Ct030440 truong quoc quan co so an toan va bao mat thong tin 3249

Cryptography and Security Information (Đại học Kinh tế Quốc dân)

HỌC VIỆN KỸ THUẬT MẬT MÃ
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP MÔN HỌC

Viết chương trình mã hóa và giải mã bằng mật mã AES

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm nhúng và di động

Người hướng dẫn: **TS. Nguyễn Đào Trường**

Khoa Công nghệ thông tin – Học viện Kỹ thuật mật mã

Sinh viên thực hiên:
Trương Quốc Quân

Hà Nội, 2021

MỤC LỤC

LỜI NÓI ĐẦU

Mật mã học và các thành tựu của nó là một lĩnh vực quan trọng trong quá trình phát triển của nhân loại. Với mục tiêu làm rối và làm loạn thông tin, rồi sau đó có thể tái tạo lại thông tin một cách chân thực, chúng ta có thể đảm bảo được tính bí mật của thông tin, có thể truyền thông tin đi xa, và dựa vào các đặc tính của quá trình mã hóa, như tính không thể chối bỏ, tìm ra được các lỗ hổng bảo mật. Nhu cầu được mã hóa, bảo vệ thông tin, giao nhận thông tin đến đúng người nhận có ý nghĩa thực tiễn, từ chiến tranh đến hòa bình, từ công cuộc bảo vệ và xây dựng đất nước. Thật vậy, ngành mật mã học xuất phát từ thời La Mã, khi tướng Caesar đã mã hóa thông tin bằng cách dịch chữ cái, hay mật mã Scytale của người Spartan, và giờ đây, thông tin nhạy cảm của ta như dữ liệu đăng nhập, dữ liệu thẻ tín dụng, dữ liệu trong chip CMND,… đều cần đến các thành tựu của ngành này. Việc tìm ra một thuật toán mã hóa an toàn khỏi sự can thiệp bên ngoài tưởng chừng đã kết thúc với chiếc chén thánh của mật mã học – OTP, thế nhưng, các hạn chế của OTP bắt buộc chúng ra phải tìm ra các phương thức mới, để thích nghi với quá trình phát triển không ngừng của công nghệ. Thuật toán AES, ứng dụng từ mật mã Rijndael được sinh ra trong bối cảnh như vậy.

AES (viết tắt của từ tiếng Anh: Advanced Encryption Standard, hay Tiêu chuẩn mã hóa tiên tiến) là một thuật toán mã hóa khối được chính phủ Mỹ áp dụng làm tiêu chuẩn mã hóa. Thuật toán AES làm việc với các khối dữ liệu 128 bit và khóa độ dài là 128 bit, 192 bit và 256 bit. Mã hóa dùng AES là mã hóa khối lặp gồm nhiều chu trình, các khóa con sử dụng trong các chu trình được tạo ra bởi quá trình tạo khóa con Rijndael.

Trong bài báo cáo này chúng ta sẽ tìm hiểu về các chu trình làm việc của phương pháp mã hóa AES và ứng dụng viết một chương trình mã hóa và giải mã bằng AES sử dụng ngôn ngữ Python, rồi sau đó sẽ tìm hiểu các dạng tấn công vào AES và phương pháp phòng tránh.

Để hoàn thành bài báo cáo này, em xin gửi lời cám ơn chân thành đến ***TS. Nguyễn Đào Trường*** đã tận tình giúp đỡ và truyền đạt những kinh nghiệm quý báu trong suốt thời gian thực hiện.

Do hạn chế về thời gian nghiên cứu đề tài và kiến thức chuyên môn nên sẽ không tránh khỏi những thiếu sót, kính mong được sự góp ý từ Thầy và mọi người để hoàn thiện bài báo cáo tốt hơn!

I. TỔNG QUAN AES

1. Khái niệm từ (Word) trong AES

Bốn byte trên mỗi cột trong mảng trạng thái state tạo thành 1 từ 32 bit, trong đó số thứ tự của hàng r ($0 \le r < 4$) cho biết chỉ số của bốn byte trong mỗi từ. Từ định nghĩa state ở trên có thể coi state là mảng một chiều chứa các từ 32 bit

$$\square_0 = \square_{00}\ \square_{10}\ \square_{20}\ \square_{30}$$
$$\square_1 = \square_{01}\ \square_{11}\ \square_{21}\ \square_{31}$$
$$\square_2 = \square_{02}\ \square_{12}\ \square_{22}\ \square_{32}$$
$$\square_3 = \square_{03}\ \square_{13}\ \square_{23}\ \square_{33}$$

Tương tự như đối với mảng khóa cũng có thể biểu diễn thành mảng một chiều chứa các từ 32 bit như công thức dưới đây với số lượng từ khóa phụ thuộc vào Nk (Nk = 4, 6, 8).

2. Thuật toán của AES

Thuật toán AES khá phức tạp, được mô tả khái quát gồm 3 bước như sau:

- 1 Vòng khởi tạo chỉ gồm phép AddRoundKey
- Nr -1 Vòng lặp gồm 4 phép biến đổi lần lượt: SubBytes, ShiftRows, MixColumns, AddRoundKey.
- 1 Vòng cuối gồm các phép biến đổi giống vòng lặp và không có phép MixColumns.

3. Khái quát

1. Mở rộng khóa - Các khóa phụ dùng trong các vòng lặp được sinh ra từ khóa chính AES sử dụng thủ tục sinh khóa Rijndael.
2. Vòng tiên quyết: AddRoundKey
3. Các vòng lặp đến Nr – 1: SubBytes – ShiftRows - MixColumns – AddRoundKey
4. Vòng cuối (không MixColumns) – SubBytes – ShiftRows – AddRoundKey.

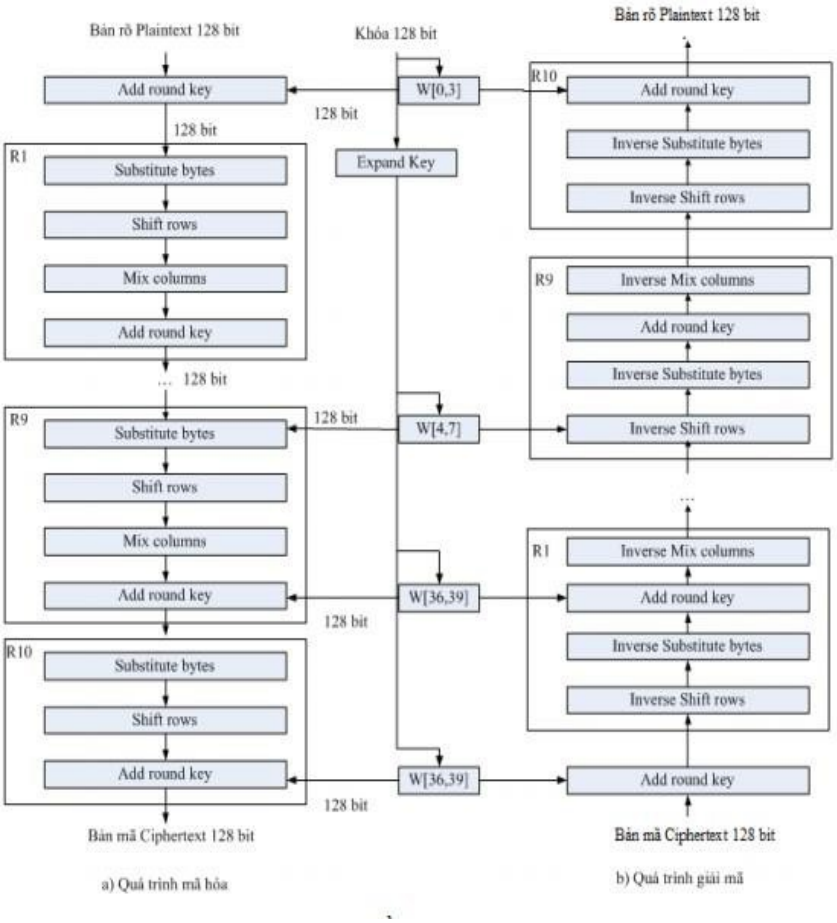
Trong đó:

AddRoundKey – Mỗi byte trong state được kết hợp với khóa phụ sử dụng XOR.

SubBytes - bước thay thế phi tuyến tính, trong đó mỗi byte trong state được thay thế bằng một byte khác sử dụng bảng tham chiếu

ShiftRows - bước đổi chỗ, trong đó mỗi dòng trong state được dịch một số bước theo chu kỳ

MixColumns - trộn các cột trong state, kết hợp 4 bytes trong mỗi cột



Các phép biến đổi Subbytes, ShiftRows, MixColumns có phép biến đổi ngược tương ứng là InvSubBytes, InvShiftRows, InvMixColumns. Riêng phép biến đổi AddRoundKey đơn giản chỉ là phép XOR nên phép biến đổi ngược cũng là AddRoundKey.

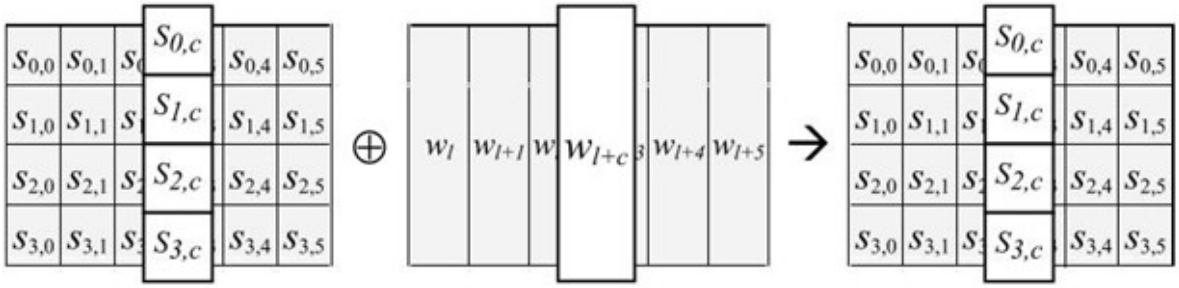
Vận dụng các phép biến đổi ngược trên, thuật toán giải mã AES cũng gồm 10 vòng thực hiện theo chiều ngược lại.

Kích thước khóa ban đầu là 128 bits, 192 bits hoặc 256 bits. AES dùng hàm keyExpansion để mở rộng kích thước khóa thành 44, 52 hoặc 60 words rồi được chia thành 11, 13 hoặc 15 cụm khóa con, mỗi khóa con 4 word làm khóa cho AddRoundKey.

II. MÃ HÓA

1. Phương thức AddRoundKey.

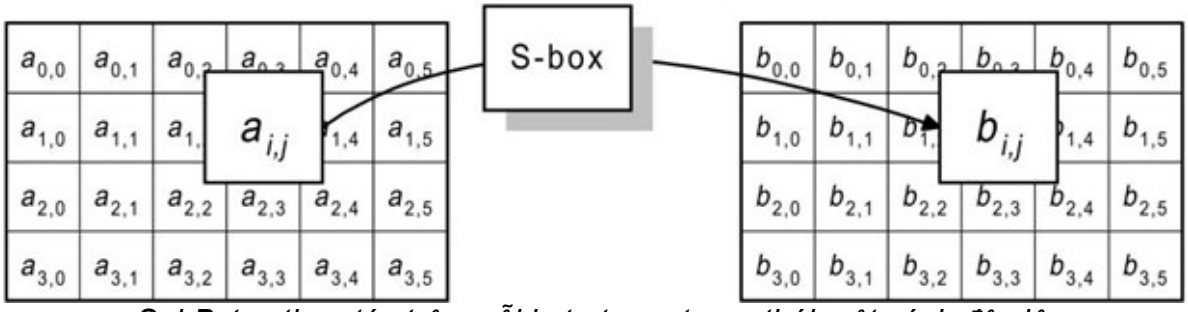
- Trong phép biến đổi này, Nb từ sẽ được thêm vào các cột của trạng thái được lấy từ bảng liệt kê khóa (đã được mô tả phía trên) sao cho:
  $[0 \quad 1\ 2\ 3] = [0\ 1\ 2\ 3] \oplus [\ _{t*} \quad _{+}]$; (với $0 < c < Nb$)
- Trong đó:
  - $[b_i]$ là từ trong bản liệt kê khóa
  - "round" là số vòng trong khoảng 1 round N_r. Số vòng bắt đầu từ 1 vì có một điều kiện về khóa khởi tạo trước hàm vòng.



Các từ trong bản liệt kê khóa được XOR với các cột trong trạng thái

2. Phương thức SubBytes.

- Đây là một phép thay thế byte không tuyến tính. Phép biến đổi này thao tác trên mỗi byte của trạng thái một cách độc lập để tạo ra một giá trị byte mới bằng cách sử dụng một bảng thay thế S-box.



SubBytes thao tác trên mỗi byte trong trạng thái một cách độc lập

hex		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Bảng S-Box

- Phép thay thế này có thể đảo ngược bằng cách sử dụng bảng Inverse Sbox, sử dụng hệt như bảng Sbox thường.

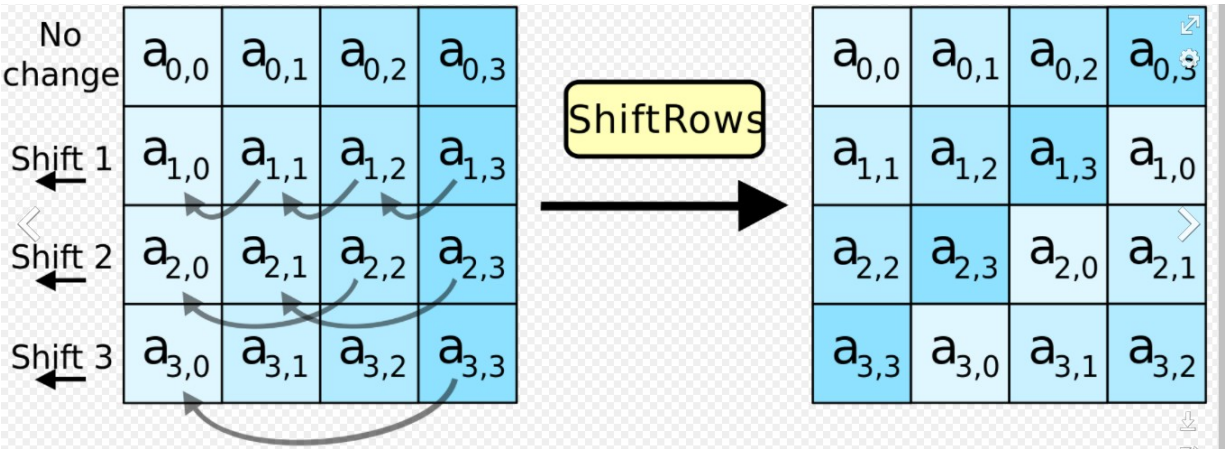
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Bảng Inverse Sbox

3. Phương thức ShiftRows

- Trong biến đổi ShiftRows(), các byte trong ba hàng cuối cùng của trạng thái được dịch vòng đi các số byte khác nhau (độ lệch). Cụ thể :
  - **S'r,c = Sr,(c + shift (r, Nb)) mod Nb (Nb = 4)**
- Trong đó giá trị dịch shift (r, Nb) phụ thuộc vào số hàng r như sau:
  - Shift(1,4) = 1, shift(2,4) = 2, shift(3,4) = 3.

- Hàng đầu tiên không bị dịch, ba hàng còn lại bị dịch tương ứng:
  - Hàng thứ 1 giữ nguyên.
  - Hàng thứ 2 dịch vòng trái 1 lần.
  - Hàng thứ 3 dịch vòng trái 2 lần.
  - Hàng thứ 4 dịch vòng trái 3 lần.



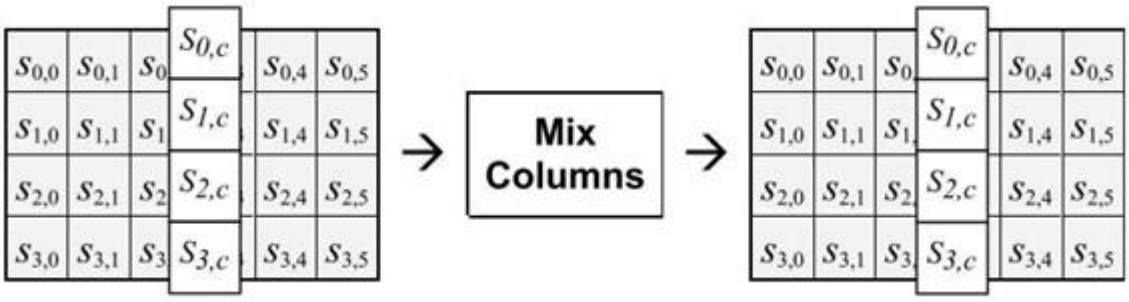
4. Phương thức MixColumns.

- Phép biến đổi này thao tác một cách độc lập trên mỗi cột của trạng thái và xem mỗi cột như một đa thức bậc 4.

- Ở dạng ma trận, phép biến đổi được dùng theo phương trình sau, với 0 ≤ □ < □ ▯

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- Trong đó tất cả giá trị là các phần tử thuộc trường hữu hạn.



MixColumns thao tác độc lập trên mỗi cột trong trạng thái

III. GIẢI MÃ

- Thuật toán giải mã khá giống với thuật toán mã hóa về mặt cấu trúc nhưng 4 hàm sử dụng là 4 hàm đảo ngược quá trình mã hóa.

- Để giải mã một bản mã được mã hóa AES, cần phải hoàn tác từng giai đoạn của hoạt động mã hóa theo thứ tự ngược lại mà chúng đã được áp dụng. Ba giai đoạn giải mã như sau:
  - Đảo ngược vòng cuối: *AddRoundKey – InvShiftRows – InvSubBytes*
  - Đảo ngược các vòng lặp: *AddRoundKey – InvMixColumns – InvShiftRows – InvSubBytes*
  - Đảo ngược vòng tiên quyết: *AddRoundKey*

- Trong số bốn hoạt động trong mã hóa AES, chỉ có phương thức AddRoundKey là nghịch đảo của chính nó (vì nó là exclusive-or). Để hoàn tác AddRoundKey, chỉ cần mở rộng toàn bộ lịch khóa AES (giống như mã hóa) và sau đó sử dụng khóa thích hợp theo chiều ngược với giai đoạn mã hóa.

- Hàm đảo ngược của SubBytes là invSubBytes, giống hệt như SubBytes, ngoại trừ việc sử dụng Sbox, ta sẽ sử dụng inverse Sbox

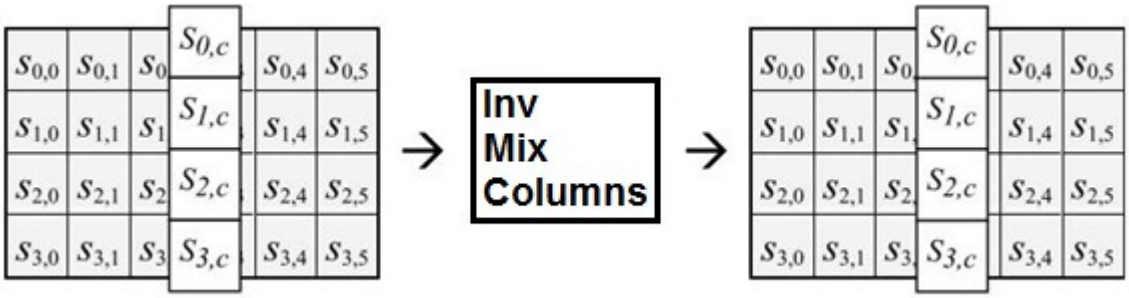
1. Phương thức invShiftRows
   - Trong biến đổi invShiftRows(), các byte trong ba hàng cuối cùng của trạng thái được dịch vòng đi các số byte khác nhau (độ lệch). Cụ thể :
     - **S'r,c = Sr,(c - shift (r, Nb)) mod Nb (Nb = 4)**
   - Trong đó giá trị dịch shift (r, Nb) phụ thuộc vào số hàng r như sau:
     - Shift(1,4) = 1, shift(2,4) = 2, shift(3,4) = 3.
   - Hàng đầu tiên không bị dịch, ba hàng còn lại bị dịch tương ứng:
     - Hàng thứ 1 giữ nguyên.
     - Hàng thứ 2 dịch vòng phải 1 lần.
     - Hàng thứ 3 dịch vòng phải 2 lần.
     - Hàng thứ 4 dịch vòng phải 3 lần

2. Phương thức InvMixColumns.
   - Phép biến đổi này thao tác một cách độc lập trên mỗi cột của trạng thái và xem mỗi cột như một đa thức bậc 4.

   - Ở dạng ma trận, phép biến đổi được dùng theo phương trình sau, với 0 ≤ □ < □□

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

- Trong đó tất cả giá trị là các phần tử thuộc trường hữu hạn.



InvMixColumns thao tác độc lập trên mỗi cột trong trạng thái

IV. THUẬT TOÁN MỞ RỘNG KHÓA keyExpansion

AES sử dụng thuật toán mở rộng khóa (thủ tục sinh khóa) để biến một khóa ngắn thành một bộ khóa vòng. AES-128, AES-192 và AES-256 sẽ có số lượng vòng khác nhau. Thủ tục sinh khóa sẽ tạo ra các khóa vòng từ khóa gốc

Để tạo ra các khóa vòng, chúng ta cần bộ hằng số vòng (round constant), vòng thứ I của thủ tục sinh khóa là một word:

$$\square\square\square\square_{\square} = [\square\square_{\square}, 00_{16}, 00_{16}, 00_{16}]$$

Trong đó, $\square\square_{\square}$ được định nghĩa là một số 8bit được định nghĩa như sau:

$$rc_i = \begin{cases} 1 & h\ \square = 1 \\ 2 \bullet \square_{\square-1} & h\ \square > 1\ \square à\ \square\square_{\square} < 80_{16} \\ (2 \bullet \square\square_{\square-1}) \oplus 11B_{16} & h\ \square > 1\ \square à\ \square\square_{\square} \geq 80_{16} \end{cases}$$

AES sử dụng $\square\square\square\square_{10}$ cho AES-128, $\square\square\square\square_{8}$ cho AES-192 và $\square\square\square\square_{7}$ cho AES-256. Giờ, ta định nghĩa:

$\square$ là độ dài khóa, đơn vị là word (32 bit). Với AES-128, $\square = 4$; với AES-192, $\square = 6$; và với AES-256, $\square = 8$.

$K_0, \square_1, \dots, \square_{\square-1}$ là các khối word của khóa gốc.

$\square$ là số khóa vòng cần có. Với AES-128, $\square = 11$; với AES-192, $\square = 13$; và với AES-256, $\square = 15$.

$\square_0, \square_1, \dots, \square_{4R-1}$ là số word của khóa vòng.

Ta đồng thời định nghĩa các hàm RotWord và SubWord:

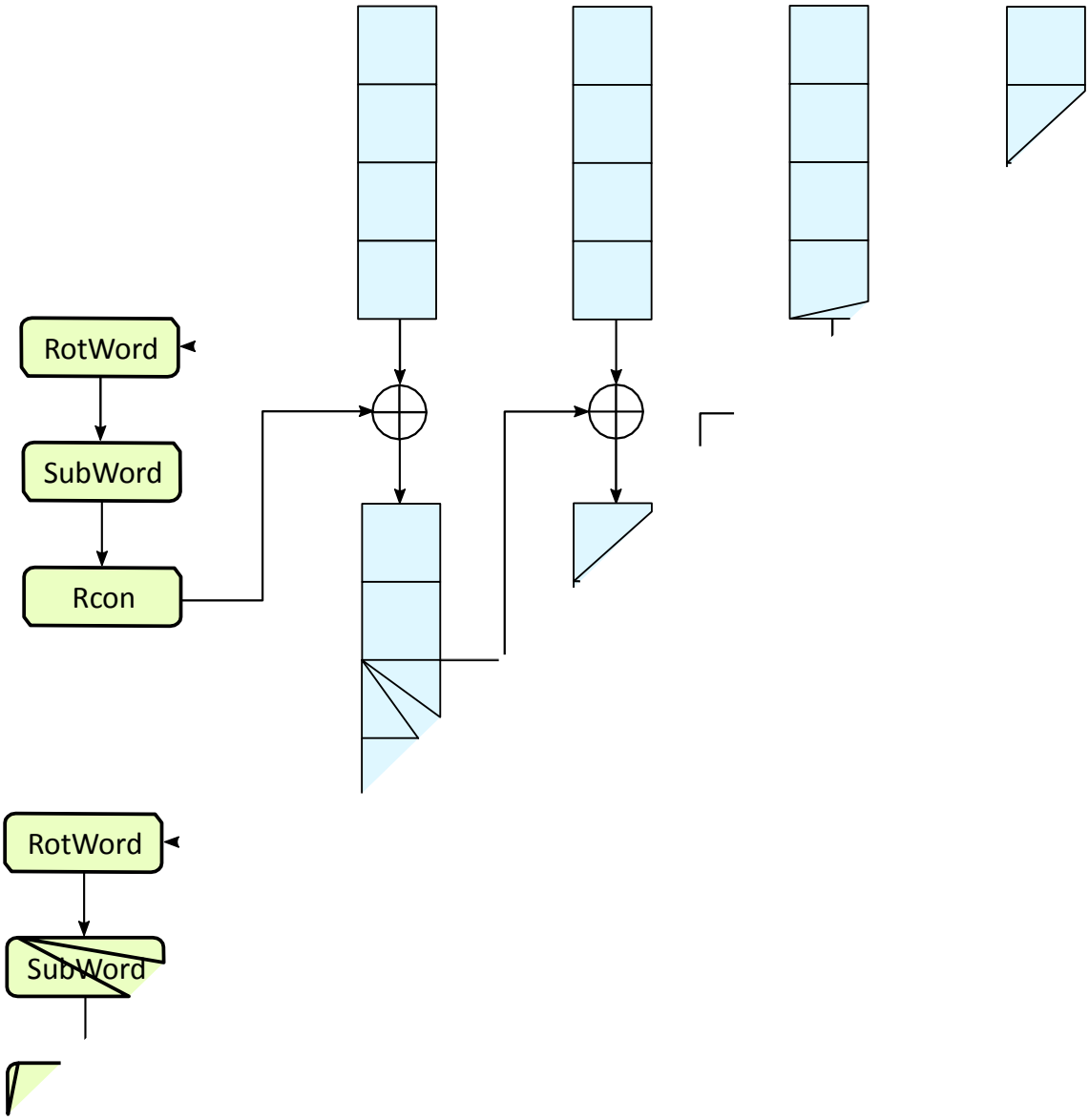
$$RotWord([b0\ \square1\ \square2\ \square3]) = [\square1\ \square2\ \square3\ \square0]$$

$$SubWord([b0\ \square1\ \square2\ \square3]) = [(0)\ (1)\ (2)\ (3)]$$

Thì với $\square = 0, \dots, 4R - 1$:

$$\square_{\square} = \begin{cases} K_{\square} & h\ \square < \square \\ \square_{-} \oplus ((\square_{\square-1})) \oplus \square\square\square\square_{\square} & h\ \square \geq \square\ à\ \square\ \square\square\ \square \equiv 0 \\ \square_{-} \oplus \square\square\square\square\square\square\square(\square_{\square-1}) & h\ \square \geq \square, \square > 6\ \square à\ \square \equiv 4(\square) \\ \square_{-} \oplus \square_{\square-1} & \text{trường hợp còn lại} \end{cases}$$

Ảnh minh họa của quá trình sinh khóa được thể hiện như sau:



V. Các dạng tấn công vào AES và phương pháp phòng chống.

1.Side-channel attack.

- Side Channels (Kênh kề) được định nghĩa là các kênh đầu ra không mong muốn từ một hệ thống.
- Tấn công kênh bên hay còn gọi là Tấn công kênh kề là loại tấn công dễ thực hiện trong các loại tấn công mạnh chống lại quá trình triển khai mã hóa, và mục tiêu của loại tấn công này là phân tích các nguyên tố, các giao thức, modul, và các thiết bị trong mỗi hệ thống.
- Phân loại :
    - Tấn công thời gian.
    - Tấn công dựa vào lỗi.
    - Tấn công phân tích năng lượng.
    - Tấn công phân tích điện từ.

2. Known attacks.

- Vào năm 2002, Nicolas Courtois và Josef Pieprzyk phát hiện một tấn công trên lý thuyết gọi là tấn công XSL và chỉ ra điểm yếu tiềm tàng của AES.
- Tuy nhiên, một vài chuyên gia về mật mã học khác cũng chỉ ra một số vấn đề trong cơ sở toán học của tấn công này và cho rằng các tác giả đã có sai lầm trong tính toán. Việc tấn công dạng này có thực sự trở thành hiện thực hay không vẫn còn để ngỏ và cho tới nay thì tấn công XSL vẫn chỉ là suy đoán.

3.Các phương pháp phòng chống.

- Phương pháp 1: Mã hóa cực mạnh
    - Sử dụng các biện pháp để tăng tính bảo mật của các thuật toán mã hóa.
- Phương pháp 2: Bảo vệ dữ liệu theo phương pháp vật lý
    - Nếu một kẻ tấn công không thể tiếp cận vật lý với dữ liệu, dĩ nhiên khả năng đánh cắp khóa mã hóa sẽ khó khăn hơn.
    - Vì vậy, trước những cuộc tấn công qua âm thanh tiềm tàng, bạn có thể sử dụng các giải pháp bảo vệ vật lý như đặt laptop vào các hộp cách ly âm thanh, không để ai lại gần máy tính khi đang giải mã dữ liệu hoặc sử dụng các nguồn âm thanh băng rộng tần số đủ cao để gây nhiễu.
- Phương pháp 3: Kết hợp cả 2 cách trên

VI. Ứng dụng viết chương trình mã hóa và giải mã AES sử dụng Python:

1. Các hàm nhân Galois:

Các hàm nhân Galois được ứng dụng theo lý thuyết của không gian hữu hạn Galois GF(2^8).

Trong đó, ta quy định phép cộng là phép XOR trên từng bit, phép nhân với 1 là phép XOR với chính nó, phép nhân với 2 là phép nhân với x, tức nhân với 00000010. Trong thuật toán nhân với 2, chúng ta sẽ dựa vào MSB của số bị nhân. Nếu MSB = 0, chúng ta sẽ dịch trái số bị nhân và thêm 0 vào cuối. Nếu MSB = 1, ta sẽ thêm một bước XOR với 00011011 (0x11B).

Thế nhưng, trong thực tiễn, chúng ta sẽ kết hợp phép bắt MSB với phép dịch trái, vì vậy chúng ta sẽ chỉ cần XOR với 00001011 (0x1B). Với phép nhân lớn hơn 2, chúng ta sẽ tận dụng các tính chất giao hoán, kết hợp và phân phối của trường Galois. Cụ thể như sau:

$$
\begin{aligned}
& B_1 \times 10000011 \\
= \quad & B_1 \times (00000001 \ + \ 00000010 \ + \ 10000000) \\
= \quad & (B_1 \times 00000001) \ + \ (B_1 \times 00000010) \ + \ (B_1 \times 10000000) \\
= \quad & (B_1 \times 00000001) \ \otimes \ (B_1 \times 00000010) \ \otimes \ (B_1 \times 10000000)
\end{aligned}
$$

Phép nhân Galois sẽ được ứng dụng như sau:

```
def mul2(r):
    b = [0 for i in range(4)]
    for c in range(0, 4):
        h = (r[c] >> 7) & 1
        b[c] = r[c] << 1
        b[c] ^= h * 0x1B
    return b

def mul3(r):
    b = mul2(r);
    for c in range(0, 4):
        b[c] = b[c] ^ r[c]
    return b

def mul9(r):
    b = list.copy(r)
    for i in range(0, 3):
        b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
```

```
    return b

def mul11(r):
    b = list.copy(r)
    b = mul2(b)
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    return b

def mul13(r):
    b = list.copy(r)
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    b = mul2(b)
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    return b

def mul14(r):
    b = list.copy(r)
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    b = mul2(b)
    for c in range(0, 4):
        b[c] ^= r[c]
    b = mul2(b)
    return b
```

2. Hàm RotWords:

Hàm RotWord được triển khai như sau:

```
def rotWord(r):
    r[0], r[1], r[2], r[3] = r[1], r[2], r[3], r[0]
    return r
```

3. Hàm KeyExpansion:

Các hàm KeyExpansion trong bài báo cáo này nhận đối số là nghịch đảo của ma trận cipherkey. Các phép nghịch đảo được ứng dụng thông qua thư viện toán học math của Python và thư viện Numpy dành cho đại số tuyến tính.

Hàm KeyExpansion được triển khai như sau:

```
def keyExpansion128(key):
    retkey = []
    retkey.append(list.copy(key))
    for i in range(0, 10):
        newkey = [];
        interkey = list.copy(retkey[-1]) # 4x4 array
        interkey = np.transpose(interkey)
        interkey = interkey.tolist()
        rconarr = [rcon[i], 0, 0, 0]
        workingarr = list.copy(interkey[-1]) # 1x4 array
        workingarr = rotWord(workingarr)
        for q in range(0, 4):
            workingarr[q] = sbox[workingarr[q]]
        for j in range(0, len(workingarr)):
            workingarr[j] = workingarr[j] ^ interkey[0][j] ^ rconarr[j]
        newkey.append(list.copy(workingarr))
        for k in range(1, 4):
            for j in range(0,
            4):
                workingarr[j] = workingarr[j] ^ interkey[k][j]
            newkey.append(list.copy(workingarr))
        newkey = np.transpose(newkey)
        newkey = newkey.tolist()
        retkey.append(newkey)

    return retkey


def keyExpansion192(key):
    retkey = []

    #key: 6 x 4 array

    for i in key:
        retkey.append(list.copy(i))

    for i in range(0, 8):
        rconarr = [rcon[i], 0, 0, 0]
        index = len(retkey) - 6
        k6n_6 = list.copy(retkey[index])
        workingarr = list.copy(retkey[-1])
        workingarr = rotWord(workingarr)
        for q in range(0, 4):
            workingarr[q] = sbox[workingarr[q]]
        for j in range(0, len(workingarr)):
            workingarr[j] = workingarr[j] ^ k6n_6[j] ^ rconarr[j]
        retkey.append(list.copy(workingarr))
        index += 1
        for k in range(0, 5):
            for j in range(0,
            4):
                workingarr[j] = workingarr[j] ^ retkey[index][j]
            retkey.append(list.copy(workingarr))
```

```
            index += 1

    expandedKey = []

    for i in range(0, 13):
        interkey = []
        for j in range(0, 4):
            interkey.append(list.copy(retkey.pop(0)))
        interkey = np.transpose(interkey)
        interkey = interkey.tolist()
        expandedKey.append(interkey)

    return expandedKey


def keyExpansion256(key):
    retkey = []

    #key: 8 x 4 array

    for i in key:
        retkey.append(list.copy(i))

    for i in range(0, 7):
        rconarr = [rcon[i], 0, 0, 0]
        index = len(retkey) - 8
        k8n_8 = list.copy(retkey[index])
        workingarr = list.copy(retkey[-1])
        workingarr = rotWord(workingarr)
        for q in range(0, 4):
            workingarr[q] = sbox[workingarr[q]]
        for j in range(0, len(workingarr)):
            workingarr[j] = workingarr[j] ^ k8n_8[j] ^ rconarr[j]
        retkey.append(list.copy(workingarr))
        index += 1
        for k in range(0, 3):
            for j in range(0,
            4):
                workingarr[j] = workingarr[j] ^ retkey[index][j]
            retkey.append(list.copy(workingarr))
            index += 1

        for q in range(0, 4):
            workingarr[q] = sbox[workingarr[q]]

        for j in range(0, 4):
            workingarr[j] = workingarr[j] ^ retkey[index][j]
        retkey.append(list.copy(workingarr))
        index += 1

        for k in range(0, 3):
            for j in range(0,
```

```
                workingarr[j] = workingarr[j] ^ retkey[index][j]
            retkey.append(list.copy(workingarr))
            index += 1

    expandedKey = []

    for i in range(0, 15):
        interkey = []
        for j in range(0, 4):
            interkey.append(list.copy(retkey.pop(0)))
        interkey = np.transpose(interkey)
        interkey = interkey.tolist()
        expandedKey.append(interkey)

    return expandedKey
```

4. Các hàm addRoundKey, subBytes, shiftRows, gMixColumns:

Hàm addRoundKey chỉ là hàm XOR từng phần tử giữa 2 ma trận với nhau, nên người viết xin được không bao đóng thành hàm.

Các hàm còn lại đã được bao đóng thành hàm, và được thể hiện dưới đây:

```
def subBytes(r):
    for i in range(0, 4):
        for j in range(0, 4): r[i]
            [j] = sbox[r[i][j]]
    return r

def shiftRows(r):
    r[1][0], r[1][1], r[1][2], r[1][3] = r[1][1], r[1][2], r[1][3], r[1][0]
    r[2][0], r[2][1], r[2][2], r[2][3] = r[2][2], r[2][3], r[2][0], r[2][1]
    r[3][0], r[3][1], r[3][2], r[3][3] = r[3][3], r[3][0], r[3][1], r[3][2]
    return r

def gMixColumn(r):

    a = [0, 0, 0, 0] #[0 for i in range(4)]
    b = [0, 0, 0, 0] #[0 for i in range(4)]
    r1 = [0, 0, 0, 0]

    for c in range(0, 4):
        a[c] = r[c]
        h = (r[c] >> 7) & 1
        b[c] = r[c] << 1
        b[c] ^= h * 0x1B

    r1[0] = (b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1]) % 256
    r1[1] = (b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2]) % 256
    r1[2] = (b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3]) % 256
```

```
    r1[3] = (b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0]) % 256

    return r1

def gMixColumns(d):

    r =

    list.copy(d)
    r = np.transpose(r)
    r = r.tolist()
    r1 = []

    for i in range(0, 4):
        r[i] = gMixColumn(r[i])
        r1.append(r[i])

    r1 = np.transpose(r1)
    r1 = r1.tolist()
    return r1
```

5. Các hàm invSubBytes, invShiftRows,

gInvMixColumns: Các hàm này được ứng dụng như dưới

đây:

```
def invSubBytes(r):
    for i in range(0, 4):
        for j in range(0, 4): r[i]
            [j] = rbox[r[i][j]]
    return r

def invShiftRows(r):
    r[1][0], r[1][1], r[1][2], r[1][3] = r[1][3], r[1][0], r[1][1], r[1][2]
    r[2][0], r[2][1], r[2][2], r[2][3] = r[2][2], r[2][3], r[2][0], r[2][1]
    r[3][0], r[3][1], r[3][2], r[3][3] = r[3][1], r[3][2], r[3][3], r[3][0]
    return r

def

    gInvMixColumn(r)

    : a = mul9(r)
    b          =
    mul11(r)   c
    =   mul13(r)
    d          =
    mul14(r)

    ret = [0, 0, 0, 0]

    ret[0] = (d[0] ^ b[1] ^ c[2] ^ a[3]) % 256
    ret[1]
```

```
ret[2] = (c[0] ^ a[1] ^ d[2] ^ b[3]) % 256
ret[3] = (b[0] ^ c[1] ^ a[2] ^ d[3]) % 256

return ret
```

```
def gInvMixColumns(d):

    r = list.copy(d)
    r = np.transpose(r)
    r = r.tolist()
    r1 = []

    for i in range(0, 4):
        r[i] = gInvMixColumn(r[i])
        r1.append(r[i])

    r1 = np.transpose(r1)
    r1 = r1.tolist()
    return r1
```

6. Các hàm phiên mã AES-128, AES-192, AES-256:

Các hàm trên sẽ nhận một ma trận 4x4 và sẽ trả về một ma trận 4x4

Các hàm này được ứng dụng như dưới đây:

```
def AES128(state, cypherkey):
    print("AES128")
    roundKey = keyExpansion128(cypherkey) # 11 x (4 x 4) array

    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[0][i][j]

    for q in range(1, 10):
        result =
        subBytes(result)
        result = shiftRows(result)
        result = gMixColumns(result)
        for i in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]

    result = subBytes(result)
    result = shiftRows(result)
    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = result[i][j] ^ roundKey[10][i][j]

    return result

def AES192(state, cypherkey):
    print("AES192")
    roundKey = keyExpansion192(cypherkey) # 11 x (4 x 4) array
```

```
    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[0][i][j]

    for q in range(1, 12):
        result =
        subBytes(result)
        result = shiftRows(result)
        result = gMixColumns(result)
        for i in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]

    result = subBytes(result)
    result = shiftRows(result)
    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = result[i][j] ^ roundKey[12][i][j]

    return result

def AES256(state, cypherkey):
    print("AES256")
    roundKey = keyExpansion256(cypherkey)

    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[0][i][j]

    for q in range(1, 14):
        result =
        subBytes(result)
        result = shiftRows(result)
        result = gMixColumns(result)
        for i in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]

    result = subBytes(result)
    result = shiftRows(result)
    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = result[i][j] ^ roundKey[14][i][j]

    return result
```

7. Các hàm giải mã AES-128, AES-192, AES-256:

Các hàm giải mã nhận một ma trận 4x4 và trả về một ma trận 4x4

```
def AES128(state, cypherkey):

    roundKey = keyExpansion128(cypherkey) # 11 x (4 x 4) array

    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[10][i][j]

    for q in range(9, 0, -1):
        result = invShiftRows(result)
        result = invSubBytes(result)
        for i in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]
        result = gInvMixColumns(result)

    result =
    invShiftRows(result) result
    = invSubBytes(result) for i
    in range(0, 4):
        for j in range(0, 4):
            result[i][j] = result[i][j] ^ roundKey[0][i][j]

    return result


def AES192(state, cypherkey):
    roundKey = keyExpansion192(cypherkey) # 11 x (4 x 4) array

    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[12][i][j]

    for q in range(11, 0, -1):
        result =
        invShiftRows(result) result
        = invSubBytes(result) for i
        in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]
        result = gInvMixColumns(result)

    result =
    invShiftRows(result) result
    = invSubBytes(result) for i
    in rang
        for
```

```
            result[i][j] = result[i][j] ^ roundKey[0][i][j]

    return result


def AES256(state, cypherkey):
    roundKey = keyExpansion256(cypherkey) # 11 x (4 x 4) array

    result = list.copy(state)

    for i in range(0, 4):
        for j in range(0, 4):
            result[i][j] = state[i][j] ^ roundKey[14][i][j]

    for q in range(13, 0, -1):
        result =
        invShiftRows(result) result
        = invSubBytes(result) for i
        in range(0, 4):
            for j in range(0, 4):
                result[i][j] = result[i][j] ^ roundKey[q][i][j]
        result = gInvMixColumns(result)

    result =
    invShiftRows(result) result
    = invSubBytes(result) for i
    in range(0, 4):
        for j in range(0, 4):
            result[i][j] = result[i][j] ^ roundKey[0][i][j]

    return result
```

8. Các hàm chức năng chuyển đổi string thành ma trận và ngược lại:

```
def stringToMat(s):

    ret = []
    interkey = []

    for i in range(0, len(s)):
        interkey.append(ord(s[i]))
        if ((i % 4 == 3)):
            ret.append(interkey)
            interkey = []

    ret = np.transpose(ret)
    ret = ret.tolist()
    return ret

def matToString(s):

    s = np.transpose(s)
    s = np.ravel(s)
```

```
    s = s.tolist()

    retString = ""

    for i in s:
        retString += chr(i);

    return retString
```

9. Đóng gói thành hàm phiên mã và giải mã hoàn chỉnh

Với các hàm đóng gói hoàn chỉnh này, ta có khả năng nhập key với độ dài 16, 24 và 32 ký tự, và hàm sẽ tự động chọn thuật toán tương đương để phiên hay giải mã

```
def encrypt(state = None, key = None):

    ret = ""

    while(state == None):
        print("Please insert your plaintext?")
        state = input()

    while(key == None or not (len(key) == 16 or len(key) == 32 or len(key) ==
24)):
        print("Please insert your cipher key? Your key must be of length 16,
24 or 32")
        key = input()

    lenkey = len(key)

    func = {
        16: en.AES128,
        24: en.AES192,
        32: en.AES256
    }

    res = [state[y - 16:y] for y in range(16, len(state) + 16, 16)]

    lim = 16 - len(res[-1])

    for i in range(0, lim):
        res[-1] +=
        chr(0x00)

    key = stringToMat(key)
    if (lenkey != 16):
        cypherkey = np.transpose(key)
        cypherkey = cypherkey.tolist()
    else: cypherkey = key

    for i in res:
        sub = stringToMat(i)
```

```
        sub = func[lenkey](sub, cypherkey)
        sub = matToString(sub)
        ret += sub

    return ret


def decrypt(state = None, key = None):

    ret = ""

    while(state == None):
        print("Please insert your plaintext?")
        state = input()

    while(key == None or not (len(key) == 16 or len(key) == 32 or len(key) ==
24)):
        print("Please insert your cipher key? Your key must be of length 16,
24 or 32")
        key = input()

    lenkey = len(key)

    func = {
        16: de.AES128,
        24: de.AES192,
        32: de.AES256
    }

    res = [state[y - 16:y] for y in range(16, len(state) + 16, 16)]

    lim = 16 - len(res[-1])

    for i in range(0, lim):
        res[-1] +=
        chr(0x00)

    key = stringToMat(key)
    if (lenkey != 16):
        cypherkey = np.transpose(key)
        cypherkey = cypherkey.tolist()
    else: cypherkey = key

    for i in res:
        sub = stringToMat(i)
        sub = func[lenkey](sub, cypherkey)
        sub = matToString(sub)
        ret += sub

    return ret
```

10. Các chế độ hoạt động (modus operandi – mode of operation) của mật mã khối, và ứng dụng vào phần mềm AES.

Mật mã khối có nhiều chế độ hoạt động khác nhau. Một mật mã khối chỉ có thể đảm bảo sự an toàn và toàn vẹn thông tin khi thuật toán đó được áp dụng lên các khối dữ liệu có độ dài bit được quy ước sẵn trước. Với các lượng dữ liệu có độ dài tùy ý, thuật toán sẽ chia dữ liệu đầu vào thành các khối có độ dài phù hợp, rồi lần lượt mã hóa và xử lý để đưa ra được bản mã hoàn chỉnh.

Để giải mã, ta thực hiện lần lượt các quy trình thành phần nghịch đảo với các quy trình thành phần ở quá trình mã hóa.

Một chế độ hoạt động của mật mã khối được quy ước là cách thuật toán thực hiện việc mã hóa lần lượt tất cả từng khối con được tách ra từ đầu vào để biến đổi đầu vào đó một cách bảo mật.

Hầu hết các chế độ hoạt động sẽ cần một chuỗi bit đặc biệt có độ dài bit phù hợp với thuật toán để thực hiện mã hóa. Chuỗi bit đó được gọi là vector khởi tạo (Initialization vector – IV). Một IV phải thỏa mãn điều kiện là không lặp lại và có tính ngẫu nhiên (có thể là ngẫu nhiên vật lý hoặc giả ngẫu nhiên toán học).

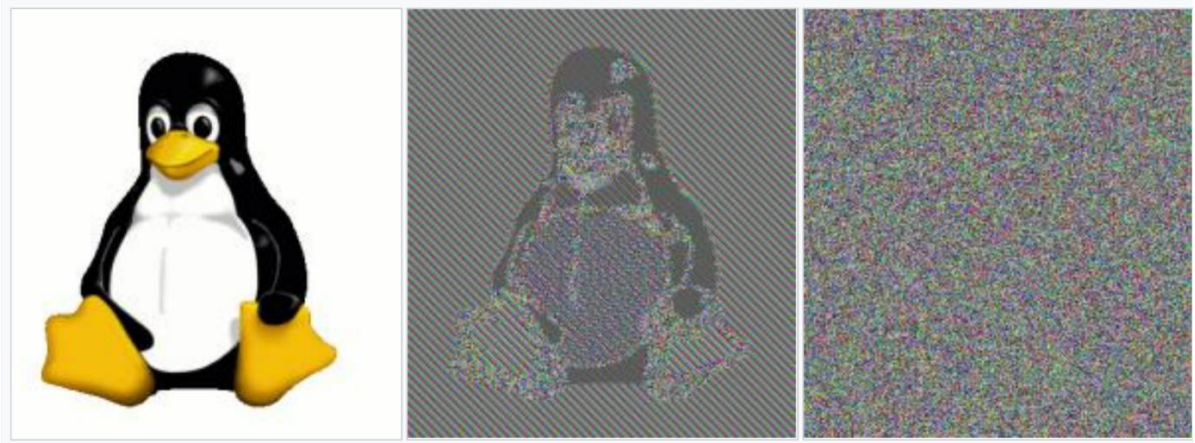
Trong khuôn khổ của bài tập lớn này, người viết xin giới hạn 4 chế độ hoạt động, đó là ECB (mã hóa sách số - electronic codebook), CBC (mã hóa khối nối tiếp – Cipher Block Chaining), CFB (mã hóa phản hồi - cipher feedback) và OFB (phản hồi đầu ra – output feedback).

Tất cả các chế độ mã hóa này khác nhau trong cách hoạt động, vậy nhưng đều sử dụng một công nghệ cốt lõi, là mã hóa khối AES.

Bên dưới sẽ là ảnh minh họa các chế độ mã hóa khối, đi kèm với đó là đoạn mã ứng dụng viết bằng ngôn ngữ Python.

a) Electronic Code Book:

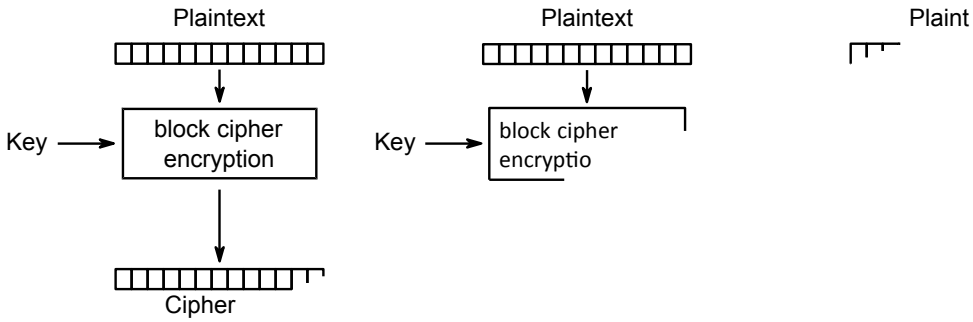
Electronic Code Book là chế độ đơn giản nhất. Người xử lý mã hóa sẽ biến đoạn mã hóa dài trở thành các khối dữ liệu 128 bit, rồi thực hiện mã hóa AES trên từng khối một. Tất cả các khối sẽ trải qua chu trình mã hóa giống hệt nhau. Chính vì vậy, mã hóa này được coi là không an toàn, vì thiếu đi tính diffusion cần thiết. Chúng ta có thể thấy rõ điều này khi ta mã hóa một file kiểu bitmap:

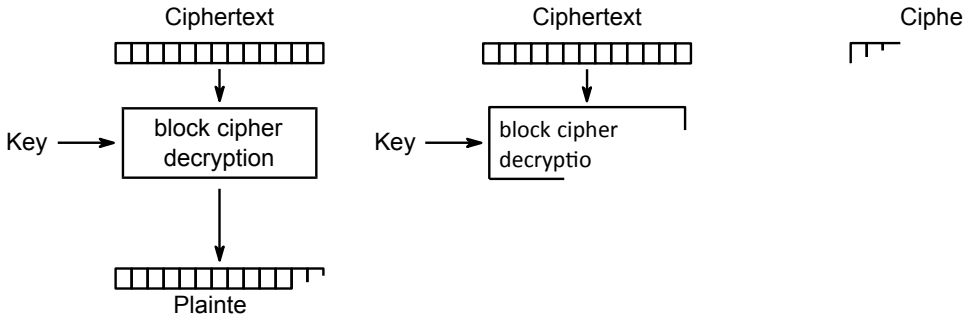


Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

Ta thấy rõ được rằng, việc sử dụng mã hóa ECB không che dấu được các pattern của dữ liệu. Giống như các dạng mã hóa cổ điển, chỉ cần thu thập được các một lượng cực lớn các dữ liệu, chúng ta có thể sử dụng frequency analysis để phá mã. Chính vì thế, ECB được coi là kiểu mã hóa không an toàn và không nên sử dụng với mục đích an ninh hay mục đích thương mại.

Hình ảnh minh họa của chế độ mã hóa và giải mã ECB được biểu diễn dưới hình vẽ sau:





Đoạn code mã hóa và giải mã sử dụng ECB như sau:

Phiên mã:

```
    else:
        for i in res:
            sub  =  strConversion[hexain](i)
            sub = func[lenkey](sub, cypherkey)
            sub = matConversion[hexaout](sub)
            ret += sub
```

Giải mã:

```
    else:
        for i in res:
            sub  =  strConversion[hexain](i)
            sub = func[lenkey](sub, cypherkey)
            sub = matConversion[hexaout](sub)
```

```
ret += sub
```

b) Cipher block chaining:

Cipher block chaining giới thiệu được tính diffusion cần thiết để mã hóa thông tin an toàn. Trong chế độ này, từng khối plaintext i + 1 sẽ được XOR với khối ciphertext i. Điều này có nghĩa là khối ciphertext sau sẽ phải phụ thuộc vào khối ciphertext đằng trước. Để từng khối được mã hóa với đặc tính ngẫu nhiên và đặc trưng, ta sẽ XOR khối đầu tiên với một IV (initialization vector).

Nếu khối đầu tiên có chỉ số index = 1, thì công thức toán học cho mã hóa sử dụng CBC là:

$$C_i = E_K(P_i \oplus C_{i-1}),$$
$$C_0 = IV$$

Và công thức toán học cho giải mã sử dụng CBC là:

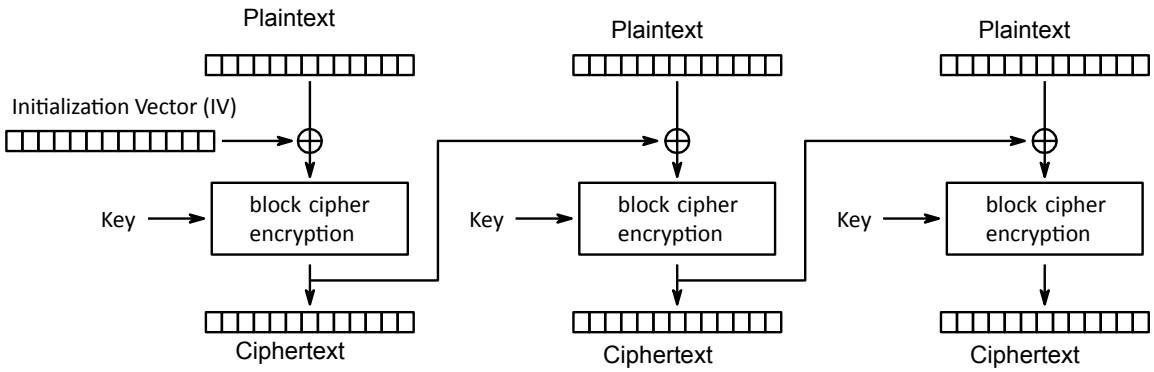
$$P_i = D_K(C_i) \oplus C_{i-1},$$
$$C_0 = IV$$

CBC là modus operandi được sử dụng rộng rãi nhất, thế nhưng nó cũng có các bất cập. Quá trình mã hóa của CBC bắt buộc phải xử lý nối tiếp nhau (do khối sau phụ thuộc vào khối trước), và vẫn yêu cầu padding dữ liệu)

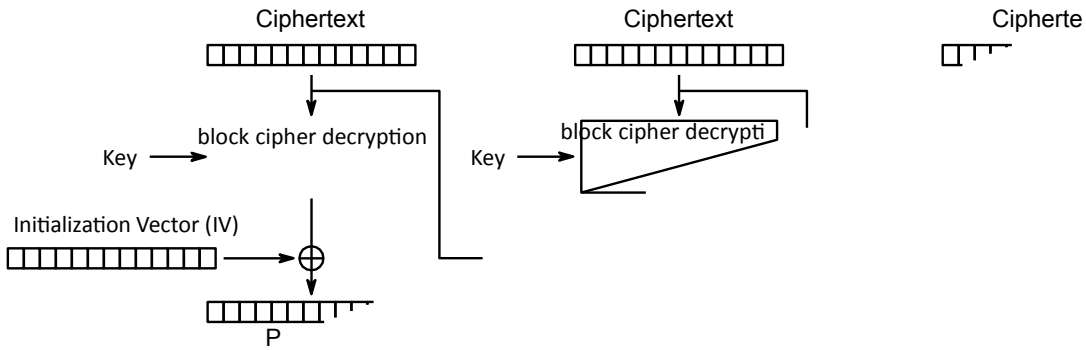
Nếu như ta giải mã mà không biết IV, thì chỉ có khối đầu tiên bị lỗi, các khối còn lại không bị ảnh hưởng. Đó là trong quá trình giải mã khối i, chúng ta chỉ cần cộng cipherkey khối i – 1. Cũng chính vì tính chất này, mà ta có thể tìm được bản rõ của khối i nhờ vào 2 khối liền kề, tức là khối i – 1 và i + 1.

Lỗ hổng của CBC dẫn đến phương pháp Explicit initialization vectors.

Hình ảnh minh họa phiên mã và giải mã CBC được biểu diễn như sau:



Cipher Block Chaining (CBC) mode encryption



Đoạn code phiên mã và giải mã của chế độ CBC được đính kèm bên dưới:

Phiên mã:

```
if (mode == "CBC"):
    temp = strConversion[hexaIV](IV)
    for i in res:
        sub = strConversion[hexain](i)
        sub = xorMatrix(sub, temp)
        sub = func[lenkey](sub, cypherkey)
        temp = sub
        sub = matConversion[hexaout](sub)
        ret += sub
```

Giải mã:

```
if (mode == "CBC"):
    temp = strConversion[hexaIV](IV)
    for i in res:
        sub = strConversion[hexain](i)
        sub = func[lenkey](sub, cypherkey)
        sub = xorMatrix(sub, temp)
        sub = matConversion[hexaout](sub)
        temp = strConversion[hexain](i)
        ret += sub
```

c) Cipher feedback

Ở đây, chúng ta sẽ sử dụng CFB dưới dạng full cipher feedback. Có nhiều loại CFB khác nhau, như CFB-1, CFB-8, CFB-64, CFB-128,…, nhưng full cipher feedback là kiểu đơn giản nhất.

Sự khác biệt giữa CFB và CBC là với CFB, các khối bản rõ không đi qua thuật toán AES. Chỉ có key và IV mới tương tác với thuật toán. IV ở đây đóng vai trò làm bản rõ, key vẫn giữ nguyên vai trò của mình. Sau khi IV được mã hóa, nó sẽ được XOR với các khối bản rõ. Kết quả của phép toán đó lại được sử dụng để mã hóa cho khối tiếp theo. Công thức toán học của CFB được thể hiện như sau:

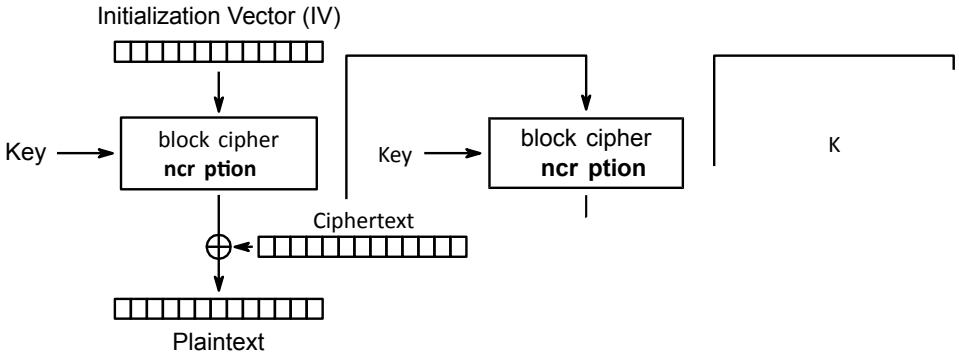
$$C_i = \begin{cases} IV, i = 0 \\ E_k(C_{i-1}) \oplus P_i; i > 0 \end{cases}$$

$$C_i = E_K(C_{i-1}) \oplus P_i$$

Ưu điểm của CFB (và cả OFB) chính là, vì CFB không trực tiếp mã hóa bản rõ, thế nên không cần padding.

Giống như CFB, quá trình phiên mã phải thực hiện tuần tự, thế nhưng quá trình giải mã có thể được lập trình để chạy song song

Hình ảnh minh họa của thuật toán CFB được biểu diễn như sau:



Đoạn code phiên mã và giải mã theo chế độ CFB được đính kèm bên dưới:

Phiên mã:

```
    elif (mode == "CFB"):
        temp = strConversion[hexaIV](IV)
        for i in res:
            sub = strConversion[hexain](i)
            temp = func[lenkey](temp, cypherkey)
            sub = xorMatrix(sub, temp)
            temp = strConversion[hexain](i)
            sub = matConversion[hexaout](sub)
            ret += sub
```

Giải mã:

```
    elif (mode == "CFB"):
        temp = strConversion[hexaIV](IV)
```

```
        for i in res:
            sub = strConversion[hexain](i)
            temp = enfunc[lenkey](temp, cypherkey)
            sub = xorMatrix(sub, temp)
            temp = sub
            sub = matConversion[hexaout](sub)
            ret += sub
```

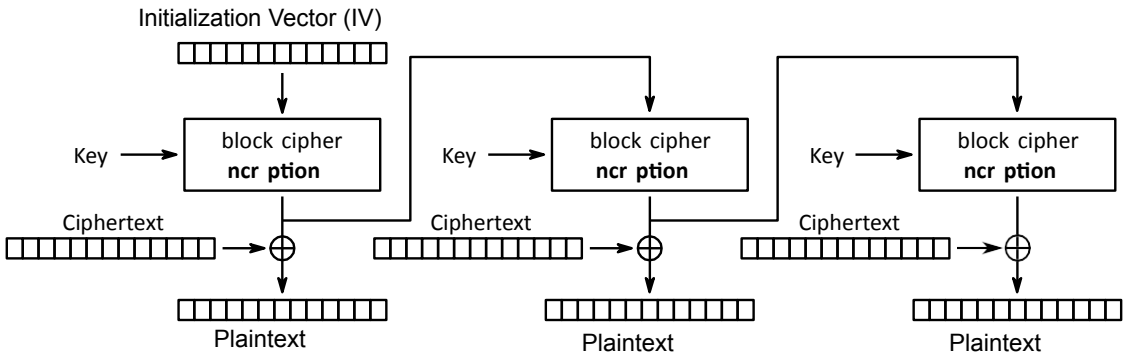
d) Output feedback:

Output feedback là chế độ hoạt động rất linh hoạt, vì nó biến AES từ thuật toán mật mã khối thành mật mã dòng. Khóa được mã hóa trở thành khóa dòng, từ đó sẽ được XOR với các khối bản rõ để trở thành bản mã. Các khối bản rõ sẽ không dính dáng đến thuật toán mã hóa, mà IV sẽ được mã hóa liên tục để trở thành khóa dòng.

Thuật toán OFB mang ưu điểm của thuật toán mã hóa dòng, đó chính là nó cho phép các đoạn code sửa lỗi hoạt động ổn định tại mọi thời điểm.

Một ưu điểm khác của OFB chính là vì nghịch đảo của phép XOR là chính nó, nên thuật toán mã hóa và giải mã là giống hệt nhau.

Hình ảnh minh họa của thuật toán OFB được đính kèm dưới đây:



Output Feedback (OFB) mode decryption

Và công thức toán học của OFB được đính kèm dưới đây:

$$C_i = P_i \oplus O_i$$

$$□_□ = □_□ \oplus □_□$$

$$□_□ = □_□(\ _□)$$

$$□_□ = \ _{□-1}$$

$$□_0 = □□$$

Ta thấy rằng các bản rõ và bản mã không hề tham gia vào quá trình mã hóa, mà chỉ mã hóa AES IV liên tục, với khóa cho trước. Điều này khiến cho từng khối bản rõ được biến đổi một cách ngẫu nhiên và độc nhất, và kết quả của khối trước là độc lập hoàn toàn với các khối lân cận.

Thế nhưng, vì đối tượng được mã hóa là IV, và khối mã IV i phụ thuộc vào khối mã IV i – 1, nên cả phiên mã và giải mã đều bắt buộc phải thực hiện tuyến tính.

Thuật toán phiên mã và giải mã của chế độ OFB được mô tả dưới đây:

Phiên mã:

```
elif (mode == "OFB"):
    temp = strConversion[hexaIV](IV)
    for i in res:
        sub = strConversion[hexain](i)
        temp = func[lenkey](temp, cypherkey)
        sub = xorMatrix(sub, temp)
        sub = matConversion[hexaout](sub)
        ret += sub
```

Giải mã:

```
elif (mode == "OFB"):
    temp = strConversion[hexaIV](IV)
    for i in res:
        sub = strConversion[hexain](i)
        temp = enfunc[lenkey](temp, cypherkey)
        sub = xorMatrix(sub, temp)
        sub = matConversion[hexaout](sub)
        ret += sub
```

TÀI LIỆU THAM KHẢO

I would like to thank the former mathematicians, the programmers that worked tirelessly to come up with these
methods to obfuscate data under such strenuous ways, in order to protect information from being stolen.

The entire code can be obtain from my GitHub repository:
https://github.com/cloudmadeofcandy/AES_implementation

I also deployed the final version of my implementation on Heroku: http://floating-ridge-18343.herokuapp.com/

Book: The Design of Rijndael -
https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf

https://crypto.stackexchange.com/questions/2569/how-does-one-implement-the-inverse-of-aes-mixcolumns
https://en.wikipedia.org/wiki/Rijndael_S-box
https://en.wikipedia.org/wiki/Rijndael_MixColumns
https://en.wikipedia.org/wiki/AES_key_schedule
https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

On the theory of Galois Field GF(2^8) and how it applies in said situation:
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture5.pdf
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture6.pdf
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture7.pdf
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture8.pdf

On the overall mechanism of AES Encryption:
https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf
https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
https://crypto.stackexchange.com/questions/2569/how-does-one-implement-the-inverse-of-aes-mixcolumns
https://crypto.stackexchange.com/questions/51951/aes-key-expansion-for-192-bit
https://crypto.stackexchange.com/questions/81712/rcon-for-aes-192-and-256
https://crypto.stackexchange.com/questions/11096/two-different-approaches-for-key-expansion-using-the-aes-256-algorithm
https://www.baeldung.com/java-aes-encryption-decryption
https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/

FOR CHECKING THE CORRECTNESS OF AES:
https://formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng-html5.html
https://github.com/chrisveness/crypto/blob/master/test/aes-tests.js

https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (Appendix A.1, A.2, and A.3)
http://aes.online-domain-tools.com/
https://string-functions.com/string-hex.aspx
https://searchcode.com/codesearch/view/15318961/
https://github.com/Anexsoft/Bolt-CMS/blob/master/vendor/passwordlib/passwordlib/test/Data/Vectors/aes-ofb.test-vectors