2. Block Ciphers - AES

A. OVERVIEW

1. Learning objective

The **learning objective** of this lab is for students to get familiar with the concepts in secret-key encryption, particularly in Advanced Encryption Standard (AES) cipher. After finishing the lab, students should gain first-hand experience with encryption algorithms, encryption modes, padding, and initial vector. Moreover, students will be able to use crypto libraries to write an application. This lab will cover the following topics:

1. Secret-key encryption
2. Encryption modes, IV and paddings
3. Common mistakes in using encryption algorithms
4. Programming using the crypto library

2. Backgrounds and Prerequisites

To ensure everything goes smoothly and you achieve better results in this lab, you are expected to be familiar with the following cryptography concepts

Stream cipher

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time (Figure 1). In the ideal case, a one-time pad version of the Vernam cipher would be used, in which the keystream (k_i) is as the plaintext bitstream (p_i). If the cryptographic keystream is random, then this cipher is **unbreakable** by any means other than acquiring the keystream (**perfect secrecy**). However, the keystream must be provided to both users in advance via some independent and secure channel.

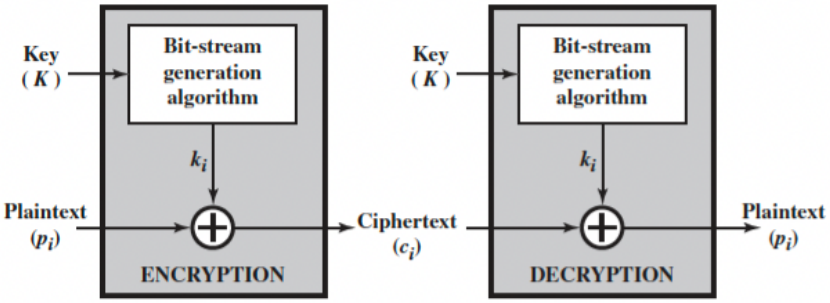

Figure 1: Stream cipher

Block cipher

A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length (Figure 2). Typically, a block size of 64 or 128 bits is used. As with stream cipher, the two users share a symmetric encryption key. Using some of the **modes of operation**, a block cipher can be used to achieve the same effect as a stream cipher.

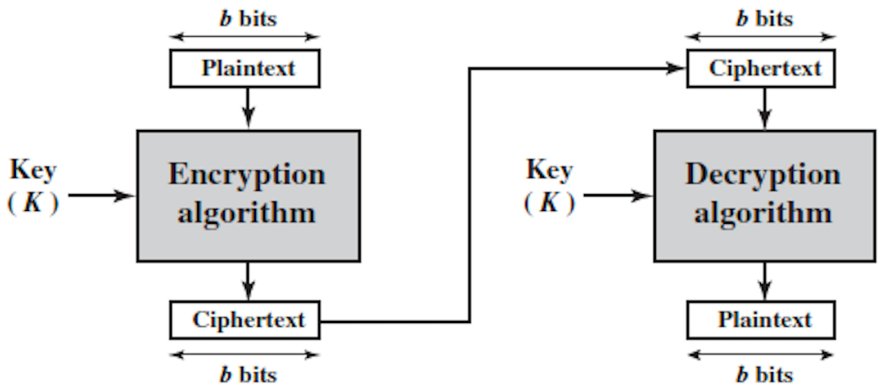

Figure 2: Block cipher

The Advanced Encryption Standard (AES) algorithms

AES *(Advanced Encryption Standard)* is a symmetric block cipher that is intended to replace DES *(Data Encryption Standard)* as the approved standard for a wide range of applications. It was published by the National Institute of Standard and Technology (NIST) in 2001.

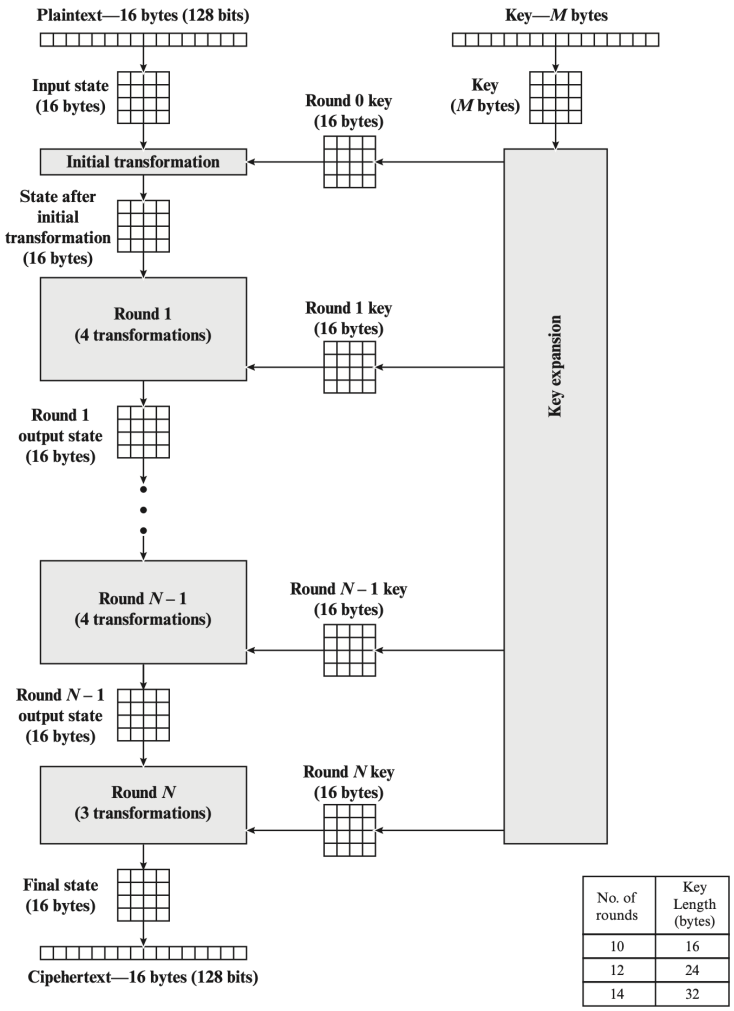

Figure 3 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.

The input to the encryption and decryption algorithms is a single 128-bit block. The cipher consists of N rounds, where the number of rounds depends on the key length: 10 round for a 16-byte key, 12 rounds for a 24-byte key, and 14 rounds for a 32-byte key.

Figure 3: AES Encryption Process

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

Figure 4: AES parameters

The modes of operation of block cipher

A block cipher takes a fixed-length block of text of length *b bits* and a key as input and produces a *b-bit* block of ciphertext. If the amount of plaintext to encrypted is greater than *b bits,* then the block cipher can still be used by breaking the plaintext up into *b-bit* blocks. When multiple blocks of plaintext are encrypted using the same key, several security issues arise.

Modes of operation is a technique for enhancing the effect of a cryptographic algorithm or adapting the algorithm for an application, such as applying a block cipher to a sequence of data blocks or a data stream. The five modes are intended to cover a wide variety of applications of encryption for which a block cipher could be used. To apply a block cipher in a variety of applications, five modes of operation have been defined by NIST (SP 800-38A):

1. **ECB** – Electronic Codebook
2. **CBC** – Cipher Block Chaining
3. **CFB** – Cipher Feedback
4. **OFB** – Output Feedback
5. **CTR** - Counter

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of plaintext bits is encoded independently using the same key.	• Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next block of plaintext and the preceding block of ciphertext.	• General-purpose block-oriented transmission • Authentication
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	• General-purpose stream-oriented transmission • Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding encryption output, and full blocks are used.	• Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	• General-purpose block-oriented transmission • Useful for high-speed requirements

Figure 5: Block Cipher Modes of Operation

You should refer to the following book section for more details:

[1] Chapter 4.1: Traditional Block Cipher structure
[2] Chapter 6: Advanced Encryption Standard
[3] Chapter 7: Block Cipher operation

William Stallings, *Cryptography and network security: Principles and practice, 7th ed, Pearson Education, 2017.*

B. LAB TASKS

1. Modes of operation of block cipher

Before working with the block cipher, we will kick off with modes of operation.

1. List and briefly define the block cipher modes of operation. What are the characteristics, advantages, and disadvantages of these modes?
2. How to encrypt plaintext in parallel on multiple blocks in CBC mode? How about decryption?
3. Create a text file of 100 bytes beginning with your name and student's ID. Encrypting this file as plaintext with at least three modes using OpenSSL (key and IV (if necessary) are your chosen).

We can use the following `openssl enc` command to encrypt/decrypt a file. To see the manual, you can type `man openssl` and `man enc`.

```
$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \
          -K 00112233445566778889aabbccddeeff \
          -iv 0102030405060708
```

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc, -bf-cbc,` etc. You can find the meaning of the command-line options and all the supported cipher types by typing `"man enc"`. We include some common options for the `openssl enc` command in the following:

-in <file>	input file
-out <file>	output file
-e	encrypt
-d	decrypt
-K/-iv	key/iv in hex is the next argument
-[pP]	print the iv/key (then exit if -P)

2. Encryption Mode – ECB vs. CBC

The file pic_original.bmp that include in this lab is a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture and use picture viewing software to display it. However, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. We can use the bless hex editor tool (already installed on our VM) to directly modify binary files. We can also use the following commands to get the header from p1.bmp, the data from p2.bmp (from offset 55 to the end of the file), and then combine the header and data together into a new file.

```
$ head -c 54 p1.bmp  > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

2. Display the encrypted picture using a picture viewing program. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.
3. Select a picture of your choice, repeat the experiment above, and report your observations.

3. Padding

For block cipher, when the size of plaintext is not a multiple of the block size, padding may be required. The **PKCS#5** padding scheme is widely used by many block ciphers

We will conduct the following experiments to understand how this type of padding works:

1. Use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

2. Practice about block cipher padding
   Phase 1: Let's create three files, which contain 5 bytes, 10 bytes, and 16 bytes, respectively. Please note that at least one of these files must included your student's id and all of them may be text or binary files.

   Phase 2 - Encyption: Use `openssl enc -aes-128-cbc -e` to encrypt these files using 128-bit AES with CBC mode. Please describe the size of the encrypted files.

   Phase 3 - Decryption: We would like to see what is added to the padding during the encryption. To achieve this goal, we will decrypt these files using `openssl enc -aes-128-cbc -d`. Unfortunately, decryption will automaticaly remove the padding by default, making it imposible for us to see the padding. However, the command does have an option called `-nopad`, which disables the padding, i.e., during the decryption, the command will not remove the padded data. Therefore, by looking at the decrypted data, we can see what data are used in the padding. Please use this technique to figure out what paddings are added to these files.

It should be noted that padding data may not be printable, so you need to use a hex tool to display the content. The following example shows how to display a file in hex format.

```
$ hexdump -C p1.txt
00000000  31 32 33 34 35 36 37 38  39 49 4a 4b 4c 0a   |123456789IJKL.|
$ xxd p1.txt
00000000: 3132 3334 3536 3738 3949 4a4b 4c0a            123456789IJKL.
```

4. Error Propagation

Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, or OFB, respectively?

To understand the error propagation property of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.

4. Decrypt the corrupted ciphertext file using the correct key and IV.

Find out whether your answer is correct or wrong after finishing this task. Please provide justification.

5. Programming using the Crypto Library (required)

Writing an encryption application (with C++ and Crypto++ library) to fulfill the following requirements:

1. Support AES algorithm.
2. Support: encrypt, decrypt data.
3. Mode of operations:
   - Support modes: ECB, CVC, OFB, CFB, CTR, XTS, CCM.
   - Select mode from the screen
4. Plaintext:
   - Case 1: Input from the screen
   - Case 2: From files (using filename)
   - *Support Vietnamese (using setmode, UTF-16) (bonus points)*
5. Ciphertext:
   - Case 1: Input from the screen in hex format
   - Case 2: From files (using filename)
6. Secret Key and Initialization Vector (IV)
   - Case 1: Input Secret Key and IV from the screen in hex format
   - Case 2: Input Secret Key and IV from files (using filename)
   - Case 3 (for encryption): Secret Key and IV are randomly chosen using random generator (using CryptoPP::AutoSeededRandomPool)

Generate a set of different input sizes (at least 3 inputs in size KBs up to GBs). Execute your code and check the computation time on average 100 running times. Sumarize the results in a table including: size of input, mode of operation, encryption time and decryption time.

C. REQUIREMENTS

You are expected to complete all tasks in section B (Lab tasks). Advanced tasks are optional, and you could get bonus points for completing those tasks. We prefer you work in a team of 2 members to get the highest efficiency.

Your submission must meet the following requirements:

- You need to submit a **detailed lab report in .docx** *(Word Document)* format, **using the report template** provided on the UIT Courses website.
- Either Vietnamese or English report is accepted. That's up to you. Using more than one language in the report is not allowed (except for the untranslatable keywords).
- When it comes to **programming tasks** *(require you to write an application or script)*, please attach all source-code and executable files (if any) in your submission. Please also list the important code snippets followed by explanations and screenshots when running your application. Simply attaching code without any explanation will not receive points.
- Submit work you are proud of – don't be sloppy and lazy!

  Your submissions must be your own. You are free to discuss with other classmates to find the solution. However, copying reports is prohibited, even if only a part of your report. Both reports of owner and copier will be rejected. Please remember to cite any source of the material (website, book,…) that influences your solution.

> **Notice:** Combine your lab report and all related files into a single **ZIP file (.zip)**, name it as follow:
>
> ***StudentID1_StudentID2_ReportLabX.zip***

D. REFERENCES

[1] William Stallings, *Cryptography and network security: Principles and practice, 7th ed*, Pearson Education, 2017. *Chapter 4, chapter 6, chapter 7*

[2] Wenliang Du (Syracuse University), *SEED Cryptography Labs*
https://seedsecuritylabs.org/Labs_20.04/Files/Crypto_Encryption

[3] Crypto++ Library
https://cryptopp.com/index.html

Attention: *Don't share any materials (slides, readings, assignments, labs, etc..) out of our class without my permission!*