

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH



MẬT MÃ VÀ AN NINH MẠNG (CO3069)

---

# Assignment 1

---

GVHD: Nguyễn Hữu Hiếu  
SV thực hiện: Vương Anh Khoa – 1711803

Tp. Hồ Chí Minh, Tháng 5/2020



## Mục lục

<b>1</b>	<b>Tóm tắt</b>	<b>2</b>
<b>2</b>	<b>Giới thiệu</b>	<b>2</b>
<b>3</b>	<b>Các thuật toán mã hóa khóa đối xứng khối</b>	<b>2</b>
3.1	Mã hóa khóa đối xứng . . . . .	2
3.2	Data Encryption Standard (DES) . . . . .	3
3.3	Triple DES (3DES) . . . . .	3
3.4	Advanced Encryption Standard (AES) . . . . .	4
<b>4</b>	<b>Các chế độ mã hóa khối</b>	<b>6</b>
4.1	Initialization vector (IV) . . . . .	7
4.2	Padding . . . . .	7
4.3	Electronic codebook (ECB) . . . . .	7
4.4	Cipher block chaining (CBC) . . . . .	8
4.5	Cipher feedback (CFB) . . . . .	9
4.6	Output feedback (OFB) . . . . .	10
<b>5</b>	<b>Xây dựng ứng dụng mã hóa tập tin</b>	<b>11</b>
5.1	Giao diện ứng dụng . . . . .	11
5.2	Quy trình mã hóa, giải mã và hash ở phía server . . . . .	14
5.2.1	Quy trình mã hóa . . . . .	14
5.2.2	Quy trình giải mã . . . . .	15
5.2.3	Quy trình hash . . . . .	16
<b>6</b>	<b>Đánh giá và kiểm chứng ứng dụng xây dựng</b>	<b>17</b>
6.1	Đánh giá tốc độ chạy của các giải thuật và chế độ . . . . .	17
6.2	Chế độ block cipher và stream cipher . . . . .	19
<b>7</b>	<b>Tổng kết và hướng phát triển</b>	<b>20</b>
<b>8</b>	<b>Tài liệu tham khảo</b>	<b>21</b>

## 1 Tóm tắt

Nội dung của bài báo cáo này sẽ gồm các phần như sau:

1. Tóm tắt các nội dung trong báo cáo.
2. Giới thiệu đề tài.
3. Cơ sở lý thuyết của các thuật toán mã hóa đối xứng khối.
4. Cơ sở lý thuyết về các chế độ mã hóa khối.
5. Xây dựng ứng dụng hiện thực.
6. Đánh giá và kiểm chứng ứng dụng hiện thực.
7. Hướng phát triển của ứng dụng trong tương lai.
8. Tài liệu tham khảo.

## 2 Giới thiệu

- Mã hóa là phương pháp giúp bảo vệ dữ liệu cá nhân nhạy cảm trên máy tính của bạn, cho dù bạn có gửi dữ liệu cho cá nhân, tổ chức nào đó qua mạng Internet, hay sao lưu dữ liệu cá nhân trên các máy chủ, Cloud, ..., thì việc mã hóa sẽ ngăn chặn bất cứ ai có thể đọc được dữ liệu trước khi được sự cho phép của bạn.
- Bài báo cáo sẽ phân tích kĩ về giải thuật mã hóa đối xứng sử dụng mã hóa dạng khối (symmetric key - block cipher). Cụ thể ở đây là giải thuật DES, 3DES và AES. Về chế độ mã hóa, bài viết sẽ nói về 4 chế độ mã hóa: Electronic codebook (ECB), Cipher block chaining (CBC), Cipher feedback (CFB) và Output feedback (OFB).
- Về hiện thực mã hóa và kiểm tra tính toàn vẹn dữ liệu của tập tin, phần giao diện cho người dùng sử dụng ReactJS. Việc xử lý mã hóa và kiểm tra tính toàn vẹn tập tin sẽ được thực hiện thông qua framework của NodeJS: NestJS và sử dụng thư viện crypto của Nodejs.

## 3 Các thuật toán mã hóa khóa đối xứng khối

### 3.1 Mã hóa khóa đối xứng

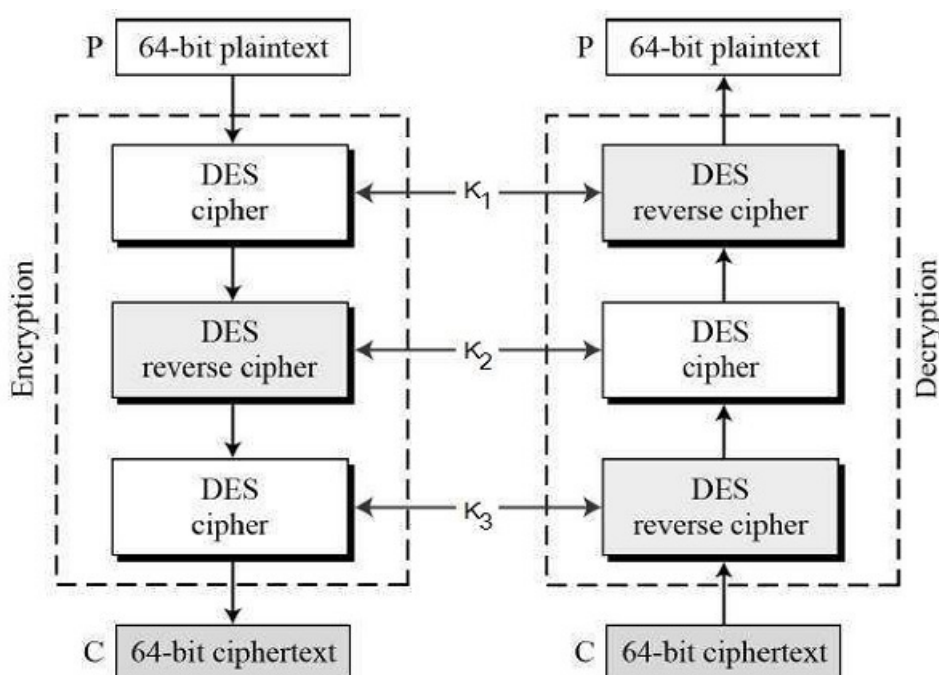
- Mật mã khóa đối xứng (hay mã hóa đối xứng) là một loại sơ đồ mã hóa trong đó một khóa giống nhau sẽ vừa được dùng để mã hóa, vừa được dùng để giải mã các tệp tin. Quá trình mã hóa bao gồm việc chạy văn bản thô (đầu vào) thông qua một thuật toán mã hóa còn gọi là mật mã (cipher) sẽ lần lượt tạo ra các bản mã - ciphertext (đầu ra). Quá trình giải mã về cơ bản sẽ chuyển đổi các bản mã trở về dạng văn bản thô ban đầu.
- Trong số các sơ đồ mã hóa đối xứng được sử dụng ngày nay thì có 2 loại thông dụng nhất là mã hóa dạng block và mã hóa dạng stream. Trong mã hóa dạng block, dữ liệu được nhóm vào từng khối theo kích thước định trước, mỗi khối được mã hóa bằng khóa đối xứng và thuật toán mã hóa. Mã hóa stream không mã hóa dữ liệu văn bản thô theo block mà mã hóa theo từng bit (mỗi văn bản thô 1-bit được mã hóa thành bản mã 1-bit mỗi lần).
- Thuật toán mã hóa dạng block được biết đến phổ biến hiện nay là DES và AES.

### 3.2 Data Encryption Standard (DES)

- Quá trình sinh khóa: Sử dụng một khóa  $K$  (64 bit, tuy vậy chỉ có 56 bit được dùng) tạo ra 16 khóa con 48 bit  $K_1, K_2, \dots, K_{16}$ .
- Quá trình mã hóa: Input đầu vào 64 bit, ta qua bảng hoán vị IP (Initial Permutation) ta được 1 khối cũng 64 bit. Ta tách ra làm 2 khối nhỏ 32 bit (Left 32 bit và Right 32 bit)  $L_0, R_0$ .
- Qua một hàm biến đổi, ta biến đổi  $L_0, R_0$  thành  $L_1, R_1$ . Tương tự thực hiện thêm 15 vòng nữa để đủ 16 vòng ta được  $L_{16}, R_{16}$ . Cộng chuỗi  $L_{16}$  với  $R_{16}$  với nhau, rồi qua bảng hoán vị IP-1 (Invert Initial Permutation) ta được bản mã Cipher Text.
- Quá trình giải mã chính là quá trình mã hóa. Nhưng dùng khóa con theo chiều ngược lại. Tức là khối đầu vào là bản mã hóa, dùng khóa con theo thứ tự ngược lại:  $K_{16}, K_{15}, \dots, K_1$ .

### 3.3 Triple DES (3DES)

- Là kĩ thuật cải tiến của mã hóa DES, với ý tưởng là dùng một key có kích thước lớn hơn để mã hóa plaintext 3 lần.



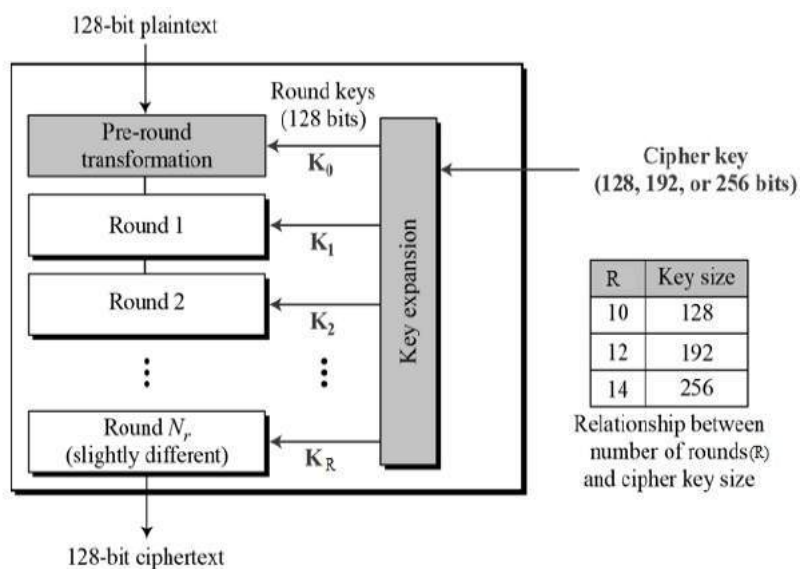
Hình 1: Sơ đồ mã hóa và giải mã 3DES

- Quá trình giải mã và mã hóa như sau:
  - Mã hóa plaintext sử dụng giải thuật DES thông thường với  $K_1$ .
  - Sử dụng giải thuật decrypt DES (reverse) đối với output ở bước trên sử dụng  $K_2$ .

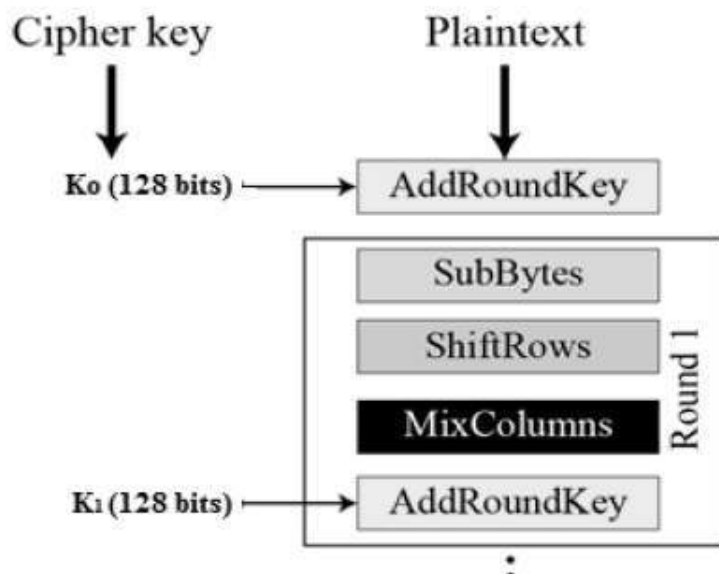
- Cuối cùng, mã hóa output của bước 2 với K3 sử dụng DES encryption.
- Output của bước 3 chính là ciphertext.
- Để giải mã ciphertext trên chỉ cần thực hiện các bước theo thứ tự ngược lại: decrypt K3, encrypt K2, decrypt K1.
- Dựa trên quá trình giải mã như trên mà 3DES còn có tên gọi khác là des-ede (encrypt - decrypt - encrypt).
- 3DES có 2 phiên bản: 3DES-2-keys và 3DES-3-keys. Khác nhau chỉ ở chỗ bước thực hiện encrypt DES cuối cùng thì thay vì 3DES-3-keys sẽ sử dụng khóa thứ 3 K3, thì 3DES-2-keys sẽ sử dụng lại khóa K1 để xử lí.

### 3.4 Advanced Encryption Standard (AES)

- AES được giới thiệu để thay thế cho DES (vì độ dài key khi sử dụng DES khá nhỏ và giải thuật chạy chậm hơn so với AES).
- Giải thuật AES có 3 biến thể: AES-128, AES-192, AES-256 tương ứng với chiều dài của khóa lần lượt là 128 bits, 192 bits và 256 bits (tương ứng với số vòng round khi chạy giải thuật là 10, 12, 14).
- AES nhận 128-bit dữ liệu đầu vào và key với độ dài thuộc vào 3 biến thể trên. AES sử dụng 128 bits chia thành 16 bytes và biểu diễn 16 bytes đó ở dạng ma trận 4x4 để xử lí.
- Ma trận 4x4 này sẽ được đưa vào bước initial transformation. Sau đó sẽ tiếp tục đem ma trận đã biến đổi đó xử lí qua 9, 11, 13 round tương ứng với AES-128, AES-192, AES-256, mỗi round có các giai đoạn từ trên xuống như sau:
  - **Subbytes:** Sử dụng S-box để thực hiện thay thế từng byte tạo thành ma trận.
  - **Shift rows:** Mỗi hàng của ma trận bị dịch chuyển sang trái (thực hiện phép dịch bit vòng, tùy mỗi hàng mà sự dịch chuyển khác đi).
  - **Mix columns:** Mỗi cột của ma trận được biến đổi dựa trên một hàm toán học. Output vẫn là một ma trận 4x4 nhưng giá trị khác đi. Bước này không được thực hiện ở round cuối cùng.
  - **Add round keys:** Thực hiện phép toán XOR của ma trận hiện tại với khóa mở rộng.
- Round cuối cùng chỉ xử lí qua các bước Subbytes, Shift Rows, và Add round keys. Output sẽ là 128 bits ciphertext.
- Việc giải mã ciphertext được thực hiện theo thứ tự ngược lại. Mỗi round cũng có 4 công đoạn nhưng được thực hiện theo thứ tự từ dưới lên.
- Do AES được tạo ra để thay thế DES nên về cả phương diện bảo mật lẫn tốc độ mã hóa, AES đều nhanh hơn DES (Kể cả 3DES).



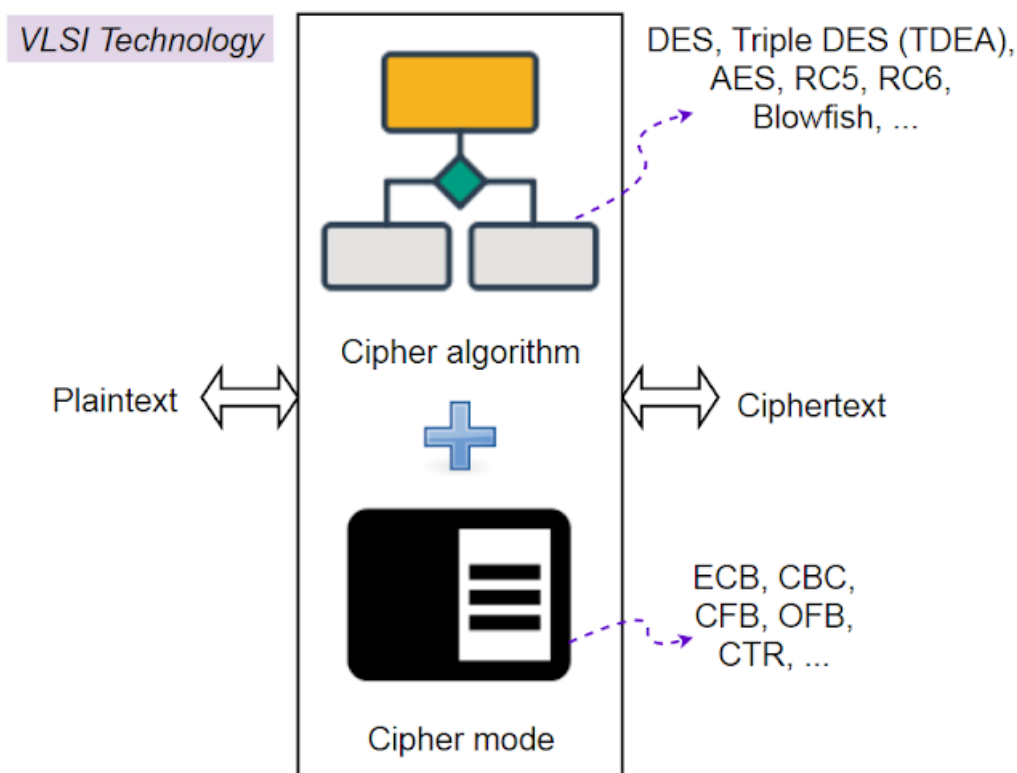
Hình 2: Cấu trúc giải mã AES



Hình 3: Pre-round transformation và qui trình xử lí của một round

## 4 Các chế độ mã hóa khối

Trong mật mã học, Chế độ mã hóa khối sử dụng các giải thuật mã hóa khối để mã hóa dữ liệu có kích thước khác với kích thước được quy định trong giải thuật mã hóa khối được chọn. Nó quy định cách áp dụng giải thuật mã hóa khối lên từng phần của dữ liệu theo một chế độ đã được chọn trước. Nói tóm lại, việc giải mã dữ liệu là sự kết hợp của chế độ mã hóa và thuật toán mã hóa.



**Hình 4:** Quá trình mã hóa là sự kết hợp giữa thuật toán mã hóa và chế độ mã hóa

#### 4.1 Initialization vector (IV)

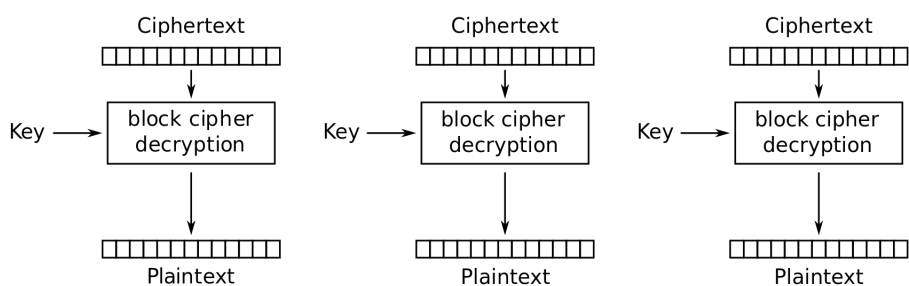
- Một vector khởi tạo (Initialization vector / IV) là một khối bits được sử dụng bởi nhiều chế độ mã hóa để ngẫu nhiên hóa quá trình mã hóa. Nó có công dụng là tạo ra 2 ciphertexts khác nhau kể cả khi plaintext giống nhau.
- IV không phải là một khóa, và vì vậy vấn đề bảo mật của IV khác với khóa của ciphertext. IV thường được tạo ra và lưu bên cạnh đoạn ciphertext trong cơ sở dữ liệu. Tuy vậy, có một điểm lưu ý là không được tái sử dụng một IV cho nhiều khóa. Việc tái sử dụng này sẽ dẫn đến việc làm rò rỉ thông tin của dữ liệu và tạo lỗ hổng cho hacker.

#### 4.2 Padding

- Là việc thêm một số bit vào một phần của dữ liệu để độ dài của dữ liệu là một bội số của plaintext được quy định tùy theo giải thuật mã hóa.
- Ví dụ nếu ta muốn mã hóa dữ liệu có độ dài 120 bits sử dụng thuật toán DES, nếu ta chọn chế độ mã hóa là ECB hoặc CBC thì ta cần thêm vào 8 bits ở cuối để được 128 bits là bội số của plaintext (64 bits) trong giải thuật DES. Chỉ khi đó ta mới có thể chạy giải thuật được.
- Luật padding có thể là thêm vào các bit 0 cho đến khi chiều dài thỏa mãn, hoặc thêm vào số lượng byte còn thiếu (Theo như ví dụ trên, do còn thiếu 8 bits tức thiếu 1 byte nên ta sẽ thêm giá trị 0000 0001 vào cuối của dữ liệu), etc.

#### 4.3 Electronic codebook (ECB)

- Là chế độ mã hóa đơn giản nhất, dữ liệu được cắt thành từng khúc với độ dài đúng như plaintext của thuật toán mã hóa khối được chọn. Ta mã hóa từng khúc đó và nối lại với nhau. Chế độ này có sử dụng padding và không sử dụng IV.
- Do không sử dụng IV nên nếu 2 dữ liệu giống nhau mã hóa sẽ cho ra 2 ciphertexts giống nhau.



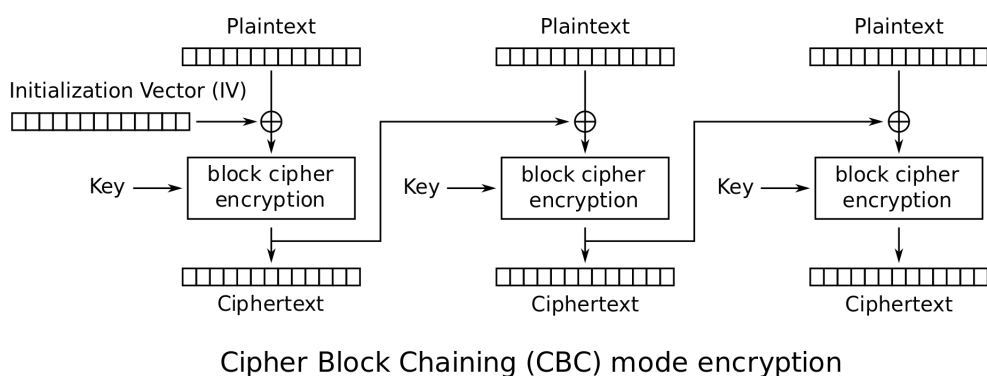
Electronic Codebook (ECB) mode decryption

Hình 5: Sơ đồ chế độ giải mã ECB

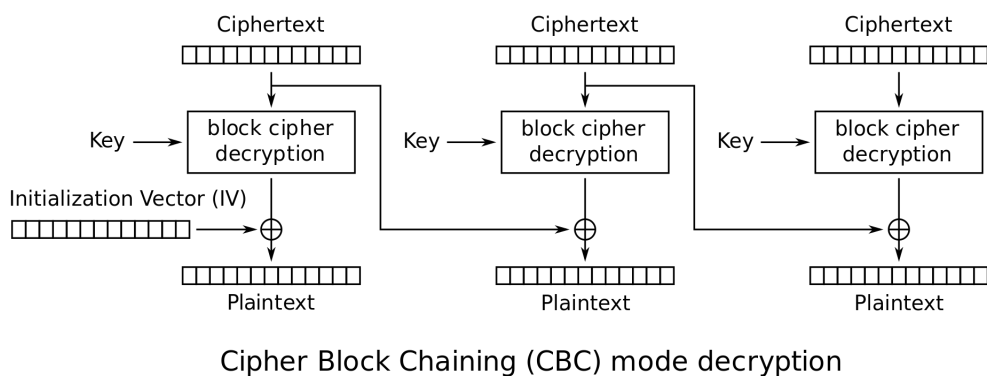


#### 4.4 Cipher block chaining (CBC)

- Ở mode này, ta không mã hóa riêng từng phân khúc của dữ liệu nữa mà thay vào đó lấy dữ liệu đã được mã hóa của khối 1 làm đầu vào plaintext cho khối 2, output của khối 2 làm input của khối 3, ... Chế độ này có sử dụng IV để mã hóa khối đầu tiên và có sử dụng padding.
- Do mã hóa dữ liệu theo từng khối một cách tuần tự chứ không tách biệt như ECB nên không thể thực hiện tính toán song song dẫn đến thời gian mã hóa so với chế độ ECB bị chậm hơn.
- Tuy vậy, việc giải mã vẫn có thể được thực hiện song song.



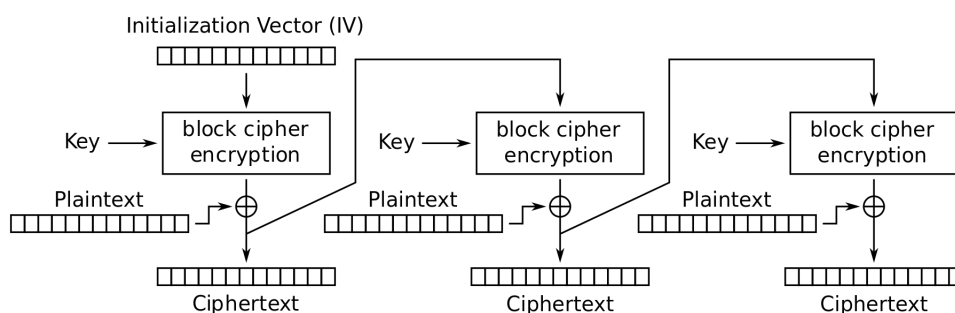
Hình 6: Sơ đồ chế độ mã hóa ECB



Hình 7: Sơ đồ chế độ giải mã ECB

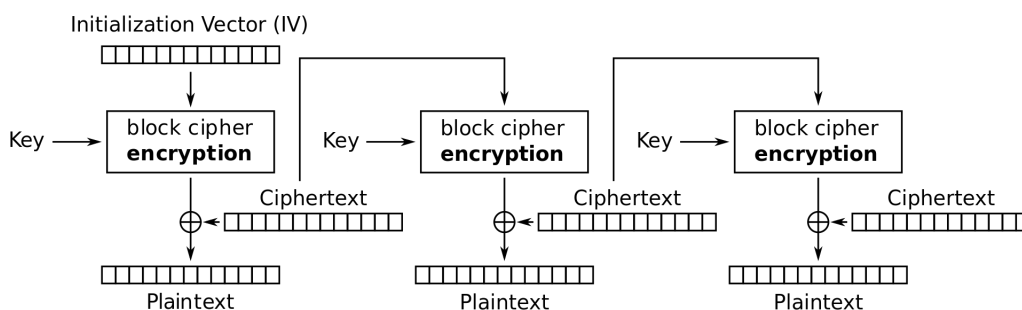
## 4.5 Cipher feedback (CFB)

- Chế độ này cũng tương tự như chế độ CBC, chỉ khác ở điểm thay vì plaintext và IV được XOR với nhau rồi chạy giải thuật mã hóa khối tương ứng với key thì CFB sẽ sử dụng IV và key để đưa vào giải thuật mã hóa khối và sau đó mới XOR với plaintext để ra ciphertext. Do IV luôn có độ dài bằng với plaintext được quy định trong thuật toán mã hóa khối được chọn nên ta không cần phải sử dụng kĩ thuật padding. Việc XOR từng bit của plaintext với đầu ra sau khi thực hiện mã hóa khối làm cho chế độ này tuy sử dụng giải thuật block cipher nhưng mã hóa ciphertext lại theo dạng stream cipher.



Cipher Feedback (CFB) mode encryption

Hình 8: Sơ đồ chế độ mã hóa CFB

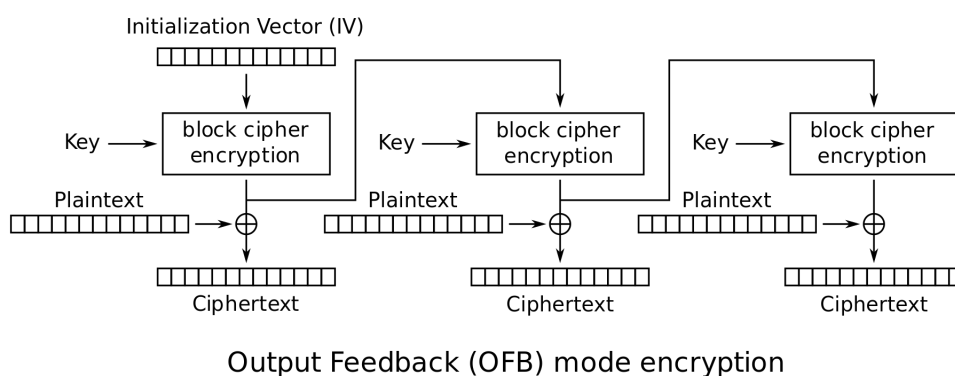


Cipher Feedback (CFB) mode decryption

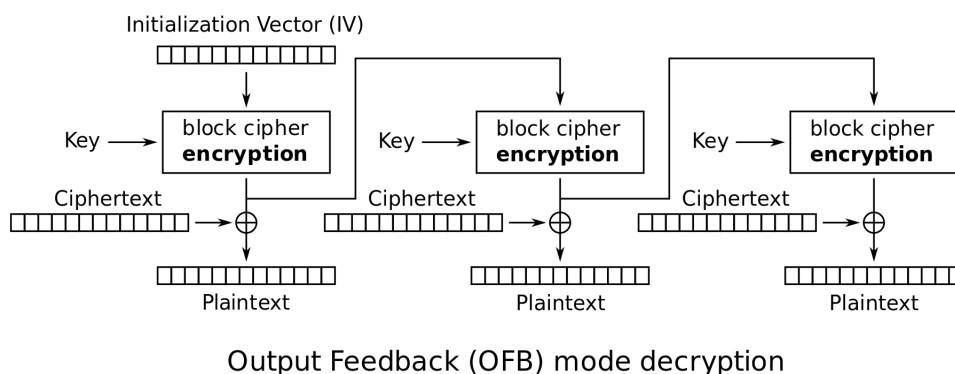
Hình 9: Sơ đồ chế độ giải mã CFB

## 4.6 Output feedback (OFB)

- Chế độ này cũng tương đồng với CFB. Điểm khác biệt là ta sẽ lấy output sau khi thực hiện mã hóa khối làm đầu vào cho việc mã hóa khối tiếp theo chứ không phải lấy giá trị đã được XOR với plaintext như ở CFB.
- Chế độ này cũng không cần phải padding và cũng biến việc mã hóa cục bộ là block cipher nhưng mã hóa toàn phần là stream cipher.



Hình 10: Sơ đồ chế độ mã hóa OFB

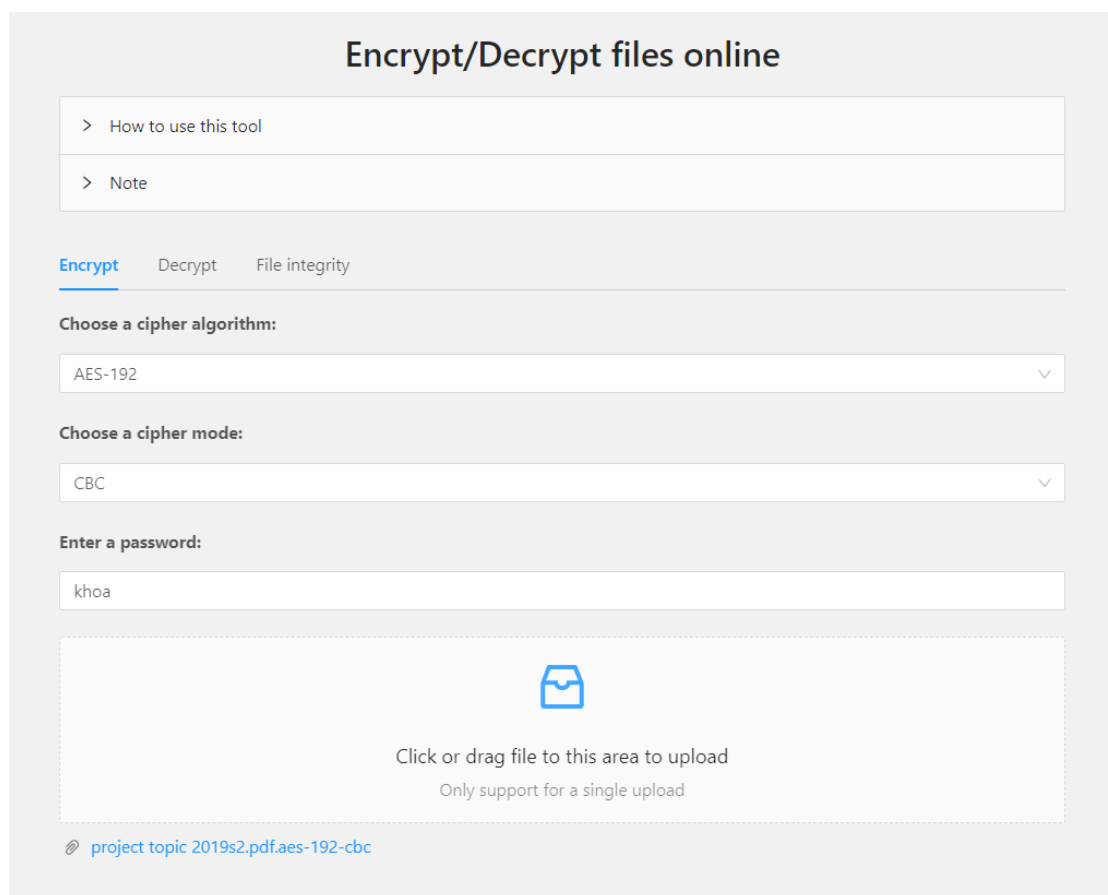


Hình 11: Sơ đồ chế độ giải mã OFB

## 5 Xây dựng ứng dụng mã hóa tập tin

### 5.1 Giao diện ứng dụng

- Ứng dụng được xây dựng dựa trên nền tảng web, giao diện sử dụng ReactJS
- Tab encrypt cho phép người dùng chọn thuật toán mã hóa (AES-128, AES-192, AES-256, DES, 3DES-2-keys, 3DES-3-keys), chế độ mã hóa (ECB, CBC, CFB, OFB, riêng thuật toán DES không hỗ trợ chế độ ECB), mật khẩu mã hóa file (Sẽ được hash lại thành key có độ dài tương ứng với giải thuật đã chọn ở phía server) và file muốn mã hóa. Sau khi chọn và điền đầy đủ các trường dữ liệu thì sẽ có một link download file được mã hóa xuất hiện. (Tên của file được mã hóa được format lại theo dạng <Tên file gốc + thuật toán + chế độ mã hóa>). Nếu người dùng không điền đầy đủ các trường dữ liệu thì sẽ có báo lỗi ở ngay chính vị trí link download.



Hình 12: Tab encrypt

- Tab decrypt cho phép người dùng điền và chọn các trường dữ liệu giống như tab encrypt. Tuy vậy để nhận lại được file gốc chính xác thì ta phải điền các trường giống như ở tab encrypt và thả file được mã hóa vào khung nhập file. Khi đó ta sẽ nhận được một link download file có tên giống với file gốc được mã hóa. Nếu nhập thuật toán, chế độ mã hóa hoặc mật khẩu mã hóa sai thì ta sẽ nhận được thông báo lỗi cũng ở vị trí link download. Tuy vậy cũng có trường hợp tuy các trường không khớp với tab encrypt nhưng vẫn nhận được file download. Khi đó dù tải về được nhưng nội dung của file chắc chắn sẽ khác so với file gốc.

### Encrypt/Decrypt files online

> How to use this tool

> Note

Encrypt

Decrypt

File integrity

Choose a cipher algorithm:

AES-192

Choose a cipher mode:


CBC

Enter a password:

khoa

Click or drag file to this area to upload

Only support for a single upload

 [project topic 2019s2.pdf](#)

**Hình 13:** Tab decrypt


- Cuối cùng, tab File integrity dùng để kiểm tra xem 2 file có giống nhau hay không. Luồng thực thi là thả 2 file vào 2 ô nhập file, nội dung hash của 2 file sẽ được hiển thị ở 2 ô phía dưới. Nếu nội dung hash giống nhau tức là nội dung của 2 file được so sánh là giống nhau và ngược lại. (Không quan tâm đến tên file).

Encrypt


Decrypt

File integrity


File #1:




Click or drag file to this area to upload  
Only support for a single upload

 project topic 2019s2.pdf


File #2:



Click or drag file to this area to upload  
Only support for a single upload

 project topic 2019s2.pdf

Compare hashed value



Equal

File 1:

1284798fe03c24f8c5df32ed233f3c158034d659b006280bb9  
ffb049e57bb89d

File 2:

1284798fe03c24f8c5df32ed233f3c158034d659b006280bb9  
ffb049e57bb89d

**Hình 14:** Tab File integrity

## 5.2 Quy trình mã hóa, giải mã và hash ở phía server

- Các trường thông tin cần thiết và file từ phía client sẽ được gửi về phía server. Server sử dụng framework của NodeJS: NestJS để hiện thực. Việc mã hóa cũng như tạo hash của các file được thực hiện thông qua thư viện crypto của NodeJS.
- Tập tin được truyền vào sẽ sử dụng thư viện fs để đọc tập tin dưới dạng cấu trúc dữ liệu Buffer (Chuỗi bit được biểu diễn dưới dạng hex). Sau đó Buffer của tập tin đó sẽ được đưa vào quá trình giải mã, mã hóa hoặc hash tùy theo cách sử dụng.
- Chương trình có thể mã hóa, giải mã và hash file với đuôi bất kì và độ lớn bất kì.
- Lưu ý rằng khi người dùng click vào link download, server sẽ thực hiện xóa file đó ở trên phía server. Việc này nhằm mục đích giảm tải trọng lưu trữ file ở server cũng như chỉ cho phép một file mã hóa hoặc giải mã được truy cập một lần duy nhất.

### 5.2.1 Quy trình mã hóa

#### 5.2.1.1 Sinh khóa từ password client gửi đến và sinh IV ngẫu nhiên

- Đối với AES-128, 192, 256 thì độ dài của key lần lượt là 16, 24, 32 bytes. DES sẽ là 8 bytes, 3DES-2-keys là 16 bytes, 3DES-2-keys là 24 bytes. IV đối với giải thuật DES sẽ là 16 bytes, còn với DES và 3DES sẽ là 8 bytes (tương ứng với độ dài ciphertext).

```
let keyLength, iv;
if (algorithm.slice(0, 3) == 'aes') {
  keyLength = parseInt(algorithm.slice(-3)) / 8;
  iv = randomBytes(16);
} else {
  // DES
  if (algorithm.slice(4, 8) == 'ede3') keyLength = 24;
  else if (algorithm.slice(4, 7) == 'ede') keyLength = 16;
  else keyLength = 8;
  iv = randomBytes(8);
}

const key = scryptSync(password, 'salt', keyLength);
```

Hình 15: Xác định độ dài khóa và IV

- Cuối cùng, ta tạo key bằng cách dùng hàm scryptSync của thư viện NodeJS. Đây là một hàm băm có tác dụng tạo key từ một chuỗi string (Trong trường hợp này là password bên phía người dùng gửi đến) và có tham số để xác định độ dài của output (Ta truyền keyLength đã tính toán vào hàm).

#### 5.2.1.2 Tạo cipher (algorithm + mode)

- Ngoại trừ chế độ ECB thì các chế độ còn lại đều cần IV

```
cipher =  
  mode == 'ecb'  
    ? createCipheriv(`${algorithm}-${mode}`, key, null)  
    : createCipheriv(`${algorithm}-${mode}`, key, iv);
```

Hình 16: Tạo cipher dựa trên hàm có sẵn của thư viện crypto

#### 5.2.1.3 Tạo ciphertext

- Do để chuẩn bị cho quá trình giải mã, IV phải được gửi kèm với ciphertext nên để thuận tiện, ta sẽ concat chuỗi IV vào đầu chuỗi ciphertext để khi gửi ciphertext vào phần giải mã sẽ có cùng IV.

```
let encryptedBuffer = Buffer.concat([  
  cipher.update(buffer),  
  cipher.final(),  
]);  
encryptedBuffer = Buffer.concat([iv, encryptedBuffer]);  
  
return encryptedBuffer;
```

Hình 17: Tạo ciphertext kèm theo IV

#### 5.2.2 Quy trình giải mã

- Việc xác định độ dài khóa và sinh khóa là tương tự như ở phần mã hóa.
- Do khi ta mã hóa, ta có thêm chuỗi IV vào đầu buffer nên ở phần giải mã, ta sẽ lấy chuỗi IV đó ra bằng cách cắt ciphertext được truyền vào một khúc ở đầu.
- Việc tạo decipher tương tự như phần tạo cipher ở mã hóa, chỉ khác là sử dụng hàm "createDecipheriv". Các tham số truyền vào giống hệt như của hàm "createCipheriv".
- Giai đoạn giải mã ngược ciphertext lại thành plaintext: ta lấy ra ciphertext ở khúc sau của buffer (Sau khi đã cắt lấy đoạn IV đầu) để tiến hành giải mã



```
const decryptedBuffer = Buffer.concat([
  decipher.update(buffer.slice(iv.length)),
  decipher.final(),
]);

return decryptedBuffer;
```

Hình 18: Giải mã ciphertext từ buffer

### 5.2.3 Quy trình hash

- Ta sử dụng hàm băm sha256 của thư viện crypto một cách đơn giản như sau:

```
const hash = createHash('sha256');
hash.update(buffer);
return hash.digest('hex');
```

Hình 19: Tạo ra chuỗi băm từ buffer được truyền vào bằng giải thuật sha256

## 6 Đánh giá và kiểm chứng ứng dụng xây dựng

### 6.1 Đánh giá tốc độ chạy của các giải thuật và chế độ

- Để đánh giá tốc độ chạy của việc giải mã và mã hóa, ta sẽ thêm đoạn lệnh đo thời gian kẹp giữa hàm mã hóa và giải mã.

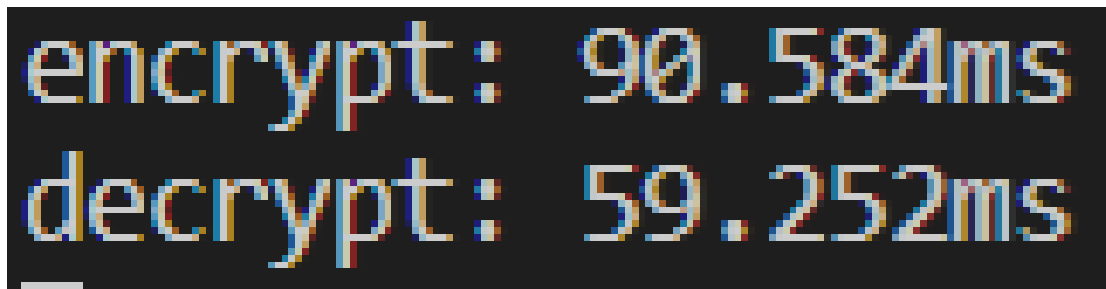
```
console.time('encrypt');  
const encrypted = this.fileService.encrypt(  
  dataBuffer,  
  algorithm,  
  password,  
  mode,  
);  
console.timeEnd('encrypt');
```

Hình 20: Đoạn mã đo thời gian qui trình mã hóa

```
console.time('decrypt');  
const decrypted = this.fileService.decrypt(  
  dataBuffer,  
  algorithm,  
  password,  
  mode,  
);  
console.timeEnd('decrypt');
```

Hình 21: Đoạn mã đo thời gian qui trình giải mã

- Output sẽ có dạng như sau:



```
encrypt: 90.584ms
decrypt: 59.252ms
```

Hình 22: Output format

- Ta sẽ thực hiện đánh giá thời gian mã hóa và giải mã trên một file có kích thước 96616 KB (Xấp xỉ 9.6MB), chọn chế độ CBC để đánh giá các giải thuật. Kết quả thu được trung bình sau 10 lần chạy mỗi giải thuật như sau:

	AES-128	AES-192	AES-256	DES	3DES-2-keys	3DES-3-keys
Encrypt	318.502	346.302	379.053	1677.099	4167.74	4173.5
Decrypt	167.55	187.827	199.665	1648.365	4127.275	4018.542
Đơn vị: ms						

Hình 23: Thời gian chạy các giải thuật mã hóa và giải mã theo chế độ CBC trên tập tin 9.6MB

- Dựa trên bảng kết quả trên, ta có thể nhận xét như sau:
  - Thời gian chạy giải thuật AES-128 < AES-192 < AES-256 (Tuy không đáng kể). Lí do là AES-128 cần chạy 10 vòng lặp, AES-192 chạy 12 vòng lặp, trong khi đó AES-256 chạy 14 vòng lặp nên thời gian chậm hơn.
  - Giải thuật DES chạy chậm hơn đáng kể so với giải thuật AES. giải thuật 3DES có thời gian chạy gấp khoảng 2.5 lần giải thuật DES thông thường. Giải thuật 3DES-2-keys và 3DES-3-keys có thời gian chạy gần như nhau.
  - Có thể thêm nhận xét thêm về chế độ CBC: Trong giải thuật AES chạy CBC, ta có thể thấy thời gian encrypt > decrypt. Lí do là như phân tích ở phần lí thuyết, khi mã hóa AES chạy chế độ CBC thì ta phải mã hóa tuần tự các khối, còn khi giải mã thì ta có thể thực hiện tính toán song song để giải mã các khối nên tốc độ nhanh hơn.
- Tiếp theo, ta sẽ đánh giá về thời gian chạy của các chế độ cipher. Ta sẽ chọn giải thuật AES-128 và cũng dùng tập tin có kích thước 96616 KB để đánh giá. Kết quả thu được trung bình sau 10 lần chạy mỗi chế độ như sau:

	ECB	CBC	CFB	OFB
Encrypt	241.24	321.94	355.418	317.556
Decrypt	179.711	172.318	299.326	263.58
Đơn vị: ms				

**Hình 24:** Thời gian chạy các chế độ mã hóa và giải mã theo giải thuật AES-128 trên tập tin 9.6MB

- Dựa trên bảng kết quả trên, ta có thể nhận xét: Về mã hóa, do chế độ ECB cho phép thực hiện mã hóa song song nên thời gian chạy chế độ ECB nhanh hơn các chế độ chạy tuần tự còn lại.

## 6.2 Chế độ block cipher và stream cipher

- Như đã nói ở phần cơ sở lý thuyết, 2 chế độ ECB và CBC có sử dụng padding để biến buffer dữ liệu thành ciphertext có độ dài là bội số của ciphertext mà giải thuật được chọn qui định.
- Ta sẽ chọn một file text đơn giản có nội dung "toidihoc" làm đầu vào để quan sát buffer trước và sau khi mã hóa (Không xét ciphertext sau khi đã concat IV vào đầu). Ta chọn AES-128-ECB để mã hóa file, thu được kết quả như sau:

```
plaintext:  8 <Buffer 74 6f 69 64 69 68 6f 63>
ciphertext: 16 <Buffer 40 f1 f6 c1 6e dc 93 31 ea 70 df 86 8f 93 12 51>
```

**Hình 25:** Buffer của plaintext và ciphertext sử dụng ECB

- Chương trình đã xử lý như sau: Ta sẽ padding thêm vào plaintext cho đủ 16 bytes (128 bits). Luật padding của thư viện crypto sử dụng PKCS7, tức chuỗi plaintext sẽ trở thành: 74 6f 69 64 69 68 6f 63 08 08 08 08 08 08 08 08. Sau khi được padding thì chuỗi plaintext sẽ được đưa vào khối xử lý thuật toán và đưa ra kết quả là ciphertext như hình.
- Chế độ ECB cũng sử dụng luật padding tương tự như vậy. Khối ciphertext chắc chắn sẽ luôn là bội số của chiều dài ciphertext qui định trong giải thuật (AES là bội số của 128 bits, DES là bội số của 64 bits).
- Tuy vậy, nếu sử dụng chế độ mã hóa stream cipher như CFB (OFB tương tự) thì ta sẽ cho ra ciphertext với độ dài bằng với độ dài của plaintext.

```
plaintext:  8 <Buffer 74 6f 69 64 69 68 6f 63>
ciphertext:  8 <Buffer 2f bb c8 96 63 5a 7a 65>
```

**Hình 26:** Buffer của plaintext và ciphertext sử dụng CFB

## 7 Tổng kết và hướng phát triển

- Bài báo cáo đã giới thiệu các giải thuật mã hóa đối xứng khối cũng như các chế độ mã hóa đi kèm. Đồng thời cũng hiện thực một chương trình cho phép mã hóa, giải mã và kiểm chứng tính toàn vẹn của tập tin sử dụng lý thuyết đã phân tích nêu trên (Bao gồm cả kiểm tra việc nhập dữ liệu đầy đủ của người dùng và xuất ra thông báo lỗi nếu người dùng điền thiếu các trường dữ liệu cũng như nhập sai mật khẩu, chế độ hoặc thuật toán ở phần giải mã).
- Tuy vậy, nội dung bài báo cáo vẫn chưa phân tích được tính bảo mật khi so sánh các thuật toán với nhau (Chỉ phân tích so sánh thời gian chạy của các giải thuật và chế độ). Bài báo cáo chỉ đưa ra thông tin AES do được sáng tạo nhằm mục đích thay thế DES nên về phương diện bảo mật AES trội hơn. Người đọc có thể tự nghiên cứu về tính bảo mật của các giải thuật qua các bài viết trên mạng. Bài báo cáo cũng chưa phân tích chi tiết về các chế độ mã hóa (Như lợi ích khi so sánh các chế độ mã hóa với nhau, khi nào nên dùng chế độ nào vv...).
- Hướng phát triển của ứng dụng: Thực chất, IV không nên được lưu cạnh ciphertext trên cơ sở dữ liệu chứ không phải là lưu bên trong ciphertext như ứng dụng đã hiện thực (Nhằm mục đích đơn giản hóa). Trong tương lai, ứng dụng có thể được cải tiến bằng cách thêm các giải thuật khác như blowfish, RSA, etc và các chế độ mã hóa khác như CTR, GCM, etc. Đồng thời tích hợp ứng dụng vào các ứng dụng khác như ứng dụng chat, file sharing, mã hóa cả một thư mục, etc.

## 8 Tài liệu tham khảo

### Tài liệu

- [0] [https://en.wikipedia.org/wiki/Block\\_cipher\\_mode\\_of\\_operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)
- [1] <https://stackoverflow.com/questions/50701311/node-js-crypto-whats-the-default-padding-for-aes>
- [2] <http://nguyenquanicd.blogspot.com/2019/10/aes-bai-6-cac-che-o-ma-hoa-va-giai-ma.html>
- [3] [https://www.tutorialspoint.com/cryptography/triple\\_des.htm](https://www.tutorialspoint.com/cryptography/triple_des.htm)
- [4] <https://nodejs.org/api/crypto.html>
- [5] <https://nestjs.com/>
- [6] [https://www.tutorialspoint.com/cryptography/advanced\\_encryption\\_standard.htm](https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm)
- [7] Slide bài giảng Mật Mã và An Ninh Mạng chương Mã hóa khóa đối xứng