -iv 01020304050607080910111213141516
[02/06/23]seed@VM:~/MP2$ ls -ld plain2.txt
-rw-rw-r-- 1 seed seed 10 Feb  6 15:51 plain2.txt
```

- For example, encrypt a file with aes-128-cbc
  - We create a plaintext(plain.txt) whose size is 10 bytes.
  - Size of ciphertext (cipher.bin) becomes 16 bytes.
  - After decryption, the plaintext(plain2.txt) size is still 10 bytes.

# Padding Experiment

- How does the decryption add the paddings?
- Use *hexdump* or *xxd* to explore the rules.
  - Decryption will automatically remove the padding by default.
  - We can use the option “-nopad” to disable padding during the decryption.

```
[02/06/23]seed@VM:~/MP2$ openssl enc -aes-128-cbc -d -in cipher.bin -out plain2.txt
 -K 00112233445566778899aabbccddeeff -iv 0102030405060708091011121314151б -nopad
[02/06/23]seed@VM:~/MP2$ xxd -g 1 plain.txt
00000000: 31 32 33 34 35 36 37 38 39 30                    1234567890
[02/06/23]seed@VM:~/MP2$ xxd -g 1 plain2.txt
00000000: 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06  1234567890......
[02/06/23]seed@VM:~/MP2$ hexdump -C plain.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[02/06/23]seed@VM:~/MP2$ hexdump -C plain2.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
```

  - For example, 6 bytes of **0x06** are added as the padding data.

# Padding Experiment – Special Case

- What if the size of the plaintext is already a multiple of the block size
  - Block size is 16 bytes in this case for the AES cipher.
  - Repeat the steps in exploring the padding roles.
  - What is the padding rule under this special case?

```
[02/06/23]seed@VM:~/MP2$ echo -n "0123456789abcdef" > plain3.txt
[02/06/23]seed@VM:~/MP2$ openssl enc -aes-128-cbc -e -in plain3.txt -out cipher3.bin
 -K 00112233445566778899aabbccddeeff -iv 01020304050607080910111213141516
[02/06/23]seed@VM:~/MP2$ openssl enc -aes-128-cbc -d -in cipher3.bin -out plain3_new
.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080910111213141516 -nopad
```

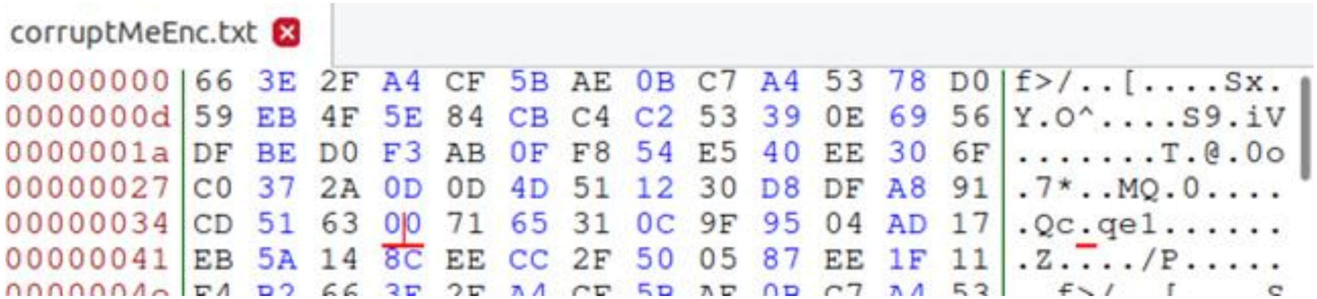
```
[02/06/23]seed@VM:~/MP2$ xxd -g 1 plain3.txt
00000000: 30 31 32 33 34 35 36 37 38 39 61 62 63 64 65 66  0123456789abcdef
[02/06/23]seed@VM:~/MP2$ xxd -g 1 plain3_new.txt
00000000: 30 31 32 33 34 35 36 37 38 39 61 62 63 64 65 66  0123456789abcdef
00000010: 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10  ................
```

# Task 3: Error Propagation

- Create a text file(e.g., f2.txt) that is at least 1000 bytes long.

  $ yes "this is a test file" | head -c 1KB > f2.txt

- Encrypt f2.txt using -aes-128-xxx, where xxx stands for ECB, CBC, CFB, and OFB
  - Assume the block size is 16 bytes and a single bit of the 56th byte in the encrypted file got corrupted during transmission
  - Use the **Bless** hex editor to view and modify file content.




Hint: press insert button to change to overwrite mode, or you will just add new value instead of modifying the value.

# Error Propagation

- Decrypt using the original key and IV
- How much information can you recover by decrypting the corrupted file?
- Compare the decrypted plaintext file with the original (in hex editor) to observe exactly how many bytes are corrupted.
  - Example of ECB mode:

```
00000020  73 74 20 66 69 6c 65 0a  74 68 69 73 20 69 73 20  |st file.this is |
00000030  99 22 c8 78 e1 5f 4f 1f  59 48 da 55 98 ee ac 27  |.".x._O.YH.U...'|
00000040  20 69 73 20 61 20 74 65  73 74 20 66 69 6c 65 0a  | is a test file.|
```

- Repeat the steps above for each of the four modes.

# Task4: IV and Common Mistakes

- How to select and use initial vectors in encryption?
  - IV is supposed to be stored or transmitted in plaintext.
  - IV should not repeat (unique).
  - IV should not be predictable(random).

- In this task, you'll encrypt the same plaintext under aes-128-cbc using:

  (1) the same IV and the same key

  (2) two different IVs and  same key

Plaintext: This is a known message!

Ciphertext:  *openssl enc* ***-aes-128-cbc***

Ciphertext_same: *openssl enc* ***-aes-128-cbc–iv <same>  -K <same>***

Ciphertext_different: *openssl enc* ***-aes-128-cbc same –iv <different> –K <same>***

*Diff* Ciphertext Ciphertext_same:

*Diff* Ciphertext Ciphertext_different:

# IV and Common Mistakes

- Can we use the same IV if the plaintext does not repeat?
  - Known-plaintext attack

| |
|---|
| Plaintext P1: This is a known message! |
| Ciphertext C1: a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159 |
| Plaintext P2: (unknown, try to decrypt) |
| Ciphertext C2: bf73bcd3509299d566c35b5d450337e1bb175f903fafc159 |

    - Given P1, C1, and C2 as above, the attacker can figure out the actual content of P2.
    - We provide a sample Python program. You can make simple modifications to make your calculation easier. Use it to help you recover P2.
- How much can you reveal if aes-128-ofb is used?
- How about aes-128-cfb?

```python
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))

MSG   = "A message"
HEX_1 = "aabbccddeeff1122334455"
HEX_2 = "1122334455778800aabbdd"

# Convert ascii string to bytearray
D1 = bytes(MSG, 'utf-8')

# Convert hex string to bytearray
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2)
r2 = xor(D2, D3)
r3 = xor(D2, D2)
print(r1.hex())
print(r2.hex())
print(r3.hex())
```

XOR operation

Strings to be processed

# Acknowledgements

- We thank SEED Crypto Lab for sharing the lecture slides.