BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM
KHOA ĐÀO TẠO CHẤT LƯỢNG CAO

Phân biệt các mode mã hóa khối (CBC, ECB, CFB, OFB)

SINH VIÊN THỰC HIỆN:

Hoàng Nguyễn Kim Duy 19110178
Đặng Lê Quang 19110270

GIẢNG VIÊN HƯỚNG DẪN:

Ths. Nguyễn Thị Thanh Vân

Tp. Hồ Chí Minh, tháng 12 năm 2021

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Ngày…….Tháng……Năm……

Giáo Viên Hướng Dẫn

(Ký và ghi rõ họ tên)

LỜI CẢM ƠN

Trong quá trình học tập, nghiên cứu đề tài *"**Phân biệt các mode mã hóa khối",*** em đã nhận được sự giúp đỡ, chỉ bảo nhiệt tình của các thầy, cô giáo trường Trường Đại học Sư phạm Kỹ thuật TP.HCM để hoàn thành bài luận này. Với tình cảm chân thành, em bày tỏ lòng biết ơn đối với Ban giám hiệu, Khoa Khoa Đào tạo chất lượng cao – Trường Đại học Sư phạm Kỹ thuật TP.HCM, các thầy giáo, cô giáo đã tham gia quản lý, giảng dạy và giúp đỡ em trong suốt quá trình học tập, nghiên cứu.

Em xin bày tỏ sự biết ơn đặc biệt đến cô Nguyễn Thị Thanh Vân - người đã trực tiếp hướng dẫn, giúp đỡ về khoa học để nhóm hoàn thành tốt bài luận này. Với lòng biết ơn sâu sắc nhất, em xin gửi đến cô. Chúc cô sẽ luôn thành công và tâm huyết hơn với nghề để tiếp tục truyền đạt những kiến thức quý báu cho chúng em trong quá trình học tập ở trường và xa hơn là ngoài cuộc sống.

Mặc dù đã có nhiều cố gắng trong suốt quá trình thực hiện đề tài, song có thể còn có những mặt hạn chế, thiếu sót. Em rất mong nhận được ý kiến đóng góp và sự chỉ dẫn của các thầy cô giáo và bạn bè.

Em xin chân thành cảm ơn!

MỤC LỤC

I. Tổng quan

I.1 Giới thiệu

Các thuật toán mã hóa như mã hóa khối (block cipher algorithm) cung cấp cơ chế chuyển đổi thuận nghịch giữa giá trị thực được dùng bởi hệ thống, gọi là bản rõ (plaintext), và giá trị mã hóa, gọi là bản mã (ciphertext).

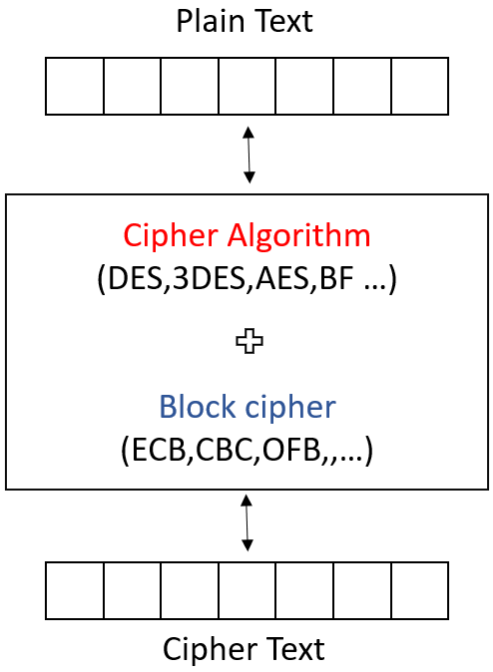
Mã hóa khối (block cipher) là mã hóa thực thi trên từng khối bit có kích thước cố định. Ví dụ, DES mã hóa trên 64 bit dữ liệu đầu vào, AES mã hóa trên 128, 192 hoặc 256 bit dữ liệu đầu vào. Mã hóa khối khác với mã hóa dòng (stream cipher), hay còn gọi là mã hóa luồng. Mã hóa dòng xử lý trên từng bit dữ liệu đầu vào.

Mã hóa khóa mã đối xứng (symmetric key) là thuật toán dùng khóa mã hóa và khóa giải mã có mối liên hệ với nhau, có thể dùng khóa này biến đổi thành khóa kia và ngược lại, hoặc hai khóa mã hoàn toàn giống nhau.

Trong khi vai trò chính của một thuật toán mã hóa là sử dụng khóa mã bảo mật (secure/secret key) kết hợp với bản rõ để tạo ra bản mã, hoặc ngược lại, kết hợp với bản mã để khôi phục lại bản rõ thì vai trò của "chế độ mã hóa" là quy định cách thức sử dụng của một thuật toán mã hóa. Cùng một thuật toán mã hóa, cách thức sử dụng khác nhau sẽ tạo ra các kết quả mã hóa dữ liệu khác nhau với mức độ bảo mật khác nhau.

Tóm lại, quá trình mã hóa dữ liệu gồm 2 phần quan trọng là:

- Thuật toán mã hóa (cipher algorithm)
- Chế độ mã hóa (cipher mode)



Hình 1: Quá trình mã hóa là sự kết hợp giữa thuật toán mã hóa và chế độ mã hóa

I.2 Một vài chú thích

Sau đây nhóm em sẽ cung cấp một vài các ký hiệu thường dùng khi thực hiện viết bài báo cáo, được sử dụng cho việc thể hiện các biểu thức định nghĩa quá trình mã hóa và giải mã.

2.1 Các ký hiệu

- *Khối* hay **block**, nó chỉ một nhóm bit có số lượng là b
- *Đoạn hoặc phân đoạn* hay **segment**, nó chỉ một nhóm bit có số lượng là s

Ký hiệu	Ý nghĩa
b	Kích thước theo bit của một khối (**block**)
j	Chỉ số của các khối hoặc đoạn (**segment**) dữ liệu theo thứ tự từ trái sang phải
n	Số block hoặc segment dữ liệu trong plaintext
s	Số bit của một segment dữ liệu
u	Số bit trong block plaintext hoặc ciphertext cuối cùng
$\mathbf{C_j}$	Block ciphertext thứ **j**
$\mathbf{C^{\#}_j}$	Segment ciphertext thứ **j**
$\mathbf{C^{*}_n}$	Block cuối cùng của ciphertext, nó có thể là khối chỉ chứa một phần dữ liệu (partial block)
$\mathbf{I_j}$	Block ngõ vào thứ **j**
IV	Vector khởi tạo (Initialization Vector)
K	Khóa mã
$\mathbf{O_j}$	Khối ngõ ra thứ **j**
$\mathbf{P_j}$	Khối plaintext thứ **j**
$\mathbf{P^{\#}_j}$	Đoạn plaintext thứ **j**
$\mathbf{P^{*}_n}$	Khối cuối cùng của plaintext, nó có thể là khối chỉ chứa một phần dữ liệu.

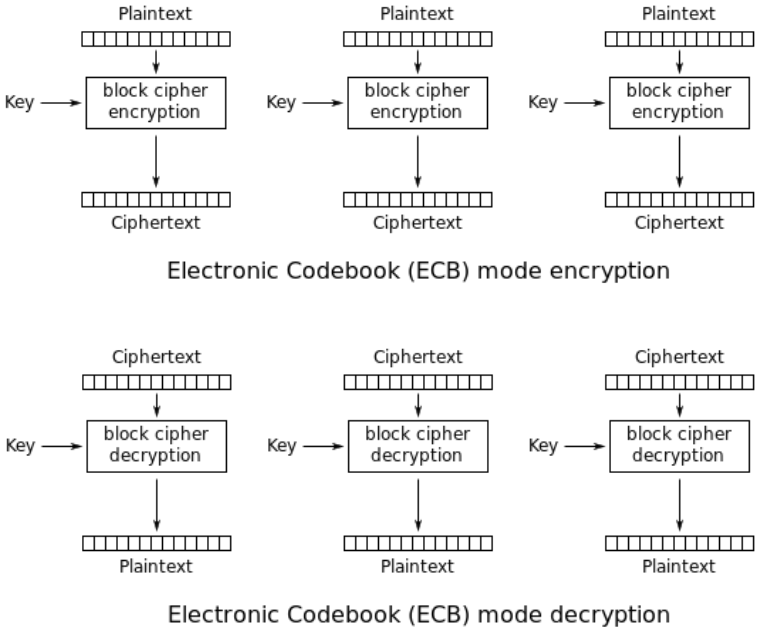
2.2 Các chức năng

Ký hiệu	Chức năng
X | Y	Nối hai chuỗi bit X và Y
$\mathbf{X \oplus Y}$	XOR hai chuỗi bit có cùng độ dài X và Y
$\mathbf{CIPH_k(X)}$	Chức năng mã hóa (mã hóa thuận) của một thuật toán mã hóa khối với khóa mã K áp dụng trên khối dữ liệu X.
$\mathbf{CIPH^{-1}_k(X)}$	Chức năng giải mã (mã hóa nghịch) của một thuật toán mã hóa khối với khóa mã K áp dụng trên khối dữ liệu X.
$\mathbf{LSB_m(X)}$	Chức năng lấy m bit LSB của chuỗi bit X
$\mathbf{MSB_m(X)}$	Chức năng lấy m bit MSB của chuỗi bit X

II. Giới thiệu về các chế độ mã hoá khối

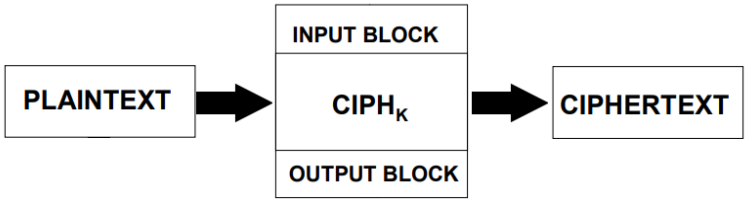
II.1 ECB (Electronic Codebook)

ECB là chế độ mã hóa từng khối bit độc lập. Với cùng một khóa K, mỗi khối plaintext ứng với một khối ciphertext cố định và ngược lại.



Hình 2: Chế độ mã hóa/giải mã ECB

1.1 Quá trình mã hóa ECB



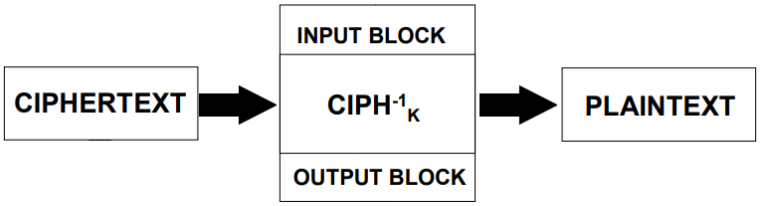
Hình 3: Chế độ mã hóa ECB

Biểu thức định nghĩa:

$$P_j = CIPH_k(C_j) \text{ với } j=1, 2, 3, \ldots, n$$

Plaintext là đầu vào trực tiếp để thực thi thuật toán mã hóa với khóa K để tạo ra ciphertext.

1.2 Quá trình giải mã ECB



Hình 4: Chế độ giải mã ECB

Biểu thức định nghĩa:

$$P_j = CIPH^{-1}_k(C_j) \text{ với } j=1, 2, 3, \ldots, n$$

Ciphertext là đầu vào trực tiếp để thực thi thuật toán giải mã với khóa K để tạo ra plaintext.

1.3 Ưu điểm và nhược điểm

1.3.1 Ưu điểm

- Thiết kế phần cứng đơn giản. Vấn đề chính là thiết kế logic cho thuật toán mã hóa.
- Không bị lỗi bị lan truyền. Nếu lỗi bit xuất hiện trên một ciphertext của một khối dữ liệu thì nó chỉ ảnh hưởng đến việc giải mã khối dữ liệu đó chứ không ảnh hưởng đến việc giải mã khác khối dữ liệu khác.
- Có thể thực hiện mã hóa/giải mã song song (parallel) nhiều khối dữ liệu cùng lúc. Điều này giúp tăng tốc độ xử lý trong các hệ thống đòi hỏi mã hóa/giải mã tốc độ cao.

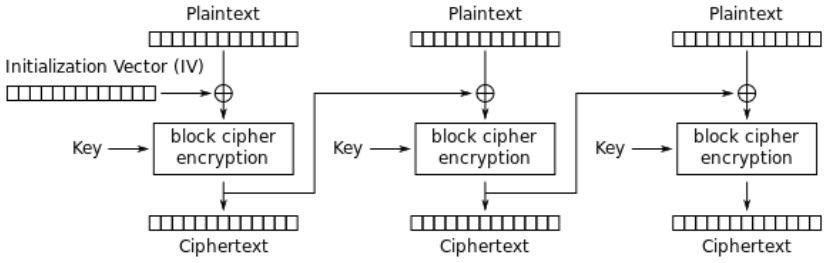
1.3.2 Nhược điểm

Khả năng bảo mật kém.

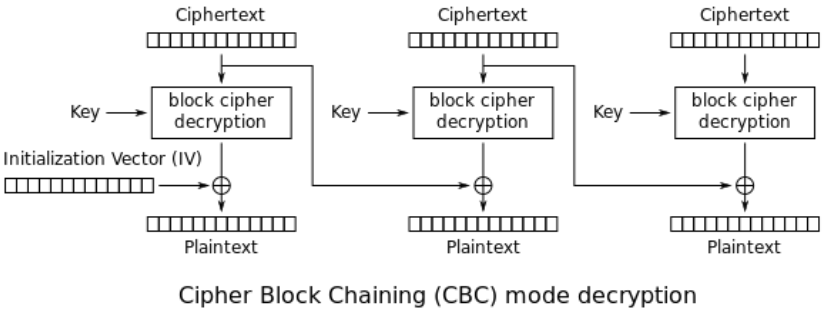
Do plaintext và ciphertext được ánh xạ một cách độc lập một-một nên thông tin mã hóa dễ bị sửa đổi bằng cách như xóa bớt khối dữ liệu, chèn thêm khối dữ liệu, hoán đổi vị trí khối dữ liệu để làm sai lệch thông tin tại nơi nhận.

II.2 CBC (Cipher Block Chaining)

CBC là chế độ mã hóa chuỗi, kết quả mã hóa của khối dữ liệu trước (ciphertext) sẽ được tổ hợp với khối dữ liệu kế tiếp (plaintext) trước khi thực thi mã hóa.

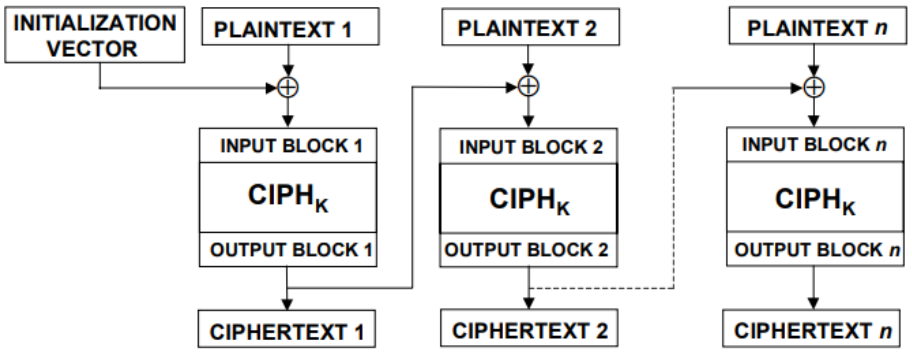


Cipher Block Chaining (CBC) mode encryption



Hình 3: Chế độ mã hóa/giải mã CBC

2.1 Quá trình mã hóa CBC



Hình 4: Quá trình mã hóa CBC

Biểu thức định nghĩa:

$$C_1 = CIPH_K\ (P_1 \oplus IV)$$
$$C_j = CIPH_K\ (P_j \oplus C_{j-1}) \qquad \text{với } j=2, 3, \ldots, n$$

(1) Lần mã hóa đầu tiên:

- Thực hiện XOR plaintext với vector khởi tạo ***IV***
- Kết quả phép toán trên là đầu vào cho việc thực thi thuật toán mã hóa với khóa **K**

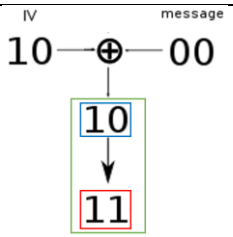
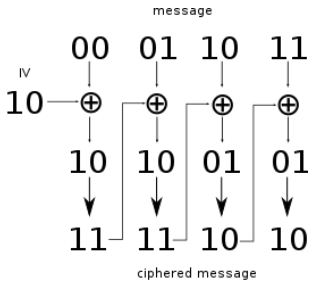
(n) Lần mã hóa sau lần đầu tiên:

- Thực hiện XOR plaintext với ciphertext của lần mã hóa trước đó.
- Kết quả phép toán trên là đầu vào cho việc thực thi thuật toán mã hóa với khóa **K**

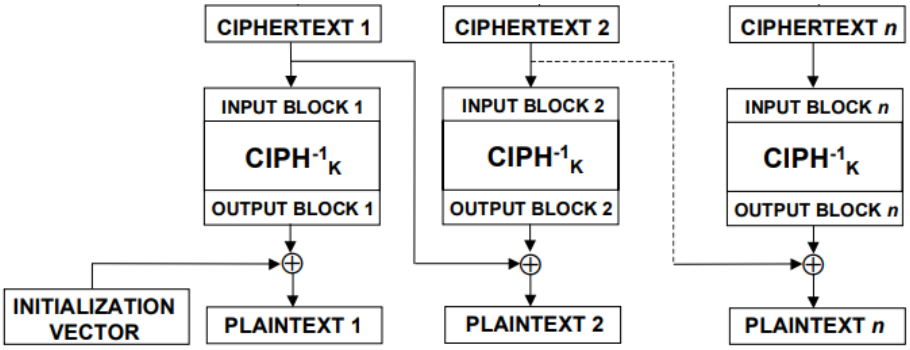
Ví dụ cho quá trình mã hoá, ta có các thành phần ban đầu như sau:

$$\begin{cases} \text{Message} = 00011011 \\ \text{IV} = 10 \end{cases}$$

Đầu tiên ta thực hiện chia message ra từng block, trong trường hợp này Size của block = 4 bit	message 00 01 10 11

Thực hiện lần mã hoá đầu tiên (1) cho block đầu tiên * Chú thích: Viền lam: IV Block message đầu tiên Viền đỏ: Block Cipher text đầu tiên Viền lục: Hàm Encrypt	
Tiếp tục thực hiện (n) cho toàn bộ block message tiếp. Kết quả cuối cùng ta nhận được: *Ciphertext = 11111010*	

2.2 Quá trình giải mã CBC



Hình 5: Quá trình giải mã CBC

Biểu thức định nghĩa:

$$P_1 = CIPH^{-1}_K\ (C_1 \oplus IV)$$
$$P_j = CIPH^{-1}_K\ (C_j) \oplus C_{j-1} \qquad \text{với } j=2, 3, \ldots, n$$

(1) Lần giải mã đầu tiên:

- Ciphertext được thực thi quá trình giải mã với khóa mã K
- Thực hiện XOR kết quả trên với vector khởi tạo *IV* để tạo ra plaintext

(n) Lần giải mã sau lần đầu tiên:

- Ciphertext được thực thi quá trình giải mã với khóa mã K
- Thực hiện XOR kết quả trên với ciphertext sử dụng trong lần giải mã trước để tạo ra plaintext

2.3 Ưu điểm và nhược điểm

2.3.1 Ưu điểm

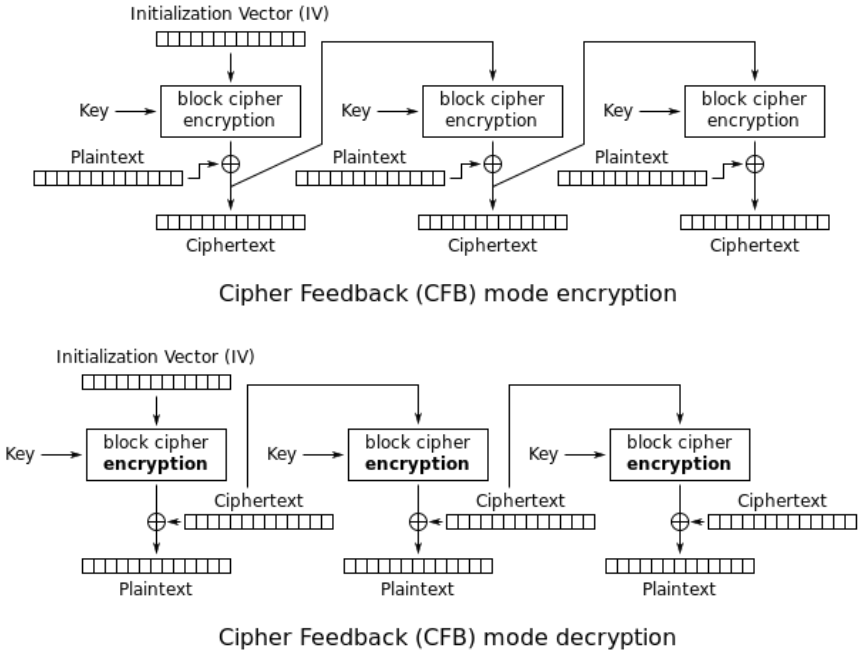
- Khả năng bảo mật cao hơn ECB. Ciphertext của một khối dữ liệu plaintext có thể khác nhau cho mỗi lần mã hóa vì nó phụ thuộc vào ***IV*** hoặc giá trị mã hóa (ciphertext) của khối dữ liệu liền trước.
- Quá trình giải mã (mã hóa nghịch) vẫn có thể thực hiện song song nhiều khối dữ liệu.

2.3.2 Nhược điểm

- Thiết kế phần cứng phức tạp hơn ECB ngoài logic thực thi thuật toán mã hóa, người thiết kế cần thiết kế thêm:
  - Logic quản lý độ dài chuỗi dữ liệu sẽ được mã hóa (cụ thể ở đây là số lượng khối dữ liệu của dữ liệu)
  - Bộ tạo giá trị ngẫu nhiên cho ***IV***.
- Lỗi bit bị lan truyền. Nếu một lỗi bit xuất hiện trên ciphertext của một khối dữ liệu thì nó sẽ làm ảnh hưởng đến kết quả giải mã của khối đữ liệu đó và khối dữ liệu tiếp theo.
- Không thể thực thi quá trình mã hóa song song vì xử lý của khối dữ liệu sau phụ thuộc vào ciphertext của khối dữ liệu trước, trừ lần mã hóa đầu tiên.

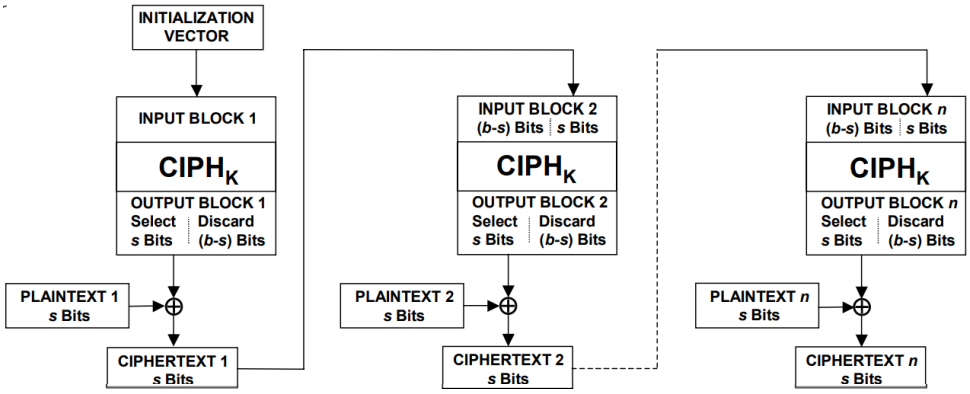
II.3 CFB (Cipher Feedback)

CFB là chế độ mã hóa mà ciphertext của lần mã hóa hiện tại sẽ được phản hồi (feedback) đến đầu vào của lần mã hóa tiếp theo. Nghĩa là, ciphertext của lần mã hóa hiện tại sẽ được sử dụng để tính toán ciphertext của lần mã hóa kế tiếp. Mô tả có vẻ giống CBC nhưng quá trình trực hiện lại khác.



Hình 6: Chế độ mã hóa/giải mã CFB

3.1 Quá trình mã hóa CFB



Hình 7: Chế độ mã hóa CFB

Biểu thức định nghĩa:

$$I_1 = IV$$
$$I_j = LSB_{b-s}(I_{j-1}) \mid C\#_{j-1} \quad \text{với } j=2, 3, \ldots, n$$
$$Oj = CIPH_K(I_j) \quad \text{với } j=1, 2, \ldots, n$$
$$C\#_j = P\#_j \oplus MSB_s(O_j) \quad \text{với } j=1, 2, \ldots, n$$

(1) Lần mã hóa đầu tiên:

- Khối dữ liệu ngõ vào của quá trình mã hóa lấy từ ***IV***, ứng với biểu thức ***I1***,
- Vector khởi tạo ***IV*** được mã hóa để tạo ra một khối giá trị chứa b bit, ứng với biểu thức O_j
- s bit MSB của kết quả trên sẽ được dùng để XOR với **s** bit dữ liệu (plaintext) để tạo ra **s** bit ciphertext, ứng với biểu thức $C\#_j$

IV là giá trị không cần bảo mật nhưng phải là giá trị không thể dự đoán trước được. ***IV*** có độ dài là **b** bit, bằng độ dài của một khối dữ liệu mà chuẩn mã hóa quy định.

s là một số nguyên và $1 \le s \le b$, đây chính là độ dài plaintext và ciphertext cho một lần mã hóa/giải mã. Giá trị của **s** có thể được tích hợp vào tên gọi của chế độ CFB để chỉ rõ chế độ số lượng bit được hỗ trợ. Ví dụ:

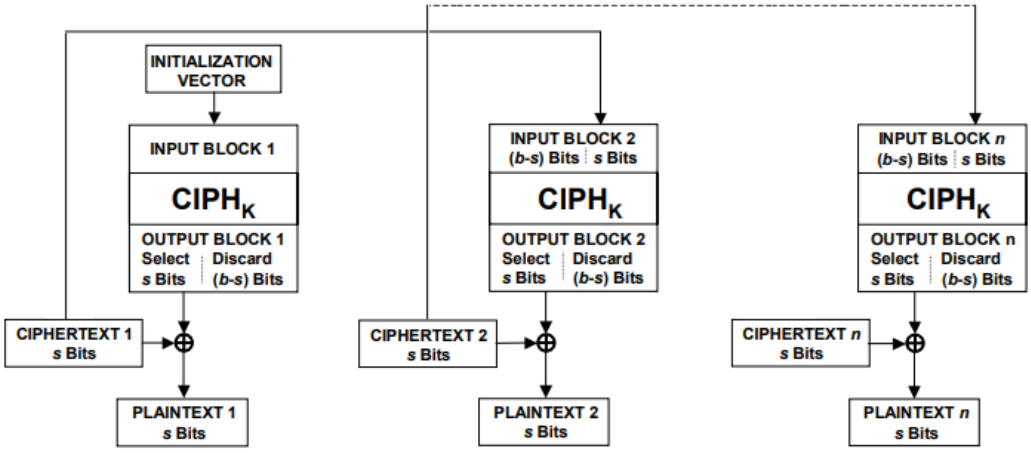
- 1-bit CFB – Mỗi lần thực hiện mã hóa/giải mã 1 bit
- 8-bit CFB – Mỗi lần thực hiện mã hóa/giải mã 8 bit
- 128-bit CFB – Mỗi lần thực hiện mã hóa/giải mã 128 bit

(n) Lần mã hóa sau lần đầu tiên:

- Khối dữ liệu ngõ vào của quá trình mã hóa được ghép giã **b-s** bit LSB của khối ngõ vào của lần mã hóa trước đó và **s** bit của ciphertext của lần mã hóa trước đó, ứng với biểu thức I_j,
- Giá trị của bước trên được mã hóa để tạo ra một khối giá trị chứa **b** bit, ứng với biểu thức O_j

- **s** bit MSB của kết quả trên sẽ được dùng để XOR với **s** bit dữ liệu (plaintext) để tạo ra **s** bit ciphertext, ứng với biểu thức $C^{\#}_j$

3.2 Quá trình giải mã CFB



Hình 8: Chế độ giải mã CFB

Biểu thức định nghĩa:

$$
\begin{aligned}
&I_1 = IV \\
&I_j = LSB_{b-s}(I_{j-1}) \mid C^{\#}_{j-1} && \text{với } j=2, 3, \ldots, n \\
&O_j = CIPH_K(I_j) && \text{với } j=1, 2, \ldots, n \\
&P^{\#}_j = C^{\#}_j \oplus MSB_s(O_j) && \text{với } j=1, 2, \ldots, n
\end{aligned}
$$

Để giải mã, cùng một lược đồ được sử dụng, ngoại trừ đơn vị bản mã nhận được được XOR với đầu ra của hàm mã hóa để tạo ra đơn vị bản rõ. Lưu ý rằng đó là chức năng mã hóa được sử dụng, không phải là chức năng giải mã. Trong lúc ấy:

$$C^{\#}_1 = P^{\#}_1 \oplus MSB_s(CIPH_K(IV))$$

Do đó:

$$P^{\#}_1 = C^{\#}_1 \oplus MSB_s(CIPH_K(IV))$$

Lập luận tương tự cũng được áp dụng cho các bước tiếp theo trong quy trình giải mã.

3.3 Ưu điểm và nhược điểm

So sánh với chế độ CBC, chế độ CFB có những điểm khác biệt sau đây:

- Thuật toán mã hóa không áp dụng trực tiếp trên plaintext mà dùng để biến đổi một khối dữ liệu sinh ra từ **IV** và ciphertext
- Số lượng bit dữ liệu được mã hóa, giải mã có thể nhỏ hơn hoặc bằng số lượng bit mà thuật toán mã hóa hỗ trợ, $1 \le s \le b$

3.3.1 Ưu điểm

- Khả năng bảo mật cao hơn ECB. Ciphertext của một khối dữ liệu plaintext có thể khác nhau cho mỗi lần mã hóa vì nó phụ thuộc vào **IV** hoặc giá trị mã hóa (ciphertext) của khối dữ liệu liền trước.

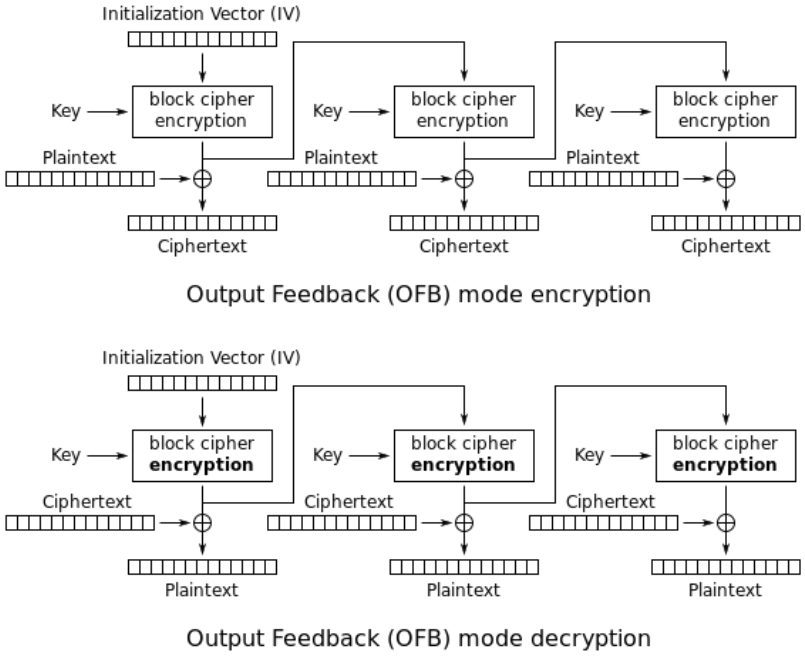
- Quá trình giải mã (mã hóa nghịch) vẫn có thể thực hiện song song nhiều khối dữ liệu.
- Tùy biến được độ dài khối dữ liệu mã hóa, giải mã thông qua thông số **s**

3.3.2 Nhược điểm

- Thiết kế phần cứng phức tạp hơn CBC. Ngoài những thành phần logic như CBC, CFB cần thêm logic để chọn số bit cần được xử lý nếu **s** là thông số cấu hình được.
- Lỗi bit bị lan truyền. Nếu một lỗi bit xuất hiện trên ciphertext của một khối dữ liệu thì nó sẽ làm sai kết quả giải mã của khối đữ liệu đó và khối dữ liệu tiếp theo.
- Không thể thực thi quá trình mã hóa song song vì xử lý của khối dữ liệu sau phụ thuộc vào ciphertext của khối dữ liệu trước, trừ lần mã hóa đầu tiên

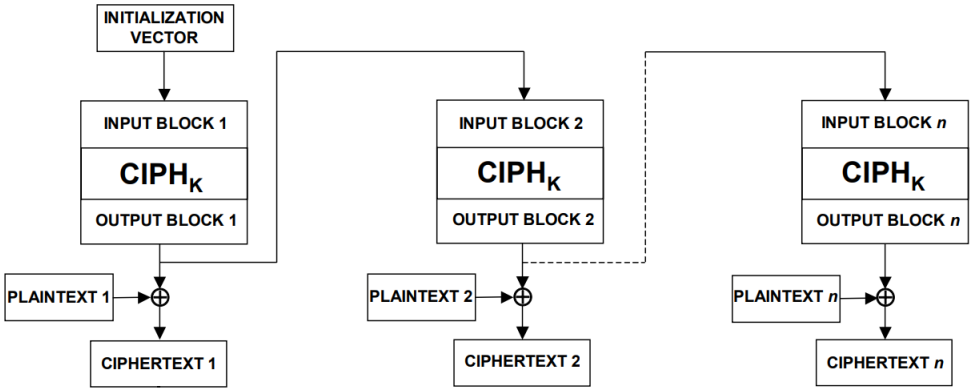
II.4 OFB (Output Feedback)

OFB là chế độ mã hóa mà giá trị ngõ ra của khối thực thi thuật toán mã hóa, **không phải ciphertext**, của lần mã hóa hiện tại sẽ được phản hồi (feedback) đến ngõ vào của lần mã hóa kế tiếp.



Hình 9: Chế độ mã hóa, giải mã OFB

4.1 Quá trình mã hóa OFB



Hình 10: Chế độ mã hóa OFB

Biểu thức định nghĩa:

$$I_1 = IV$$
$$I_j = Oj\text{-}1 \quad \text{với } j = 2, 3, \ldots, n$$
$$O_j = CIPH_k(I_j) \quad \text{với } j = 1, 2, \ldots, n$$
$$C_j = P_j \oplus O_j \quad \text{với } j = 1, 2, \ldots, n\text{-}1$$
$$C^*_n = P^*_n \oplus MSB_u(O_n)$$

(1) Lần mã hóa đầu tiên:

- Giá trị **IV** được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_1}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR plaintext và kết quả của bước trên, ứng với biểu thức $\mathbf{C_j}$

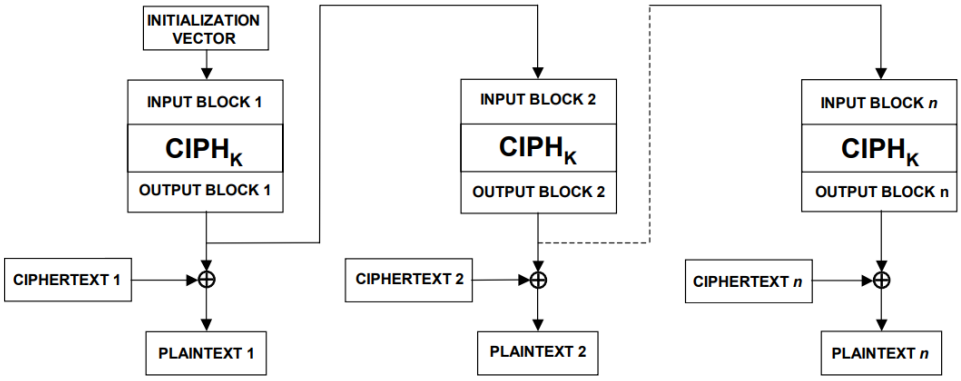
(2) Lần mã hóa sau lần đầu tiên và trước lần cuối cùng:

- Giá trị của khối ngõ ra (output block) trước đó được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_j}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR plaintext và kết quả của bước trên, ứng với biểu thức $\mathbf{C_j}$

(n) Lần mã hóa cuối cùng (thứ n):

- Giá trị của khối ngõ ra (output block) trước đó được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_j}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR plaintext và với các bit MSB của kết quả của bước trên *theo độ dài của plaintext (độ dài plaintext của lần cuối cùng có thể ngắn hơn độ dài của khối ngõ ra)*, ứng với biểu thức $\mathbf{C^*_n}$

4.2 Quá trình giải mã OFB



Hình 10: Chế độ giải mã OFB

Biểu thức định nghĩa:

$$I_1 = IV$$
$$I_j = O_{j-1} \quad \text{với } j = 2, 3, \ldots, n$$
$$O_j = CIPH_k(I_j) \quad \text{với } j = 1, 2, \ldots, n$$
$$P_j = C_j \oplus O_j \quad \text{với } j = 1, 2, \ldots, n-1$$
$$P^*_n = C^*_n \oplus MSB_u(O_n)$$

(1) Lần giải mã đầu tiên:

- Giá trị **IV** được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_1}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR ciphertext và kết quả của bước trên, ứng với biểu thức $\mathbf{P_j}$

(3) Lần giải mã sau lần đầu tiên và trước lần cuối cùng:

- Giá trị của khối ngõ ra (output block) trước đó được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_j}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR ciphertext và kết quả của bước trên, ứng với biểu thức $\mathbf{P_j}$

(n) Lần giải mã cuối cùng (thứ n):

- Giá trị của khối ngõ ra (output block) trước đó được lấy làm khối giá trị đầu vào mã hóa, ứng với biểu thức $\mathbf{I_j}$
- Thực thi giải thuật mã hóa cho khối trên với khóa mã **K**, ứng với biểu thức $\mathbf{O_j}$
- XOR ciphertext và với các bit MSB của kết quả của bước trên *theo độ dài của ciphertext (độ dài ciphertext của lần cuối cùng có thể ngắn hơn độ dài của khối ngõ ra)*, ứng với biểu thức $\mathbf{P^*_n}$

4.3 Ưu điểm và nhược điểm

Chế độ OFB có những đặc điểm cần chú ý như sau:

- Thuật toán mã hóa không áp dụng trực tiếp trên plaintext mà dùng để biến đổi một khối dữ liệu sinh ra từ **IV** và khối ngõ ra của lần mã hóa trước đó. Điểm này tương tự với CFB.
- OFB khác với CFB là nó xử lý trên một khối dữ liệu với độ dài bit đầy đủ như thuật toán mã hóa quy định chứ không xử lý trên một phần hay một vài bit của khối dữ liệu.

4.3.1 Ưu điểm

- Khả năng bảo mật cao hơn ECB. Ciphertext của một khối dữ liệu plaintext có thể khác nhau cho mỗi lần mã hóa vì nó phụ thuộc vào **IV** hoặc khối ngõ ra của lần mã hóa trước đó.
- Lỗi bit không bị lan truyền. Khi một lỗi bit xuất hiện trên một ciphertext, nó chỉ ảnh hưởng đến kết quả giải mã của khối dữ liệu hiện tại
- Thiết kế phần cứng đơn giản hơn CFB.

4.3.2 Nhược điểm

Không thể thực hiện mã hóa/giải mã song song nhiều khối dữ liệu vì lần mã hóa/giải mã sau phụ thuộc vào khối ngõ ra của lần mã hóa/giải mã liền trước nó.

II.5 Kết luận về các chế độ mã hoá khối

5.1 Tổng quan

Chế độ mã hoá khối	**Mô tả**	**Ứng dụng**
Electronic Codebook (ECB)	Các khối plaintext được mã hoá độc lập với nhau bởi cùng 1 key duy nhất	• Đảm bảo cho việc truyền tin cho các single value
Cipher Block Chaining (CBC)	Đầu vào của hàm mã hoá là kết quả của phép XOR giữa khối plaintext tiếp theo và khối ciphertext trước đó	• Áp dụng cho hầu hết việc truyền dữ liệu theo khối (block-oriented transmission) • Authentication – Xác thực
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	• Thường dùng cho hầu hết việc truyền dữ liệu theo dòng (stream-oriented transmission) • Authentication – Xác thực
Output Feedback (OFB)	Tương tự như CFB, nhưng input của hàm mã hoá tiếp theo là đầu ra của hàm mã hoá trước đó	Stream-oriented transmission cho các kênh bị nhiễu (ví dụ: giao tiếp giữa các vệ tinh)

5.2 Truy cập ngẫu nhiên và khả năng xử lý song song

Chế độ mã hoá khối	**Khả năng mã hoá song song**	**Khả năng Giải mã song song**	**Truy cập ngẫu nhiên**
Electronic Codebook (ECB)	Có	Có	Có
Cipher Block Chaining (CBC)	Không	Có	Có
Cipher Feedback (CFB)	Không	Có	Có
Output Feedback (OFB)	Không	Không	Không

5.3 Sự lan truyền lỗi

Chế độ mã hoá khối	**Sự ảnh hưởng khi thay đổi bit trong C_i**	**Sự ảnh hưởng khi thay đổi bit trong IV**
Electronic Codebook (ECB)	Lỗi bit ngẫu nhiên trong P_i	Không có IV
Cipher Block Chaining (CBC)	Lỗi bit ngẫu nhiên trong P_i Lỗi bit cụ thể trong các P_{i+1}	Lỗi bit cụ thể trong việc giải mã C_1
Cipher Feedback (CFB)	Lỗi bit cụ thể trong P_i Lỗi bit ngẫu nhiên trong các P_{i+1}	Lỗi bit ngẫu nhiên trong việc giải mã C_1
Output Feedback (OFB)	Lỗi bit ngẫu nhiên trong P_i	Lỗi bit ngẫu nhiên trong việc giải mã các khối $C_1,C_2,\ldots C_n$

III. Kết hợp các chế độ mã hoá khối với thuật toán mã hoá

Ở đây tụi em sẽ sử dụng thuật toán AES cho mỗi chế độ giải mã/mã hoá dữ liệu, để triển khai theo mô hình Mã hoá khối. Các ví dụ sẽ có chung khoá Key có size là 128 bit và một IV cũng có size 128 bit.

```
Key: 2b7e151628aed2a6abf7158809cf4f3c
 IV: 000102030405060708090a0b0c0d0e0f
```

Plaintext cho tất cả các ví dụ này sẽ tương đương với chuỗi ký tự thập lục phân, được định dạng thành bốn khối 128 bit như sau:

```
Block 1: 6bc1bee22e409f96e93d7e117393172a
Block 2: ae2d8a571e03ac9c9eb76fac45af8e51
Block 3: 30c81c46a35ce411e5fbc1191a0a52ef
Block 4: f69f2445df4f9b17ad2b417be66c3710
```

III.1 ECB (Electronic Codebook)

1.1 Thực hiện mã hoá ECB-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
Block #1	
Plaintext	6bc1bee22e409f96e93d7e117393172a
Input Block	6bc1bee22e409f96e93d7e117393172a
Output Block	3ad77bb40d7a3660a89ecaf32466ef97
Ciphertext	3ad77bb40d7a3660a89ecaf32466ef97
Block #2	
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Input Block	ae2d8a571e03ac9c9eb76fac45af8e51
Output Block	f5d3d58503b9699de785895a96fdbaaf
Ciphertext	f5d3d58503b9699de785895a96fdbaaf
Block #3	
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Input Block	30c81c46a35ce411e5fbc1191a0a52ef
Output Block	43b1cd7f598ece23881b00e3ed030688
Ciphertext	43b1cd7f598ece23881b00e3ed030688
Block #4	
Plaintext	f69f2445df4f9b17ad2b417be66c3710
Input Block	f69f2445df4f9b17ad2b417be66c3710
Output Block	7b0c785e27e8ad3f8223207104725dd4
Ciphertext	7b0c785e27e8ad3f8223207104725dd4

1.2 Thực hiện giải mã ECB-AES128.Decrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
Block #1	
Plaintext	3ad77bb40d7a3660a89ecaf32466ef97
Input Block	3ad77bb40d7a3660a89ecaf32466ef97

Output Block	6bc1bee22e409f96e93d7e117393172a
Ciphertext	6bc1bee22e409f96e93d7e117393172a
Block #2	
Plaintext	f5d3d58503b9699de785895a96fdbaaf
Input Block	f5d3d58503b9699de785895a96fdbaaf
Output Block	ae2d8a571e03ac9c9eb76fac45af8e51
Ciphertext	ae2d8a571e03ac9c9eb76fac45af8e51
Block #3	
Plaintext	43b1cd7f598ece23881b00e3ed030688
Input Block	43b1cd7f598ece23881b00e3ed030688
Output Block	30c81c46a35ce411e5fbc1191a0a52ef
Ciphertext	30c81c46a35ce411e5fbc1191a0a52ef
Block #4	
Plaintext	7b0c785e27e8ad3f8223207104725dd4
Input Block	7b0c785e27e8ad3f8223207104725dd4
Output Block	f69f2445df4f9b17ad2b417be66c3710
Ciphertext	f69f2445df4f9b17ad2b417be66c3710

III.2 CBC (Cipher Block Chaining)

2.1 Thực hiện mã hoá CBC-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Block #1	
Plaintext	6bc1bee22e409f96e93d7e117393172a
Input Block	6bc0bce12a459991e134741a7f9e1925
Output Block	7649abac8119b246cee98e9b12e9197d
Ciphertext	7649abac8119b246cee98e9b12e9197d
Block #2	
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Input Block	d86421fb9f1a1eda505ee1375746972c
Output Block	5086cb9b507219ee95db113a917678b2
Ciphertext	5086cb9b507219ee95db113a917678b2
Block #3	
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Input Block	604ed7ddf32efdff7020d0238b7c2a5d
Output Block	73bed6b8e3c1743b7116e69e22229516
Ciphertext	73bed6b8e3c1743b7116e69e22229516
Block #4	
Plaintext	f69f2445df4f9b17ad2b417be66c3710
Input Block	8521f2fd3c8eef2cdc3da7e5c44ea206
Output Block	3ff1caa1681fac09120eca307586e1a7
Ciphertext	3ff1caa1681fac09120eca307586e1a7

2.2 Thực hiện giải mã CBC-AES128.Decrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Block #1	

Ciphertext	7649abac8119b246cee98e9b12e9197d
Input Block	7649abac8119b246cee98e9b12e9197d
Output Block	6bc0bce12a459991e134741a7f9e1925
Plaintext	6bc1bee22e409f96e93d7e117393172a
Block #2	
Ciphertext	5086cb9b507219ee95db113a917678b2
Input Block	5086cb9b507219ee95db113a917678b2
Output Block	d86421fb9f1a1eda505ee1375746972c
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Block #3	
Ciphertext	73bed6b8e3c1743b7116e69e22229516
Input Block	73bed6b8e3c1743b7116e69e22229516
Output Block	604ed7ddf32efdff7020d0238b7c2a5d
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Block #4	
Ciphertext	3ff1caa1681fac09120eca307586e1a7
Input Block	3ff1caa1681fac09120eca307586e1a7
Output Block	8521f2fd3c8eef2cdc3da7e5c44ea206
Plaintext	f69f2445df4f9b17ad2b417be66c3710

III.3 CFB (Cipher Feedback)

Ở CFB, các plaintext hay ciphertext, thông thường quá trình mã hoá/giải mã sẽ diễn ra theo 3 kiểu: xử lý theo từng bit (bit-by-bit) xử lý theo từng byte (byte-by-byte) và xử lý theo từng block (block-by-block). Ở đây, nhóm em sẽ thực hiện triển khai ở 2 kiểu đó là

- byte-by-byte CFB8 (đối với chế độ này,tụi em sẽ thực hiện xử lý ở duy nhất block 1 của plaintext)
- block-by-block CFB128

3.1 Thực hiện mã hoá CFB8-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Segment #1	
Input Block	000102030405060708090a0b0c0d0e0f
Output Block	7649abac8119b246cee98e9b12e9197d
Plaintext	6b
Ciphertext	**3b**
Segment #2	
Input Block	0102030405060708090a0b0c0d0e0f**3b**
Output Block	b8eb865a2b026381abb1d6560ed20f68
Plaintext	c1
Ciphertext	**79**
Segment #3	
Input Block	02030405060708090a0b0c0d0e0f3b**79**
Output Block	fce6033b4edce64cbaed3f61ff5b927c
Plaintext	be
Ciphertext	**42**

Segment #4	
Input Block	030405060708090a0b0c0d0e0f3b79**42**
Output Block	ae4e5e7ffe805f7a4395b180004f8ca8
Plaintext	e2
Ciphertext	**4c**
Segment #5	
Input Block	0405060708090a0b0c0d0e0f3b7942**4c**
Output Block	b205eb89445b62116f1deb988a81e6dd
Plaintext	2e
Ciphertext	**9c**
Segment #6	
Input Block	05060708090a0b0c0d0e0f3b79424c**9c**
Output Block	4d21d456a5e239064fff4be0c0f85488
Plaintext	40
Ciphertext	**0d**
Segment #7	
Input Block	060708090a0b0c0d0e0f3b79424c9c**0d**
Output Block	4b2f5c3895b9efdc85ee0c5178c7fd33
Plaintext	9f
Ciphertext	**d4**
Segment #8	
Input Block	0708090a0b0c0d0e0f3b79424c9c0d**d4**
Output Block	a0976d856da260a34104d1a80953db4c
Plaintext	96
Ciphertext	**36**
Segment #9	
Input Block	08090a0b0c0d0e0f3b79424c9c0dd4**36**
Output Block	53674e5890a2c71b0f6a27a094e5808c
Plaintext	e9
Ciphertext	**ba**
Segment #10	
Input Block	090a0b0c0d0e0f3b79424c9c0dd436**ba**
Output Block	f34cd32ffed495f8bc8adba194eccb7a
Plaintext	3d
Ciphertext	**ce**
Segment #11	
Input Block	0a0b0c0d0e0f3b79424c9c0dd436ba**ce**
Output Block	e08cf2407d7ed676c9049586f1d48ba6
Plaintext	7e
Ciphertext	**9e**
Segment #12	
Input Block	0b0c0d0e0f3b79424c9c0dd436bace**9e**
Output Block	1f5c88a19b6ca28e99c9aeb8982a6dd8
Plaintext	11
Ciphertext	**0e**
Segment #13	

Input Block	0c0d0e0f3b79424c9c0dd436bace9e**0e**
Output Block	a70e63df781cf395a208bd2365c8779b
Plaintext	73
Ciphertext	**d4**
Segment #14	
Input Block	0d0e0f3b79424c9c0dd436bace9e0e**d4**
Output Block	cbcfe8b3bcf9ac202ce18420013319ab
Plaintext	93
Ciphertext	**58**
Segment #15	
Input Block	0e0f3b79424c9c0dd436bace9e0ed4**58**
Output Block	7d9fac6604b3c8c5b1f8c5a00956cf56
Plaintext	17
Ciphertext	**6a**
Segment #16	
Input Block	0f3b79424c9c0dd436bace9e0ed458**6a**
Output Block	65c3fa64bf0343986825c636f4a1efd2
Plaintext	2a
Ciphertext	**4f**

3.2 Thực hiện giải mã CFB8-AES128.Decrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Segment #1	
Input Block	000102030405060708090a0b0c0d0e0f
Output Block	50fe67cc996d32b6da0937e99bafec60
Ciphertext	**3b**
Plaintext	6b
Segment #2	
Input Block	0102030405060708090a0b0c0d0e0f**3b**
Output Block	b8eb865a2b026381abb1d6560ed20f68
Ciphertext	**79**
Plaintext	c1
Segment #3	
Input Block	02030405060708090a0b0c0d0e0f3b**79**
Output Block	fce6033b4edce64cbaed3f61ff5b927c
Ciphertext	**42**
Plaintext	be
Segment #4	
Input Block	030405060708090a0b0c0d0e0f3b79**42**
Output Block	ae4e5e7ffe805f7a4395b180004f8ca8
Ciphertext	**4C**
Plaintext	E2
Segment #5	
Input Block	0405060708090a0b0c0d0e0f3b7942**4c**
Output Block	b205eb89445b62116f1deb988a81e6dd
	9c

Ciphertext Plaintext	2e
Segment #6	
Input Block Output Block Ciphertext Plaintext	05060708090a0b0c0d0e0f3b79424c**9c** 4d21d456a5e239064fff4be0c0f85488 **0d** 40
Segment #7	
Input Block Output Block Ciphertext Plaintext	060708090a0b0c0d0e0f3b79424c9c**0d** 4b2f5c3895b9efdc85ee0c5178c7fd33 **d4** 9f
Segment #8	
Input Block Output Block Ciphertext Plaintext	0708090a0b0c0d0e0f3b79424c9c0d**d4** a0976d856da260a34104d1a80953db4c **36** 96
Segment #9	
Input Block Output Block Ciphertext Plaintext	08090a0b0c0d0e0f3b79424c9c0dd4**36** 53674e5890a2c71b0f6a27a094e5808c **ba** e9
Segment #10	
Input Block Output Block Ciphertext Plaintext	090a0b0c0d0e0f3b79424c9c0dd436**ba** f34cd32ffed495f8bc8adba194eccb7a **ce** 3d
Segment #11	
Input Block Output Block Ciphertext Plaintext	0a0b0c0d0e0f3b79424c9c0dd436ba**ce** e08cf2407d7ed676c9049586f1d48ba6 **9e** 7e
Segment #12	
Input Block Output Block Ciphertext Plaintext	0b0c0d0e0f3b79424c9c0dd436bace**9e** 1f5c88a19b6ca28e99c9aeb8982a6dd8 **0e** 11
Segment #13	
Input Block Output Block Ciphertext Plaintext	0c0d0e0f3b79424c9c0dd436bace9e**0e** a70e63df781cf395a208bd2365c8779b **d4** 73
Segment #14	
Input Block Output Block	0d0e0f3b79424c9c0dd436bace9e0e**d4** cbcfe8b3bcf9ac202ce18420013319ab **58**

Ciphertext Plaintext	93
Segment #15	
Input Block Output Block Ciphertext Plaintext	0e0f3b79424c9c0dd436bace9e0ed4**58** 7d9fac6604b3c8c5b1f8c5a00956cf56 **6a** 17
Segment #16	
Input Block Output Block Ciphertext Plaintext	0f3b79424c9c0dd436bace9e0ed458**6a** 65c3fa64bf0343986825c636f4a1efd2 **4f** 2a

3.3 Thực hiện mã hoá CFB128-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Segment #1	
Input Block Output Block Plaintext Ciphertext	000102030405060708090a0b0c0d0e0f 50fe67cc996d32b6da0937e99bafec60 6bc1bee22e409f96e93d7e117393172a 3b3fd92eb72dad20333449f8e83cfb4a
Segmnet #2	
Input Block Output Block Plaintext Ciphertext	3b3fd92eb72dad20333449f8e83cfb4a 668bcf60beb005a35354a201dab36bda ae2d8a571e03ac9c9eb76fac45af8e51 c8a64537a0b3a93fcde3cdad9f1ce58b
Segment #3	
Input Block Output Block Plaintext Ciphertext	c8a64537a0b3a93fcde3cdad9f1ce58b 16bd032100975551547b4de89daea630 30c81c46a35ce411e5fbc1191a0a52ef 26751f67a3cbb140b1808cf187a4f4df
Segment #4	
Input Block Output Block Plaintext Ciphertext	26751f67a3cbb140b1808cf187a4f4df 36d42170a312871947ef8714799bc5f6 f69f2445df4f9b17ad2b417be66c3710 c04b05357c5d1c0eeac4c66f9ff7f2e6

3.4 Thực hiện giải mã CFB128-AES128.Decrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Segment #1	
Input Block Output Block Plaintext Ciphertext	50fe67cc996d32b6da0937e99bafec60 000102030405060708090a0b0c0d0e0f 3b3fd92eb72dad20333449f8e83cfb4a 6bc1bee22e409f96e93d7e117393172a
Segmnet #2	
Input Block	668bcf60beb005a35354a201dab36bda

Output Block	3b3fd92eb72dad20333449f8e83cfb4a
Plaintext	c8a64537a0b3a93fcde3cdad9f1ce58b
Ciphertext	ae2d8a571e03ac9c9eb76fac45af8e51
Segment #3	
Input Block	16bd032100975551547b4de89daea630
Output Block	c8a64537a0b3a93fcde3cdad9f1ce58b
Plaintext	26751f67a3cbb140b1808cf187a4f4df
Ciphertext	30c81c46a35ce411e5fbc1191a0a52ef
Segment #4	
Input Block	36d42170a312871947ef8714799bc5f6
Output Block	26751f67a3cbb140b1808cf187a4f4df
Plaintext	c04b05357c5d1c0eeac4c66f9ff7f2e6
Ciphertext	f69f2445df4f9b17ad2b417be66c3710

III.4 OFB (Output Feedback)

4.1 Thực hiện mã hoá OFB-AES128.Encrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Block #1	
Input Block	000102030405060708090a0b0c0d0e0f
Output Block	50fe67cc996d32b6da0937e99bafec60
Plaintext	6bc1bee22e409f96e93d7e117393172a
Ciphertext	3b3fd92eb72dad20333449f8e83cfb4a
Block #2	
Input Block	50fe67cc996d32b6da0937e99bafec60
Output Block	d9a4dada0892239f6b8b3d7680e15674
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Ciphertext	7789508d16918f03f53c52dac54ed825
Block #3	
Input Block	d9a4dada0892239f6b8b3d7680e15674
Output Block	a78819583f0308e7a6bf36b1386abf23
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Ciphertext	9740051e9c5fecf64344f7a82260edcc
Block #4	
Input Block	a78819583f0308e7a6bf36b1386abf23
Output Block	c6d3416d29165c6fcb8e51a227ba994e
Plaintext	f69f2445df4f9b17ad2b417be66c3710
Ciphertext	304c6528f659c77866a510d9c1d6ae5e

4.2 Thực hiện giải mã OFB-AES128.Decrypt

Key	2b7e151628aed2a6abf7158809cf4f3c
IV	000102030405060708090a0b0c0d0e0f
Block #1	
Input Block	000102030405060708090a0b0c0d0e0f
Output Block	50fe67cc996d32b6da0937e99bafec60
Ciphertext	3b3fd92eb72dad20333449f8e83cfb4a
Plaintext	6bc1bee22e409f96e93d7e117393172a

Block #2	
Input Block	50fe67cc996d32b6da0937e99bafec60
Output Block	d9a4dada0892239f6b8b3d7680e15674
Ciphertext	7789508d16918f03f53c52dac54ed825
Plaintext	ae2d8a571e03ac9c9eb76fac45af8e51
Block #3	
Input Block	d9a4dada0892239f6b8b3d7680e15674
Output Block	a78819583f0308e7a6bf36b1386abf23
Ciphertext	9740051e9c5fecf64344f7a82260edcc
Plaintext	30c81c46a35ce411e5fbc1191a0a52ef
Block #4	
Input Block	a78819583f0308e7a6bf36b1386abf23
Output Block	c6d3416d29165c6fcb8e51a227ba994e
Ciphertext	304c6528f659c77866a510d9c1d6ae5e
Plaintext	f69f2445df4f9b17ad2b417be66c3710

III.5 Mã hoá hình ảnh

5.1 Làm quen với Openssl

Openssl là một framework hỗ trợ cho toàn bộ các ứng dụng bảo mật truyền thông thông qua mạng máy tính. Ở đây, tụi em sẽ sử dụng nó như một công cụ để thực hiện giải mã và mã hoá dữ liệu thông qua các chế độ mã hoá và thuật toán mã hoá.

Cấu trúc của một câu lệnh openssl sẽ như sau:

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

Chúng ta cần thay đổi giá trị `-ciphertype` bằng kiểu mã hoá mà ta muốn sử dụng, ví dụ như `-aes-128-ecb`, `-bf-cbc`, `-des-cfb`, … Ngoài ra chúng ta cần phải sử dụng một vài tuỳ chọn khác cho lệnh `openssl enc`, chúng được liệt kê như sau:

```
-in               input file
-out              output file
-e                mã hoá
-d                giải mã
-K/-iv            khoá/iv được biểu diễn dưới dạng hexa
-[pP]             in ra giá trị của iv/khoá
```

5.2 Thực hiện mã hoá

Ở nội dung này, tụi em sẽ thực hiện mã hoá hình ảnh ở cả 4 chế độ và so sánh kết quả mã hoá của chúng có gì khác biệt. Chúng ta có một file ảnh như sau:

Để thực hiện mã hoá, tụi em sẽ tạo một file bash scripts ***enc.sh*** và thực hiện qua nó. Lần lượt mã hoá 4 chế độ với thuật toán mã hoá AES128

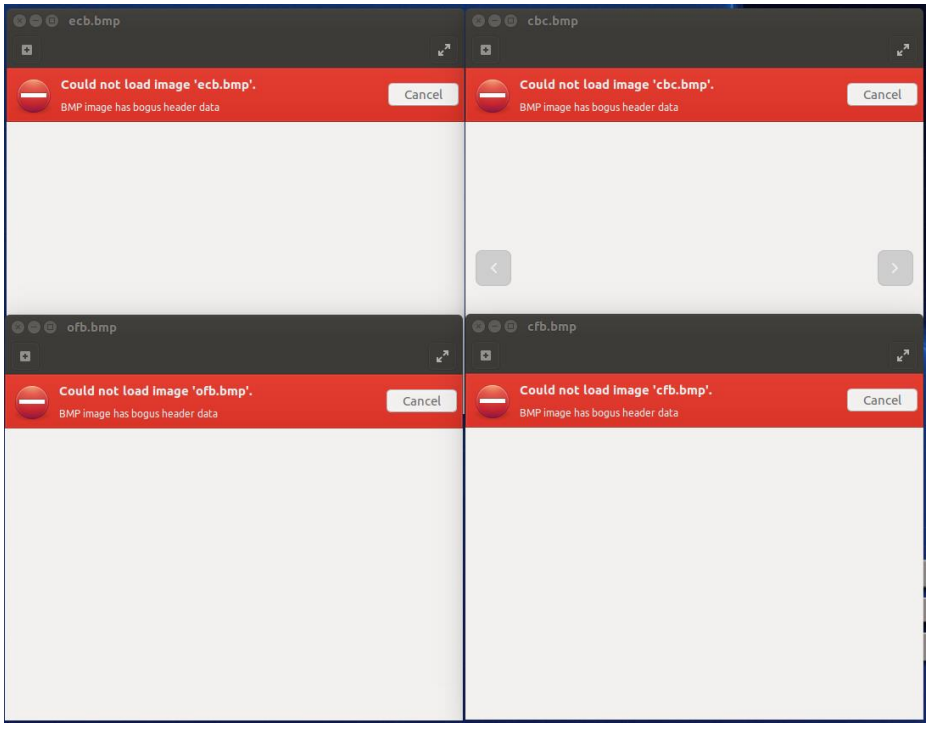
```
#!/bin/bash
#mã hoá với ECB
openssl enc -aes-128-ecb -e -in pic_original.bmp -out ecb.bmp
-K 2b7e151628aed2a6abf7158809cf4f3c
# mã hoá với CBC
openssl enc -aes-128-cbc -e -in pic_original.bmp -out cbc.bmp
-K 2b7e151628aed2a6abf7158809cf4f3c   /
-iv 000102030405060708090a0b0c0d0e0f
# mã hoá với CFB
openssl enc -aes-128-cfb -e -in pic_original.bmp -out cfb.bmp
-K 2b7e151628aed2a6abf7158809cf4f3c    /
```

```
-iv 000102030405060708090a0b0c0d0e0f
# mã hoá với OFB
openssl enc -aes-128-ofb -e -in pic_original.bmp -out ofb.bmp
-K 2b7e151628aed2a6abf7158809cf4f3c   /
-iv 000102030405060708090a0b0c0d0e0f
```

Sau khi tiến hành mã hoá, ta được các tệp tin như sau:

```
[12/10/21]seed@Server:~/.../images$ zenc.sh
[12/10/21]seed@Server:~/.../images$ ls -la
total 940
drwxrwxr-x 3 seed seed   4096 Dec 10 16:12 .
drwxrwxr-x 5 seed seed   4096 Dec 10 15:34 ..
drwxrwxr-x 2 seed seed   4096 Dec 10 15:27 1
-rw-rw-r-- 1 seed seed 184976 Dec 10 16:12 cbc.bmp
-rw-rw-r-- 1 seed seed 184974 Dec 10 16:12 cfb.bmp
-rw-rw-r-- 1 seed seed 184976 Dec 10 16:12 ecb.bmp
-rw-rw-r-- 1 seed seed 184974 Dec 10 16:12 ofb.bmp
-rw-rw-r-- 1 seed seed 184974 Dec  9 18:26 pic_original.bmp
```

Khi mà cố gắng xem các tập tin hình ảnh đã được mã hoá bằng một phần mềm xem ảnh bất kì, nó sẽ xuất hiện thông báo lôi "BMP Image has bogus header data" bởi vì 54 bytes đầu tiên của tệp chứa tiêu đề thông tin của tệp đó (hay Header của hình ảnh) mà khi mã hoá ta đã thực hiện thay đổi các giá trị ở 54 bytes này nên ta sẽ không xem được hình ảnh



Để có thể xem được các tập tin ảnh này, ta cần phải tiến hành chỉnh sửa các giá trị của 54 byte đầu tiên của các tệp đã mã hoá giống như 54 byte đầu tiên của hình ảnh gốc ban đầu – tức nhiệm vụ chính là chỉnh sửa thành phần header của tập tin.

Ở đây, tụi em có một bash scripts đơn giản ***fix.sh*** để thực hiện việc chỉnh sửa

```
#!/bin/bash

head -c 54 pic_original.bmp > header
```

```
tail -c +55 ecb.bmp > body_ecb
cat header body_ecb > fix_ecb.bmp

tail -c +55 cbc.bmp > body_cbc
cat header body_cbc > fix_cbc.bmp

tail -c +55 cfb.bmp > body_cfb
cat header body_cfb > fix_cfb.bmp

tail -c +55 ofb.bmp > body_ofb
cat header body_ofb > fix_ofb.bmp
```

Cở bản việc chỉnh sửa này rất đơn giản, đầu tiên ta cần phải lưu 54 bytes đầu tiên của tệp ảnh gốc vào một file mới có tên là ***header*** bằng lệnh

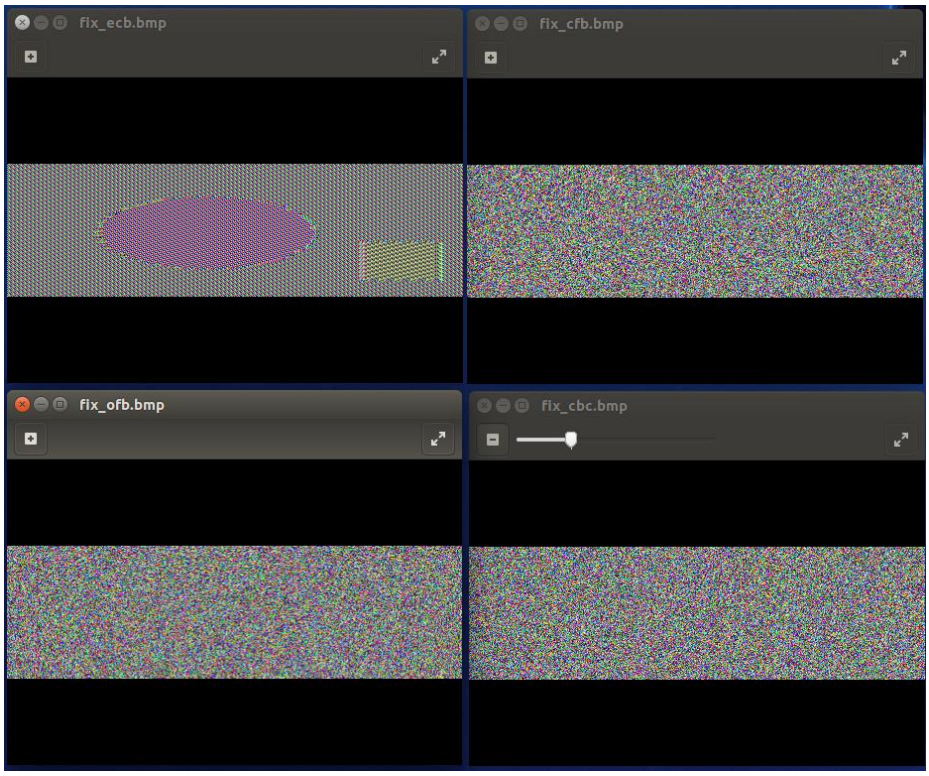
```
head -c 54 pic_original.bmp > header
```

Tiếp theo lấy các bytes còn lại của các tệp mã hoá và lưu vào file ***body_**** (với * là tên của chế độ mã hoá tương ứng) – tức chỉ lấy các bytes từ byte thứ 55 của tệp mã hoá bằng lệnh

```
tail -c +55 *.bmp > body_*
```

Sau đó thực hiện ghép 2 file này lại, ta sẽ được một file bmp hoàn chỉnh

```
cat header body_* > fix_cbc*.bmp
```



Nhận xét:

- Hình ảnh được mã hoá với chế độ ECB sẽ bị nhiễu chút ít so với hình ảnh gốc và người quan sát có thể ước lượng được bố cục của hình ảnh gốc. ECB là một chế độ tiêu chuẩn, bản rõ/bản mã ban đầu được phân ra thành các khối, mỗi khối sẽ được mã hoá/giải mã cùng với một khoá, các khối là hoàn toàn độc lập với nhau cho nên khi một khối bị hỏng thì nó sẽ không làm ảnh hưởng đến các khối khác và hình ảnh vẫn được đảm bảo người quan sát có thể nhìn ra được.
- Hình ảnh được mã hoá với chế độ CBC, CFB, OFB dường như bị nhiễu hoàn toàn và người quan sát không thể nào có thể ước lượng được bố cục của hình ảnh gốc. Các thuật toán trên ngoài việc mỗi khối sử dụng khoá ra thì nó còn sử dụng vector khởi tạo (IV). IV có cùng kích thước với khối được mã hoá/giải mã và thường là một số ngẫu nhiên. Bởi vì có sử dụng thêm IV sẽ làm tăng tính an toàn cho dữ liệu mã hoá. Mỗi khối sẽ được tiến hành thực hiện phép XOR với IV rồi tiến hành mã hoá/giải mã, đầu ra kết quả của việc giải mã trên sẽ được sử dụng cho khối tiếp theo nó vì vậy khi một khối mã bị hỏng thì nó sẽ ảnh hưởng đến các khối mã khác (mức ảnh hưởng sẽ tuỳ thuộc vào từng chế độ).

IV. Các vấn đề liên quan

IV.1 Tìm hiểu về Padding

Ở nội dung này để nắm bắt được các mục tiêu:

- Sự khác nhau về Padding của tệp khi thực hiện mã hoá ở các chế độ mã hoá ECB, CBC, CFB, OFB
- Quan sát các giá trị của Padding khi thực hiện mã hoá

(1) Mục tiêu đầu tiên

Ở đây tụi em sẽ tạo nên file plaintext có kích thước là 6bytes với câu lệnh sau (prefix -n giúp xoá đi khoảng trống đằng sau chuỗi):

```
$ echo -n “6bytes” > plaintext.txt
```

Sau đó ta sẽ thực hiện mã hoá plaintext.txt với thuật toán AES128 ở các chế độ bằng file zenc.sh có nội dung nhu sau:

```
#!/bin/bash
openssl enc -aes-128-ecb -e -in plaintext.txt -out enc_ecb.txt -K
00112233445566778889aabbccddeeff

openssl enc -aes-128-cbc -e -in plaintext.txt -out enc_cbc.txt -K
00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-cfb -e -in plaintext.txt -out enc_cfb.txt -K
00112233445566778889aabbccddeeff -iv 0102030405060708
```

```
openssl enc -aes-128-ofb -e -in plaintext.txt -out enc_ofb.txt -K
00112233445566778889aabbccddeeff -iv 0102030405060708
```

Đây là kết quả khi thực hiện mã hoá, ta sử dụng lệnh ***ls -l*** để nhìn thấy được kích thước của các tập tin sau khi mã hoá:

- Ở chế độ CFB,OFB các tệp tin có kích thước như plaintext (6 bytes)
- Ở chế độ ECB,CBC các tệp tin có kích thước lớn hơn plaintext là 10bytes (16 bytes)

```
[12/10/21]seed@Server:~/.../padding$ zenc.sh
[12/10/21]seed@Server:~/.../padding$ ls -l
total 24
-rw-rw-r-- 1 seed seed  16 Dec 10 16:26 enc_cbc.txt
-rw-rw-r-- 1 seed seed   6 Dec 10 16:26 enc_cfb.txt
-rw-rw-r-- 1 seed seed  16 Dec 10 16:26 enc_ecb.txt
-rw-rw-r-- 1 seed seed   6 Dec 10 16:26 enc_ofb.txt
-rw-rw-r-- 1 seed seed   6 Dec 10 16:25 plaintext.txt
```

Sau khi thực hiện mã hoá thì ta có thể thấy rằng

- Chế độ OFB và CFB sẽ không có đệm vì chế độ OFB và CFB cho phép các khối mã hoá được sử dụng như là hệ mã dòng (Stream Cipher) do đó bản rõ không cần phải là bội số của kích thước khối, các chế độ này sẽ không mã hoá trực tiếp bản rõ mà chỉ sử dụng bản mã để XOR với bản rõ để lấy bản mã nên nó không cần phải chèn thêm dữ liệu.
- Chế độ CBC và ECB có thực hiện việc thêm đệm vì các chế độ này bản rõ sẽ được chia thành các khối mã (Block Cipher) với kích thước cố định (ví dự như 64 hay 128 bit), nếu như khối không đủ kích thước thì nó sẽ tiến hành chèn thêm đệm vào để đảm bảo rằng mọi người đều có kích thước như nhau. Sau khi thực hiện giải mã thì phần đệm cũng sẽ được loại bỏ để đảm bảo nội dung y như lúc ban đầu.

(4) Mục tiêu thứ 2

Ở nội dung này, tụi em sẽ tạo ra 3 file 5bytes.txt, 10bytes.txt, 16bytes.txt cos kích thước lần lượt là 5, 10, 16 bytes

```
[12/10/21]seed@Server:~/.../padding$ echo -n "12345" > 5bytes.txt
[12/10/21]seed@Server:~/.../padding$ echo -n "1234512345" > 10bytes.txt
[12/10/21]seed@Server:~/.../padding$ echo -n "1234567890abcdef" > 16bytes.txt
[12/10/21]seed@Server:~/.../padding$ ls -l
total 24
-rw-rw-r-- 1 seed seed  10 Dec 10 17:07 10bytes.txt
-rw-rw-r-- 1 seed seed  16 Dec 10 17:07 16bytes.txt
-rw-rw-r-- 1 seed seed   5 Dec 10 17:07 5bytes.txt
```

Thực hiện mã hoá AES với chế độ CBC cho các tập tin trên bằng file ***enc.sh*** có nội dung như sau

```
#!/bin/bash
openssl enc -aes-128-cbc -e -in 5bytes.txt -out enc_5bytes.txt -K
00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cbc -e -in 10bytes.txt -out enc_10bytes.txt
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

```
openssl enc -aes-128-cbc -e -in 16bytes.txt -out enc_16bytes.txt -
K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Thực hiện giải mã ba tập tin trên bằng file ***dec.sh***, ở đây cần thêm prefix -nopad để có thể vô hiệu hoá việc xoá các dữ liệu padding khi thực hiện giải mã. Chỉ khi sử dụng prefix này ta mới quan sát được các giá trị của padding của các tệp. Nội dung của file như sau:

```
#!/bin/bash
openssl enc -aes-128-cbc -d  -in enc_5bytes.txt  -out
dec_5bytes.txt  -K 00112233445566778889aabbccddeeff -iv
0102030405060708 -nopad
openssl enc -aes-128-cbc -d  -in enc_10bytes.txt -out
dec_10bytes.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708 -nopad
openssl enc -aes-128-cbc -d  -in enc_16bytes.txt -out
dec_16bytes.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708 -nopad
```

Các giá trị padding sẽ không in ra được cùng với dữ liệu của tệp, nên ta sẽ sử dụng hexdump để in ra chúng. Ở đây, bash scripts ***hexdump.sh*** sẽ xử lý việc này

```
#!/bin/bash
echo "dec_5bytes.txt"
hexdump -C dec_5bytes.txt
echo "dec_10bytes.txt"
hexdump -C dec_10bytes.txt
echo "dec_16bytes.txt"
hexdump -C dec_16bytes.txt
```

Và đây là kết quả sau khi thực hiện:

```
[12/10/21]seed@Server:~/.../padding$ enc.sh
[12/10/21]seed@Server:~/.../padding$ dec.sh
[12/10/21]seed@Server:~/.../padding$ hexdump.sh
dec_5bytes.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
dec_10bytes.txt
00000000  31 32 33 34 35 31 32 33  34 35 06 06 06 06 06 06  |1234512345......|
00000010
dec_16bytes.txt
00000000  31 32 33 34 35 36 37 38  39 30 61 62 63 64 65 66  |1234567890abcdef|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
```

Quan sát được các giá trị Padding sẽ là hiệu của kích thước của 1 block (16bytes) với kích thước của block plaintext (16 - 5=11=> hex = 0x0b,16 – 10 = 6 => hex = 0x06). Còn nếu đã đủ 1 block thì Padding sẽ được thêm cho toàn bộ block kế tiếp với giá trị là 0x10 (16 bytes, kích thước của 1 block)

IV.2 Lỗi lan truyền – dữ liệu mã hoá bị tổn hại

Nhiệm vụ của chúng ta ở phần này sẽ là thay đổi giá trị của các bytes ở các tệp đã mã hoá rồi giải mã các tệp đã chỉnh sửa để so sánh với plaintext ban đầu để xem sự thay đổi.

Đầu tiên ta tạo một tập tin văn bản plaintext.txt với kích thước 1000 bytes, có nội dung:

```
THE OSCARS TURN  ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE AWARDS TRIP THE
BAGGER FEELS LIKE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET AND THE
APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY THE EMERGENCE OF
METOO TIMES UP BLACKGOWN POLITICS ARMCANDY ACTIVISM AND A NATIONAL CONVERSATION AS BRIEF
AND MAD AS A FEVER DREAM ABOUT WHETHER THERE OUGHT TO BE A PRESIDENT WINFREY THE SEASON
DIDNT ZUST SEEM EXTRA LONG IT WAS EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST
WEEKEND IN MARCH TO AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS
THANKS PYEONGCHANG

ONE BIG QUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE CEREMONY WILL
ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME A ZUBILANT COMINGOUT PARTY
FOR TIMES UP THE MOVEMENT SPEARHEADED BY  POWERFUL HOLLYWOOD WOMEN WHO HELPED RAISE MILLIONS
OF DOLLARS TO FIGHT SEXUAL HARASSMENT AROUND THE COUNTRY
```

Tiến hành mã hoá tập tin trên với các chế độ ECB, CBC, CFB và OFB

```
$ openssl enc -aes-128-ecb -in plaintext.txt -out ecb_enc.txt -K
00112233445566778899aabbccddeeff
$ openssl enc -aes-128-cbc -in plaintext.txt -out cbc_enc.txt -K
00112233445566778899aabbccddeeff -iv 0102030405060708
$ openssl enc -aes-128-cfb -in plaintext.txt -out cfb_enc.txt -K
00112233445566778899aabbccddeeff -iv 0102030405060708
$ openssl enc -aes-128-ofb -in plaintext.txt -out ofb_enc.txt -K
00112233445566778899aabbccddeeff -iv 0102030405060708
```

Sau khi mã hoá, ta được các tệp tin như sau:

```
[12/10/21]seed@Server:~/.../CIPHERTEXT$ ls -la
total 24
drwxrwxr-x 2 seed seed 4096 Dec 10 05:08 .
drwxrwxr-x 5 seed seed 4096 Dec 10 05:08 ..
-rw-rw-r-- 1 seed seed 1008 Dec 10 03:29 cbc_enc.txt
-rw-rw-r-- 1 seed seed 1000 Dec 10 03:29 cfb_enc.txt
-rw-rw-r-- 1 seed seed 1008 Dec 10 03:29 ecb_enc.txt
-rw-rw-r-- 1 seed seed 1000 Dec 10 03:30 ofb_enc.txt
```

Em sẽ sử dụng công cụ Ghex để xem vị trí byte thứ 56 của tập tin đã bị mã hoá sau đó tiến hành sửa đổi giá trị tại vị trí này thành giá trị “AA”. Sau khi đã sửa đổi nội dung tại byte thứ 56 thì ta tiến hành giải mã các tập tin bị mã hoá và quan sát kết quả:

```
$ openssl enc -d -aes-128-ecb -in CIPHERTEXT/ecb_enc.txt -out ecb_dec.txt
-K 00112233445566778899aabbccddeeff
$ openssl enc -d -aes-128-cbc -in CIPHERTEXT/cbc_enc.txt -out cbc_dec.txt
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
$ openssl enc -d -aes-128-cfb -in CIPHERTEXT/cfb_enc.txt -out cfb_dec.txt
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
$ openssl enc -d -aes-128-ofb -in CIPHERTEXT/ofb_enc.txt -out ofb_dec.txt
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

```
[12/10/21]seed@Server:~/.../PLAINTEXT_AFTER_MODIFY$ ls -la
total 24
drwxrwxr-x 2 seed seed 4096 Dec 10 05:08 .
drwxrwxr-x 5 seed seed 4096 Dec 10 05:08 ..
-rw-rw-r-- 1 seed seed 1000 Dec 10 05:08 cbc_dec.txt
-rw-rw-r-- 1 seed seed 1000 Dec 10 05:08 cfb_dec.txt
-rw-rw-r-- 1 seed seed 1000 Dec 10 05:08 ecb_dec.txt
-rw-rw-r-- 1 seed seed 1000 Dec 10 05:08 ofb_dec.txt
```

(1) Quan sát đối với chế độ ECB

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 FF 44 78 A7 E3 0F 13 E2 0C 79 F3 92 F2 E7 06 00 .Dx......y......
00000040 4E 47 20 53 54 52 41 4E 47 45 0A 41 57 41 52 44 NG STRANGE.AWARD
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

(5) Quan sát đối với chế độ CBC

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 C0 EF A1 98 6C 2F F8 FE FB 87 90 C4 C0 6F A0 7E ....l/.......o.~
00000040 4E 47 20 53 54 52 41 65 47 45 0A 41 57 41 52 44 NG STRAeGE.AWARD
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 C0 EF A1 98 6C 2F F8 FE FB 87 90 C4 C0 6F A0 7E ....l/.......o.~
00000040 4E 47 20 53 54 52 41 65 47 45 0A 41 57 41 52 44 NG STRAeGE.AWARD
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

(6) Quan sát đối với chế độ CFB

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 48 54 20 41 46 54 45 0D 20 54 48 49 53 20 4C 4F HT AFTE. THIS LO
00000040 D3 61 A2 09 E0 57 37 B1 0E 1D FC 69 6D 9D 7B C7 .a...W7....im.{.
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 48 54 20 41 46 54 45 0D 20 54 48 49 53 20 4C 4F HT AFTE. THIS LO
00000040 D3 61 A2 09 E0 57 37 B1 0E 1D FC 69 6D 9D 7B C7 .a...W7....im.{.
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

(7) Quan sát đối với chế độ OFB

```
00000000 54 48 45 20 4F 53 43 41 52 53 20 54 55 52 4E 20 THE OSCARS TURN
00000010 20 4F 4E 20 53 55 4E 44 41 59 20 57 48 49 43 48  ON SUNDAY WHICH
00000020 20 53 45 45 4D 53 20 41 42 4F 55 54 20 52 49 47  SEEMS ABOUT RIG
00000030 48 54 20 41 46 54 45 73 20 54 48 49 53 20 4C 4F HT AFTEs THIS LO
00000040 4E 47 20 53 54 52 41 4E 47 45 0A 41 57 41 52 44 NG STRANGE.AWARD
00000050 53 20 54 52 49 50 20 54 48 45 20 42 41 47 47 45 S TRIP THE BAGGE
00000060 52 20 46 45 45 4C 53 20 4C 49 4B 45 20 41 20 4E R FEELS LIKE A N
00000070 4F 4E 41 47 45 4E 41 52 49 41 4E 20 54 4F 4F 0A ONAGENARIAN TOO.
00000080 0A 54 48 45 20 41 57 41 52 44 53 20 52 41 43 45 .THE AWARDS RACE
00000090 20 57 41 53 20 42 4F 4F 4B 45 4E 44 45 44 20 42  WAS BOOKENDED B
```

Ở đây chúng ta có thể viết một chương trình kiểm tra số byte khác so với bản rõ plaintext ban đầu. Đoạn chương trình như sau:

```python
#!/usr/bin/python3
from sys import argv
_, first, second = argv
with open(first, 'rb') as f:
    f1 = f.read()
with open(second, 'rb') as f:
    f2 = f.read()
res = 0
for i in range(min(len(f1), len(f2))):
    if f1[i] != f2[i]:
        res += 1
print("diff bytes: "+str(res+abs(len(f1)-len(f2))))
```

Tiến hành chạy chương trình để xem có bao nhiêu byte bị hỏng đối với từng chế độ

```
[12/10/21]seed@Server:~/.../propagation$ diff.sh
diff bytes: 16
diff bytes: 17
diff bytes: 17
diff bytes: 1
```

Nhận xét:

- Đối với chế độ ECB: Chỉ có các byte từ thứ 49 đến byte thứ 64 (Offset 0x30 đến 0x3F) thuộc khối thứ 4 sẽ bị hỏng, còn các byte và các khối khác không bị ảnh hưởng gì.
- Đối với chế độ CBC: Các byte từ thứ 49 đến byte thứ 64 (Offset 0x30 đến 0x3F) thuộc khối thứ 4 và byte thứ 72 (Offset 0x47) thuộc khối thứ 5 bị hỏng, còn các byte và các khối khác không bị ảnh hưởng gì.
- Đối với chế độ CFB: Byte thứ 56 (Offset 0x37) thuộc khối thứ 4 và các byte thứ 65 đến byte thứ 80 (Offset 0x40 đến 0x4F) thuộc khối thứ 5 bị hỏng, còn các byte và các khối khác không bị ảnh hưởng gì.
- Đối với chế độ OFB: Chỉ có byte thứ 56 (Offset 0x37) thuộc khối thứ 4 mà chúng ta đã chính sửa là bị hỏng, còn các byte và các khối khác không bị ảnh hưởng gì

Phân tích kết quả sau khi quan sát được:

- Chế độ ECB: Các khối trong ECB được mã hoá/giải mã hoàn toàn độc lập với nhau cho nên khi thay đổi 1 bit của byte trong 1 khối bất kỳ thì chỉ có khối duy nhất có byte bị thay đổi sẽ bị hỏng, các khối khác không bị ảnh hưởng.

- Chế độ CBC: Do 1 bit của byte thứ 56 bị thay đổi nên khối chưa byte có bit bị thay đổi sẽ hỏng, đồng thời trong khối tiếp theo có byte ở vị trí tương ứng ở đây là byte thứ 72 so với byte thứ 56 cũng sẽ bị hỏng Do mỗi khối plaintext được XOR với khối ciphertext trước khi được mã hoá nên mỗi khối ciphertext phụ thuộc vào tất cả plaintext xuất hiện từ đầu đến thời điểm đó.
- Chế độ CFB: Tương tự như CBC, khi thực hiện thay đổi 1 bit của byte thứ 56 thì ngoài việc byte thứ 56 bị hỏng nó sẽ còn ảnh hưởng đến khối tiếp theo.
- Chế độ OFB: Do 1 bit của byte thứ 56 bị hỏng nên khi khối mã hoá được XOR với bản mã thì chỉ có 1 bit trong byte thứ 56 của khối đó bị hỏng, các byte khác trong khối cũng như các khối khác cũng không bị ảnh hưởng gì.

IV.3 IV và những sai lầm phổ biến

3.1 Thử nghiệm sử dụng IV

Đầu tiên tụi em sẽ tạo một tập tin plaintext.txt với nội dung “an toan thong tin” và tiến hành mã hoá tập tin bằng thuật toán AES128 kết hợp với chế độ CBC cho ra 2 tập tin same_iv_I.txt và same_iv_II.txt.

```
[12/10/21]seed@Server:~/.../1$ ls
plaintext.txt
[12/10/21]seed@Server:~/.../1$ echo -n "an toan thong tin" > plaintext.txt
[12/10/21]seed@Server:~/.../1$ openssl enc -aes-128-cbc -e -in plaintext.txt -out same_iv_I.txt
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
[12/10/21]seed@Server:~/.../1$ openssl enc -aes-128-cbc -e -in plaintext.txt -out same_iv_II.txt
 -K 00112233445566778899aabbccddeeff -iv 0102030405060708
```

Quan sát kết quả tập tin sau khi mã hoá có cùng IV thì ta có thể thấy nội dung đã mã hoá của cả hai tập tin là như nhau:

```
00000000 C4 10 05 00 31 6F 67 5F A6 6E 62 9B 0B F0 DC 73  ....1og_.nb....s
00000010 0D 5B 99 A1 75 1D F6 05 40 C7 34 A0 FB FB A4 30  .[..u...@.4....0
```

Nội dung của tệp tin same_iv_I.txt

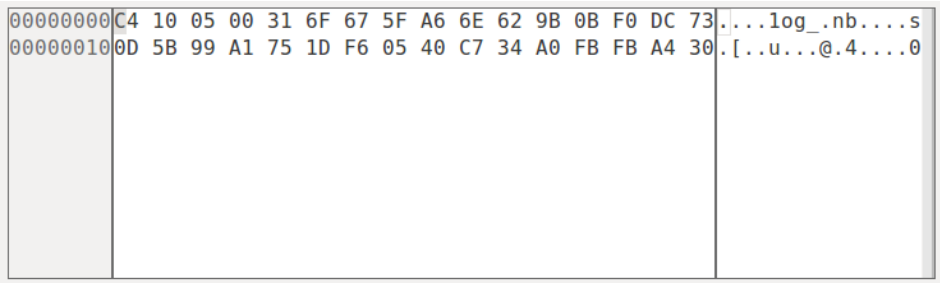
```
00000000 C4 10 05 00 31 6F 67 5F A6 6E 62 9B 0B F0 DC 73  ....1og_.nb....s
00000010 0D 5B 99 A1 75 1D F6 05 40 C7 34 A0 FB FB A4 30  .[..u...@.4....0
```

Nội dung của tệp tin same_iv_II.txt

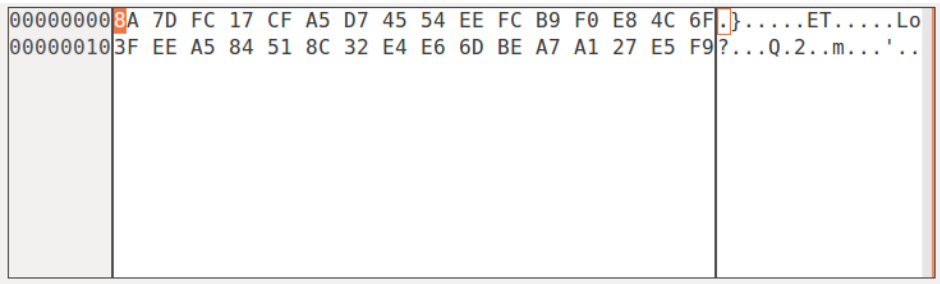
Tiếp tục thực hiện mã hoá với trường hợp khác IV

```
[12/10/21]seed@Server:~/.../1$ openssl enc -aes-128-cbc -e -in plaintext.txt -out diff_iv_I.txt
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
[12/10/21]seed@Server:~/.../1$ openssl enc -aes-128-cbc -e -in plaintext.txt -out diff_iv_II.txt
 -K 00112233445566778899aabbccddeeff -iv 0102030405060709
```

Quan sát tập tin sau khi mã hoá với IV khác nhau thì ta có thể thấy nội dung đã mã hoá của hai tập tin là khác nhau



Nội dung của tệp tin diff_iv_I.txt



Nội dung của tệp tin diff_iv_II.txt

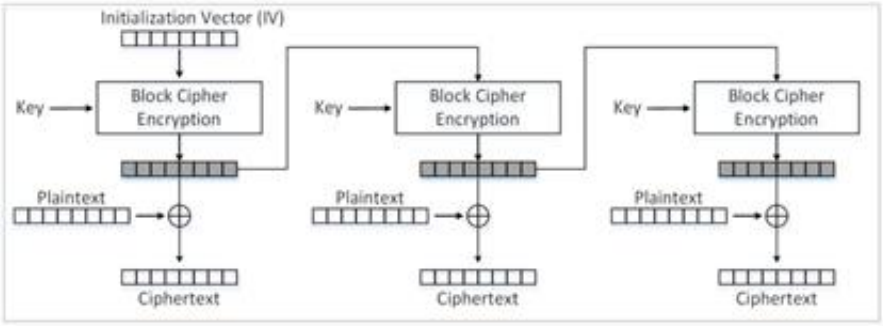
Kết luận

- Việc sử dụng cùng một IV trên cùng một bản rõ đã dẫn đến kết quả đầu ra giống hệt nhau
- Việc sử dụng khác IV trên cùng một bản rõ dẫn đến kết quả đầu ra khác nhau.
- IV cần là phải là duy nhất bởi vì nếu chúng ta sử dụng bản rõ nhiều hơn một lần, việc không có IV duy nhất mỗi lần với cùng một khoá sẽ dẫn đến một bản mã giống hệt nhau, từ đó mà kẻ tấn công có thể thu thập và phân tích ra nội dung bản rõ.

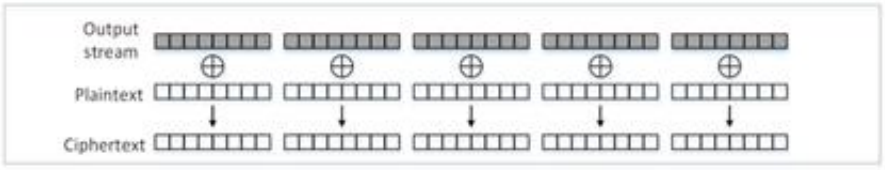
3.2 Sai lầm đầu tiên: Sử dụng cùng một IV

Đối với chế độ OFB, nếu khoá và IV không thay đổi thì việc tấn công bằng kỹ thuật known-plaintext có khả thi.

Đầu ra có thể thu được bằng bản rõ XOR với bản mã theo khối. Tương tự, để có được bản rõ, chúng ta có thể XOR bản rõ và bản mã. Khi sử dụng cùng một khoá và IV cho chế độ OFB thì đầu ra giống hết nhau.



(a) Output Feedback (OFB) mode encryption



(b) XOR the plaintext with the output stream

Giả sử rằng chúng ta biết một bản rõ P1 và bản mã OFB của nó là C1. Một bản mã OFB khác C2 có cùng khoá và IV. Nhưng chúng ta chưa biết bản rõ P2 của C2. Bằng cách nhìn vào hai bản mã C1 và C2 thì ta có thể đoán ký tự kết thúc của P2 cũng là một dấu "!".

```
Plaintext (P1): This is a known message!
Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
Plaintext (P2): (unknown to you)
Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903f
```

Đầu tiên ta thực lấy luồng kết quả mã hoá đầu ra của bản rõ P1

- Output_stream = P1 XOR C1

Bản rõ P2 được xác định:

- P2 = Output_stream XOR C2

Ta có thể viết ngắn ngọn lại thành:

- P2 = P1 XOR C1 XOR C2

Bây giờ ta sẽ viết một chương trình tấn công để tìm bản rõ P2

```python
#!/usr/bin/python3
from sys import argv

_, first, second, third = argv
p1 = bytearray(first,encoding='utf-8')
c1 = bytearray.fromhex(second)
c2 = bytearray.fromhex(third)
p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, c1, c2))

f = open("plaintext.txt", "w")
f.write(p2.decode('utf-8'))
```

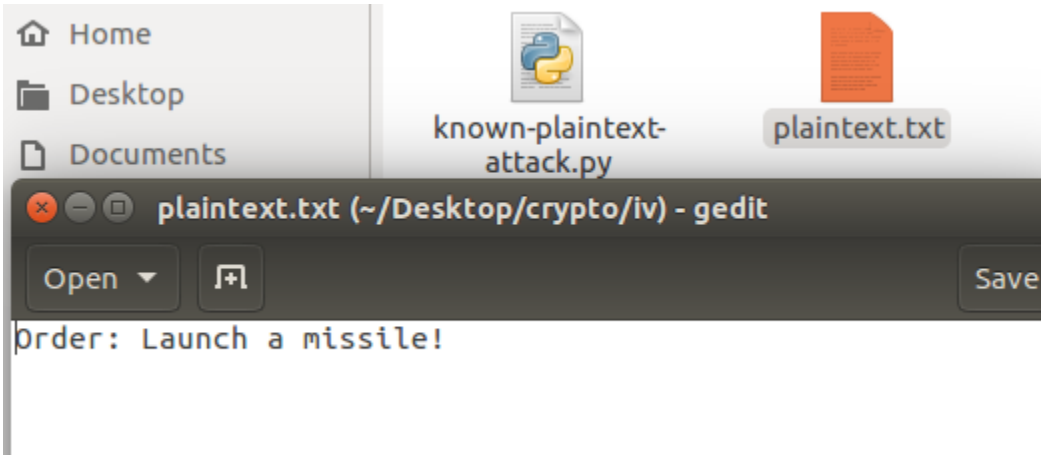
```
f.close()
```

Chương trình known-plaintext-attack.py

Kết quả sau khi thực hiện chương trình

```
[12/10/21]seed@Server:~/.../iv$ known-plaintext-attack.py "This is
a known message!" a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
[12/10/21]seed@Server:~/.../iv$
```

Bản rõ P2 tìm được là "Order: Launch a missible!"



Nội dung của P2

Với trường hợp chế độ CFB, kẻ tấn công chỉ biết khối đầu tiên của bản rõ mà thôi.

3.3 Sai lầm thứ hai: Sử dụng IV có thể dự đoán trước

Từ các nhiệm vụ trước, chúng ta biết rằng các IV không thể sử dụng lại một cách hoàn toàn. Một yêu cầu quan trọng khác đối với IV là tính không thể đoán trước được, tức là IV cần được tạo ra một cách ngẫu nhiên. Trong nhiệm vụ này, chúng ta sẽ tìm hiểu xem điều gì sẽ xảy ra nếu IV có thể dự đoán được.

Giả sử rằng Bob vừa gửi một tin nhắn được mã hóa và Eve biết rằng nội dung của nó là "Yes" hoặc "No"; Eve có thể nhìn thấy ciphertext và IV được sử dụng để mã hóa tin nhắn, nhưng vì thuật toán mã hóa AES khá mạnh nên Eve không biết nội dung thực sự là gì. Tuy nhiên, vì Bob sử dụng một IV có thể dự đoán trước (trong trường hợp này $IV_{sau} = IV_{trước} + 1$), nên Eve biết chính xác IV tiếp theo mà Bob sẽ sử dụng. Sau đây là cái nhìn tổng quan những gì Bob và Eve biết:

```
Key (in hex): 00112233445566778899aabbccddeeff (Mình Bob biết)
Ciphertext (C1): bef65565572ccee2a9f9553154ed9498 (Cả 2 biết)
IV used on P1 (Cả 2 biết)
      (in ascii): 1234567890123456
      (in hex)  : 31323334353637383930313233343536
Next IV (Cả 2 biết)
      (in ascii): 1234567890123457
      (in hex)  : 31323334353637383930313233343537
```

Một mật mã được coi là tốt, khi:

(1) Chịu được cuộc tấn công ***know-plaintext attack*** được mô tả trước đó

(2) Mà còn phải chịu đựng cuộc tấn ***chosen-plaintext attack***, đây là mô hình tấn công - phân tích mật mã trong đó kẻ tấn công có thể lấy được ciphertext cho một plaintext tùy ý.

Vì AES là thuật toán mạnh nên có thể chịu được cuộc tấn công ***know-plaintext attack***, Bob không ngại mã hoá bất kỳ bản rõ nào do Eve đưa ra; anh ta sử dụng một IV khác nhau cho mỗi bản rõ, nhưng thật không may, IV mà anh ta tạo ra không phải ngẫu nhiên mà chúng luôn có thể dự đoán được.

Công việc của em sẽ xây dựng một message `P2` và yêu cầu Bob mã hoá nó để cung cấp cho mình plaintext. Mục tiêu của tụi em là sử dụng cơ hội này để tìm hiểu xem nội dung thực sự bên trong tin nhắn của Bob (`P1`) là "`Yes`" hay "`No`" .

Đối với nhiệm vụ này, tụi em được cấp một cuộc mô phỏng mã hoá của Bob và việc mã hoá này sử dụng thuật toán AES128 với chế độ CBC.

Để thực hiện nhiệm vụ:

(1) Tụi em sẽ giả sử `P1` là "`Yes`".

(2) Cho nên tụi em xây dựng được P2 bằng công thức nhu sau:

`P2 = "Yes" ⊕ IV ⊕ IV_Next`

(3) Ở đây tụi em có xây dựng một ứng dụng để thực hiện tìm P2 dựa trên công thức trên

```
#!/usr/bin/python3
from sys import argv

_, first, second, third = argv
p1 = bytearray(first, encoding='utf-8')
padding = 16 - len(p1) % 16  # thực hiện đệm thêm vào block để đủ
128bit
p1.extend([padding]*padding)
IV = bytearray.fromhex(second)
IV_NEXT = bytearray.fromhex(third)
p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, IV, IV_NEXT))
print(p2.decode('utf-8'), end='')
```

Thực hiện chạy chương trình trên ta được kết quả P2 là Yes

```
[12/10/21]seed@Server:~/.../3$ cipher_cons.py "Yes" 313233343536373
83930313233343536 31323334353637383930313233343537 > p2
```

```
[12/10/21]seed@Server:~/.../3$ cat p2
Yes
```

Tiến hành mã hoá P2 để xác định được C2. Vì đây là phương thức AES-128-CBC nên nó cần được chèn thêm đệm trước khi thực hiện mã hoá. Để so sánh với C1 thực tế, chúng ta chỉ cần khối đầu tiên của C2

```
[12/10/21]seed@Server:~/.../3$ openssl enc  -aes-128-cbc  -e -in p2
 -out c2 -K 00112233445566778899aabbccddeeff -iv 313233343536373839
30313233343537
```

```
[12/10/21]seed@Server:~/.../3$ xxd -p c2
bef65565572ccee2a9f9553154ed94983402de3f0dd16ce789e5475779ac
a405
```

Ta có thể thấy rằng 16 bytes (128 bit) của C2 giống với C1 mà đề cho nên ta có thể kết luận dự đoán ban đầu mà ta đưa ra là C1 có nội dung là “Yes” là chính xác. Để chắc chắn ta lấy 128 bit đầu của C2 cho nội dung của C1. Sau đó tiến hành mã hoá C1 ra để tìm nội dung của P1.

```
[12/10/21]seed@Server:~/.../3$ echo -n "bef65565572ccee2a9f9553154e
d9498" | xxd -r -p > c1
[12/10/21]seed@Server:~/.../3$ openssl enc  -aes-128-cbc  -d -in c1
 -out p1 -K 00112233445566778899aabbccddeeff -iv 313233343536373839
30313233343536
[12/10/21]seed@Server:~/.../3$ cat p1
Yes[12/10/21]seed@Server:~/.../3$ S
```

Vậy là kết quả P1 có nội dung là “Yes”

TÀI LIỆU THAM KHẢO

(1) Chương 21 Secrect-key Encryption của Sách Computer Internet Security A Hands-on Approach by Wenliang Du
(2) Chương 20 Symmetric Encryption and Message Confidentiality của sách Computer security Principles and Practice Third Edition
(3) Block cipher mode of operation - Block cipher mode of operation - Wikipedia