Doc1 an toan va bao mat thong tin

Toán cao cấp (Trường Đại học Công nghệ Thành phố Hồ Chí Minh)

HỌC VIỆN KỸ THUẬT MẬT MÃ

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

CƠ SỞ AN TOÀN VÀ BẢO MẬT THÔNG TIN

ĐỀ TÀI:

VIẾT CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ BẰNG MẬT MÃ AES BẰNG JAVA

Giáo viên hướng dẫn: TS. Nguyễn Đào Trường

Sinh viên thực hiện: Phạm Xuân Hiếu CT030121

Lại Văn Hiếu CT030122

Nhóm: 20

Hà Nội, 12-2021

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

LỜI NÓI ĐẦU

Trong bối cảnh cuộc Cách mạng công nghiệp 4.0, việc ứng dụng các công nghệ hiện đại là xu thế tất yếu. Đi kèm với sự phát triển nhanh chóng đó thì việc làm thế nào để đảm bảo an toàn bí mật cho các thông tin trên không gian mạng cũng được con người đặc biệt quan tâm. Do đó yêu cầu về việc có một cơ chế, giải pháp để bảo vệ sự an toàn và bí mật ngày càng trở lên cần thiết. Nổi bật trong đó là AES - phương pháp mã hóa tiêu chuẩn.

AES (Advanced Encryption Standard) là một thuật toán mã hóa khối được chính phủ Mỹ áp dụng làm tiêu chuẩn mã hóa. Thuật toán AES làm việc với 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.

Sau một thời gian học tập và nghiên cứu bộ môn học phần "Cơ sở an toàn và bảo mật thông tin" dưới chỉ dẫn của thầy TS. Nguyễn Đào Trường, nhóm chúng em đã học hỏi được nhiều kiến thức bổ ích để hoàn thiện Báo Cáo này. Trong bài báo cáo này chúng ta sẽ tìm hiểu về các chu trình làm việc của phương pháp mã hóa AES, các chế độ mã hóa ECB và CBC và ứng dụng viết một chương trình mã hóa và giải mã bằng AES sử dụng ngôn ngữ java, rồi tìm hiểu các dạng tấn công vào AES và việc pháp phòng tránh. Bài báo cáo trình bày những nội dung sau:

- Chương 1: Tổng quan AES.
- Chương 2: Hàm sử dụng trong AES.
- Chương 3: Chế độ mã hóa.
- Chương 4: Ứng dụng viết chương trình mã hóa và giải mã AES sử dụng Java.
- Chương 5: Các dạng tấn công vào AES và phương pháp phòng chống.

Em xin chân thành cảm ơn.

Sinh Viên

Lại Văn Hiếu

CHƯƠNG 1. TỔNG QUAN AES

1.1 Giới thiệu chung

Tiêu chuẩn Advanced Encryption Standard (AES) - Tiêu chuẩn mã hóa tiên tiến là một thuật toán tiêu chuẩn của chính phủ Hoa Kỳ nhằm mã hóa và giải mã dữ liệu do Viện Tiêu chuẩn và Công nghệ quốc gia Hoa Kỳ (National Institute Standards and Technology– NIST) phát hành ngày 26/11/2001 và được đặc tả trong Tiêu chuẩn Xử lý thông tin Liên bang 197 (Federal Information Processing Standard – FIPS 197) sau quá trình kéo dài 5 năm trình phê duyệt, AES tuân theo mục 5131 trong Luật Cải cách quản lý công nghệ thông tin năm 1996 và Luật An toàn máy tính năm 1997.

AES là một thuật toán "mã hóa khối" (block cipher) ban đầu được tạo ra bởi hai nhà mật mã học người Bỉ là Joan Daemen và Vincent Rijmen. Kể từ khi được công bố là một tiêu chuẩn, AES trở thành một trong những thuật toán mã hóa phổ biến nhất sử dụng khóa mã đối xứng để mã hóa và giải mã (một số được giữ bí mật dùng cho quy trình mở rộng khóa nhằm tạo ra một tập các khóa vòng). Ở Việt Nam, thuật toán AES đã được công bố thành tiêu chuẩn quốc gia TCVN 7816:2007 năm 2007 về Thuật toán mã hóa dữ liệu AES.

1.2 Đặc tả kỹ thuật

AES là một thuật toán mã hóa khối đối xứng với độ dài khóa là 128 bít (một chữ số nhị phân có giá trị 0 hoặc 1), 192 bít và 256 bít tương ứng dọi là AES-128, AES-192 và AES-256. AES-128 sử dụng 10 vòng (round), AES-192 sử dụng 12 vòng và AES-256 sử dụng 14 vòng.

Vòng lặp chính của AES thực hiện các hàm sau: SubBytes(), ShiftRows(), MixColumns() và AddRoundKey(). Ba hàm đầu của một vòng AES được thiết kế để ngăn chặn phân tích mã bằng phương thức "mập mờ" (confusion) và phương thức "khếch tán" (diffusion), còn hàm thứ tư mới thực sự được thiết kế để mã hóa dữ liệu. Trong đó "khuếch tán" có nghĩa là các kiểu mẫu trong bản rõ (Dữ liệu đầu vào của phép mã hóa hoặc dữ liệu đầu ra của phép giải mã) được phân tán trong các bản mã (Dữ liệu đầu ra của phép mã hóa hoặc dữ liệu đầu vào của phép giải mã), "mập mờ" nghĩa là mối quan hệ giữa bản rõ và bản mã bị che khuất. Một cách đơn giản hơn để xem thứ tự hàm

AES là: Trộn từng byte (SubBytes), trộn từng hàng (ShiftRows), trộn từng cột (MixColumns) và mã hóa (AddRoundKey).

1.3 Đặc tả thuật toán AES

Đối với thuật toán AES, độ dài của khối đầu vào, khối đầu ra và trạng thái là 128 bít, số các cột (các từ có độ dài 32 bít) tạo nên trạng thái là Nb = 4.

Trong thuật toán AES, độ dài khóa mã K có thể là 128, 192 hay 256 bít. Độ dài khóa được biểu diễn bằng Nk = 4, 6 hoặc 8 thể hiện số lượng các từ 32 bít (số cột) của khóa mã.

Đối với thuật toán AES, số vòng được thay đổi trong quá trình thực hiện thuật toán phụ thuộc vào kích cỡ khóa. Số vòng này được ký hiệu là Nr. Nr = 10 khi Nk = 4, Nr = 12 khi Nk = 6 và Nr = 14 khi Nk = 8.

	Độ dài khóa	Độ dài khối	Số vòng
AES-128	4	4	10
AES-196	6	4	12
AES-256	8	4	14

Bảng 1-1 Tổ hợp khóa – khối – vòng

Đối với phép mã hóa và phép giải mã, thuật toán AES sử dụng một hàm vòng gồm bốn phép biến đổi byte như sau: phép thay thế byte (một nhóm gồm 8 bít) sử dụng một bảng thay thế (Hộp-S), phép dịch chuyển hàng của mảng trạng thái theo các offset (số lượng byte) khác nhau, phép trộn dữ liệu trong mỗi cột của mảng trạng thái, phép cộng khóa vòng và trạng thái.

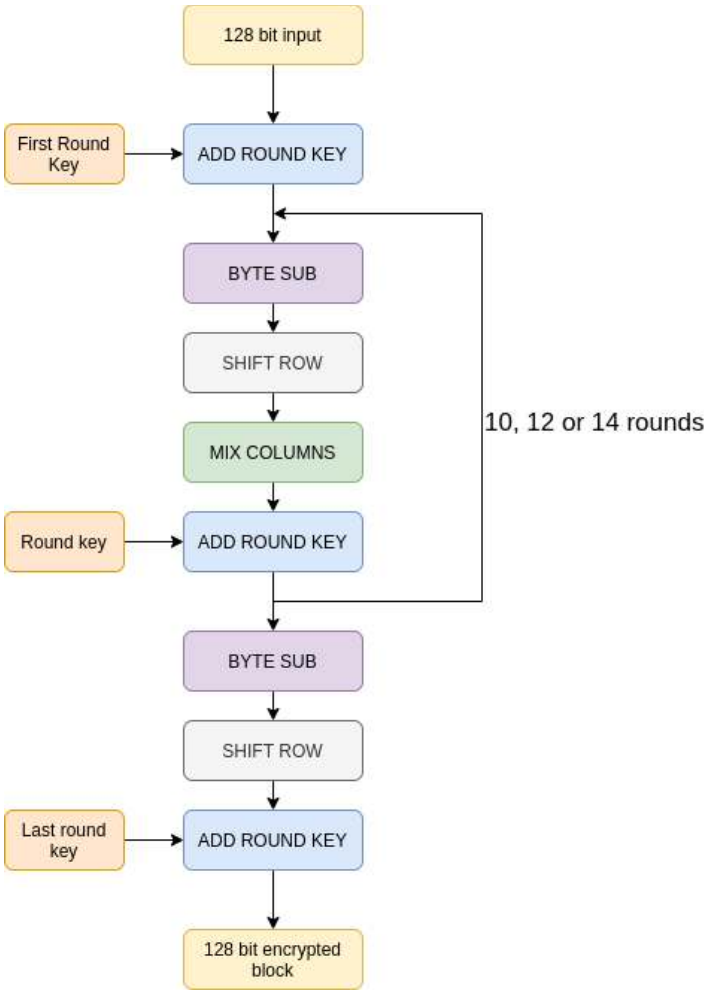
1.4 Phép mã hóa

Tại thời điểm bắt đầu phép mã hóa, đầu vào được sao chép vào mảng trạng thái sử dụng các quy ước. Sau phép cộng khóa vòng khởi đầu, mảng trạng thái được biến đổi bằng cách thực hiện một hàm vòng liên tiếp với số vòng lặp là 10, 12 hoặc 14 (tương ứng với độ dài khóa), vòng cuối cùng khác biệt không đáng kể với Nr-1 vòng đầu tiên. Trạng thái cuối cùng được chuyển thành đầu ra. Hàm vòng được tham số hóa bằng cách sử dụng một lược đồ khóa – mảng một chiều chứa các từ 4 byte nhận từ phép mở rộng khóa.

Phép biến đổi cụ thể gồm SubBytes(), ShiftRows(), MixColumns() và AddRoundKey() dùng để xử lý trạng thái.

1.5 Qúa trình mã hóa

AES có thể mã hóa dữ liệu đầu vào lớn bằng cách chia chúng thành các khối 128 bits. AES có khóa thuộc ba loại kích thước 128, 192, 256 bits và khóa càng dài thì mã hóa càng mạnh. Thuật toán này bao gồm 4 thao tác: Byte Sub, Shift Rows, Mix Columns và Add Round Key. Để trực quan hóa, khối văn bản đầu vào 128 bits sẽ được biểu diễn bằng một ma trận 4 x 4, mỗi vị trí đại diện cho 8 bits từ khối đầu vào.



Hình 1 Encrytion

Vòng lặp chính của AES thực hiện các hàm sau:

- Byte Sub: thay thế các byte dữ liệu (trạng thái)
- Shift Row: dịch vòng dữ liệu (trạng thái)

-

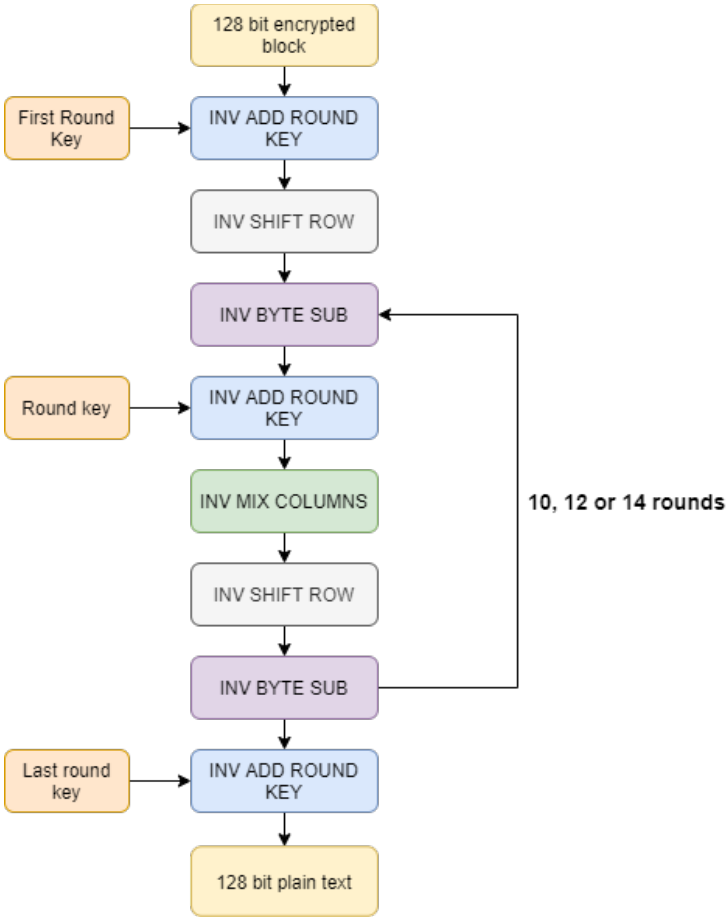
Mix Columns: trộn cột dữ liệu (trạng thái) vào

-

Add Round Key: chèn khóa vòng

1.6 Qúa trình giải mã

Để hoàn thành quá trình giải mã, chúng ta phải đảo ngược quá trình mã hóa. Chúng ta sẽ phải đảo ngược quá trình mã hóa. Chúng ta sẽ thực hiện các thao tác tương tự, chỉ theo thứ tự ngược lại.

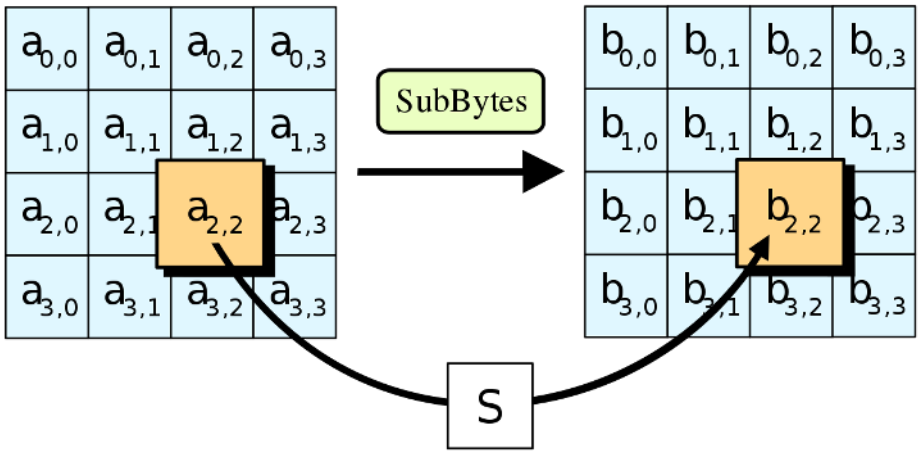


Hình 2 Decryption

CHƯƠNG 2. HÀM SỬ DỤNG TRONG AES

2.1 Hàm Byte Sub

Phép biến đổi dùng trong phép mã hóa áp dụng lên trạng thái (kết quả mã hóa trung gian, được mô tả dưới dạng một mảng chữ nhật của các byte) sử dụng một bảng thay thế byte phi tuyến (Hộp S – bảng thay thế phi tuyến, được sử dụng trong một số phép thay thế byte và trong quy trình mở rộng khóa, nhằm thực hiện một phép thay thế 1-1 đối với giá trị mỗi byte) trên mỗi byte trạng thái một cách độc lập.



Hình 3 Byte Sub

Các giá trị trong bảng tra cứu được biểu diễn theo hệ thập lục phân, mỗi chữ số tương ứng với 4 bits.

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

Hình 4 S-box

2.2 Hàm Inverse Sub Bytes

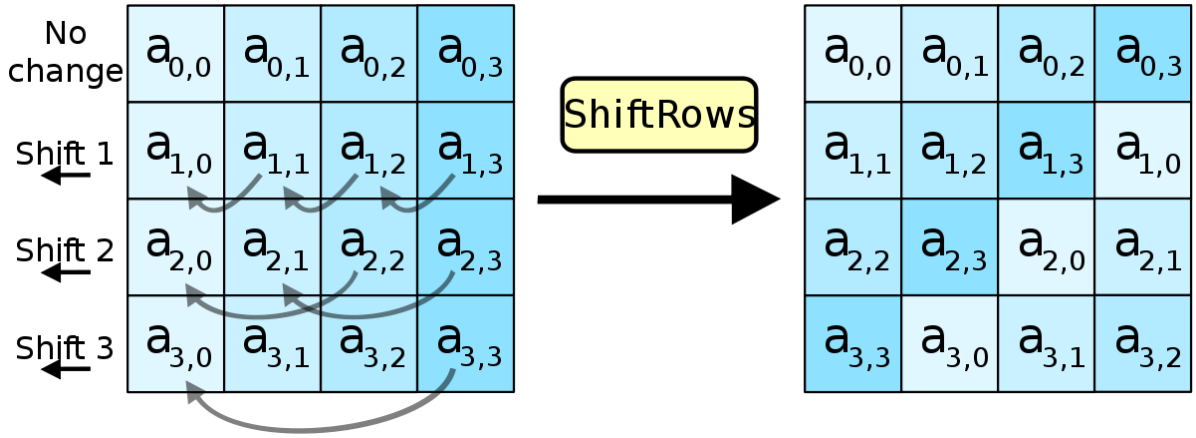
Về bản chất là thao tác giống với Sub Bytes: nó sẽ nhận mỗi khối từ ma trận và hoán đổi nó với một khối khác từ một ma trận được xác định trước. Sự khác biệt duy nhất giữa Inverse Sub Bytes và Sub Bytes ở bảng sau.

		y														
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	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
x	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

Hình 5 Inverse S-box

2.3 Hàm Shift Rows

Đơn giản hơn Byte Sub, shift rows cũng giống như tên của nó, dịch mỗi dòng của ma trận sang bên trái. Dòng đầu tiên sẽ không bị dịch, dòng thứ hai sẽ dịch 1 vị trí, dòng thứ ba dịch 2 vị trí và dòng thứ tư cũng là dòng cuối cùng sẽ dịch 3 vị trí.



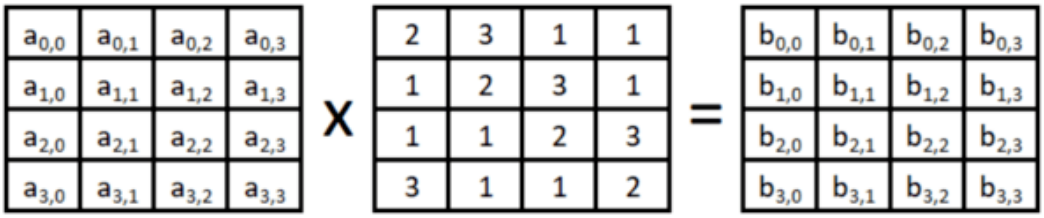
Hình 6 Shift Rows

2.4 Hàm Inverse Shift Row

Shift rows chỉ dịch mỗi dòng một số lượng vị trí nhất định sang bên trái. Do đó để nghịch đảo, chúng ta phải di chuyển mỗi dòng một số lượng vị trí tương tự nhưng về phía bên phải.

2.5 Hàm Mix Columns

Trong Mix Columns, chúng ta sẽ thực hiện một phép nhân ma trận giữa ma trận hiện tại và ma trận đã được xác định trước, bất biến qua các vòng. Tuy nhiên đó sẽ là một phép nhân ma trận phức tạp hơn một tí, vì phép tổng được thay thế bởi phép xor và phép nhân được thay thế bởi phép and.



Hình 7 Mix Columns

2.6 Hàm Inverse Mix Columns

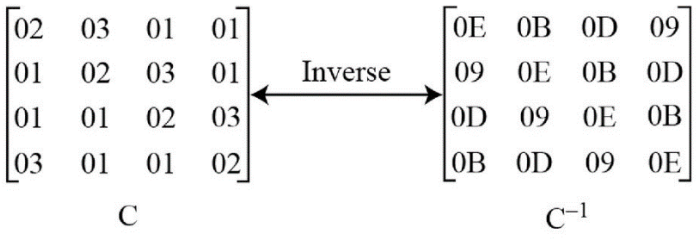
Nếu chúng ta nhân một ma trận C với nghịch đảo của nó, C^-1, thì chúng ta được kết quả là ma trận đơn vị.

C . C^-1 = In

Do đó, gọi ma trận dữ liệu của chúng ta là X. Theo Mix Columns, ta đặt X = X C. Để hoàn tác phép nhân này ta phải nhân nó với nghịch đảo của C, như sau:

X . C . C^-1 = X . In = X

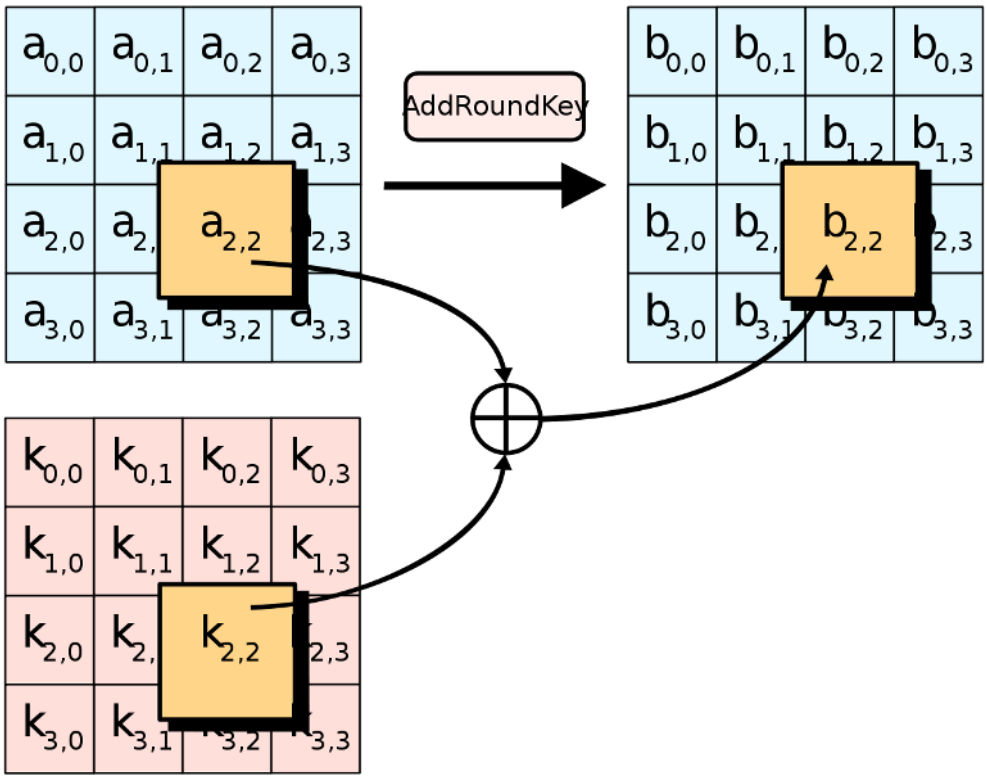
Chúng ta biểu diễn ma trận C theo Mix Columns, vì vậy ở đây chúng ta sẽ biểu diễn ma trận nghịch đảo của C:



Hình 8 Inverse C

2.7 Hàm Add Round Key

Phép biến đổi trong phép mã hóa và phép giải mã. Trong đó, một khóa vòng (các giá trị sinh ra từ khóa mã bằng quy trình mở rộng khóa) được cộng thêm vào trạng thái bằng phép toán XOR (phép toán hoặc và loại trừ). Độ dài của khóa vòng bằng độ dài của trạng thái.



Hình 9 Add Round Key

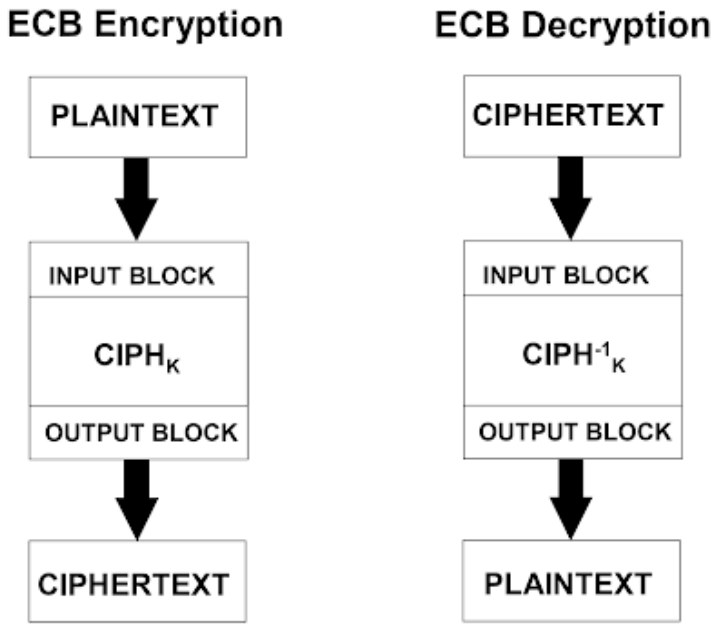
2.8 Hàm Inverse Add Round Key

Inverse Add Round Key là nghịch đảo của chính nó: nếu bạn áp dụng Add Round Key 2 lần, bạn sẽ nhận lại được những gì lúc bạn bắt đầu. Vì tính chất này, để hoàn tác Add Round Key, chúng ta phải áp dụng nó thêm một lần nữa. Đối với điều này, Inverse Add Round Key và Add Round Key là cùng một thao tác.

CHƯƠNG 3. CHẾ ĐỘ MÃ HÓA

3.1 Chế độ ECB (Electronic CodeBook)

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



Hình 10 Chế độ mã hóa/ giải mã ECB

- Ưu điểm:
  + Thiết kế phần cứng đơn giản. Vấn đề cần quan tâm chính là thiết kế logic cho thuật toán mã hóa.
  + Lỗi bit không 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.

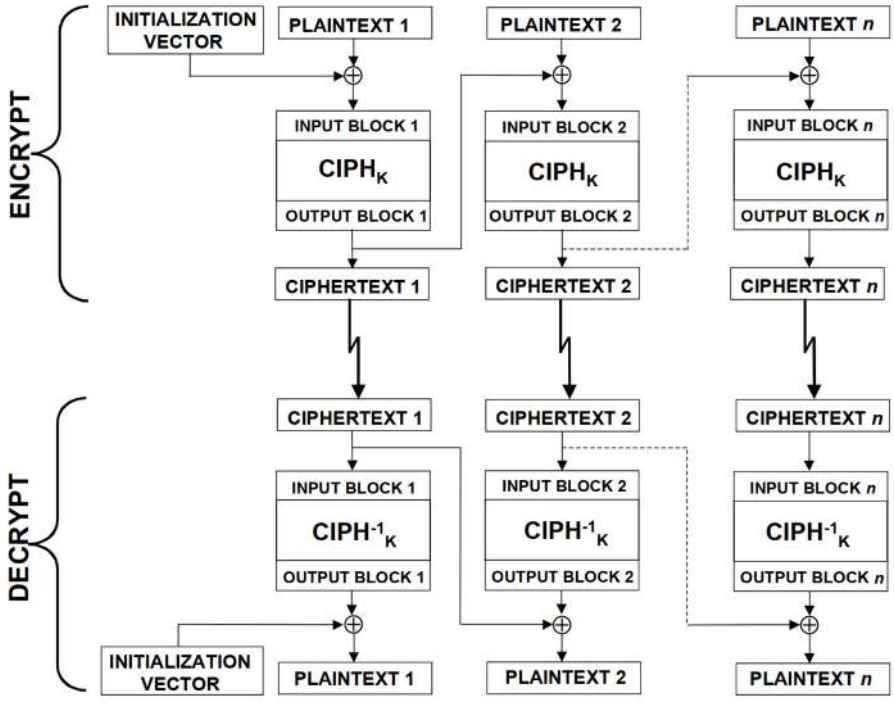
- Nhược điểm:

+

Khả năng bảo mật kém. Do giá trị plaintext và ciphertext được ánh xạ 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.

3.2 Chế độ 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.



Hình 11 Chế độ mã hóa/ gải mã CBC

- 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.

- Nhược điểm:

+

Thiết kế phần cứng phức tạp hơn ECB.

+

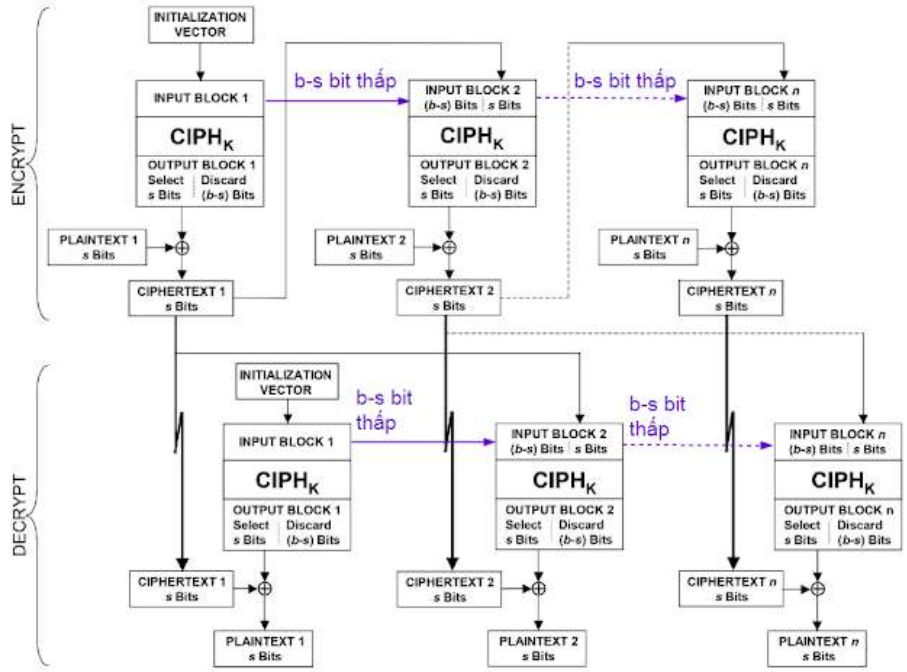
Lỗi bit bị lan truyền. Nếu một lỗi bit xuất hiện trên ciphertext của một khối dữ liệu thì nó sẽ làm sai kết quả giải mã của khối đữ liệu đó và khối dữ liệu tiếp theo.

+

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.

3.3 Chế độ 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 12 Chế độ mã hóa/ giải mã CFB

- 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
- Nhược điểm:
  +

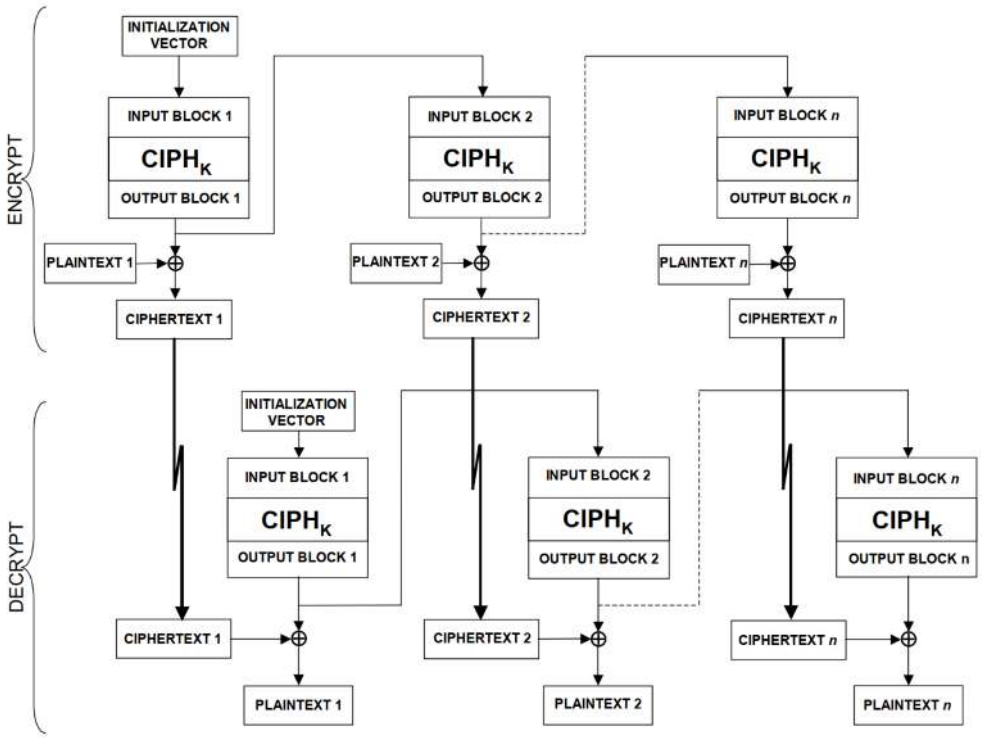
  Thiết kế phần cứng phức tạp hơn CBC. Ngoài những thành phần logic như CBC, CFB cần thêm logic để chọn số bit cần được xử lý nếu s là thông số cấu hình được.

+

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.

3.4 Chế độ 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 13 Chế độ mã hóa/ giải mã OFB

- 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.

- 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.

CHƯƠNG 4. ỨNG DỤNG VIẾT CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ AES SỬ DỤNG JAVA

4.1 Tổng quan chương trình mã hóa và giải mã AES

4.1.1 Định dạng PKCS7

PKCS là viết tắt của Public Key Cryptography Standards là một chuẩn do phòng thí nghiệm RSA Data Security Inc phát triển. Nó dựa vào các cấu trúc ASN.1 và thiết kế cho phù hợp với chứng chỉ X.09. PKCS#7 là một định dạng mã hóa cho việc lưu trữ một chứng chỉ số và chuỗi chứng nhận của nó dưới dạng các ký tự ASCII. Nó chủ yếu đươc sử dụng trên các nền tảng của Window và Java Tomcat. Hiện nay, khách hàng đang sử dụng CMS (Cryptographic Message Syntax), cũng giống như với SSL và TLS, PKCS#7 là tên thông thường mà tất cả chúng ta sử dụng và đang được thay thế. PKSC#7 có 2 định dạng mở rộng là: p7b và p7c. Không giống như PEM, PKCS#7 không thể lưu trữ khóa bảo mật (private key), nó chỉ có thể lưu trữ chứng chỉ gốc và chứng chỉ trung gian.

4.1.2 Cách mã hóa hoạt động

Mã hóa lấy văn bản thuần túy và chuyển nó thành dạng mật mã, làm cho văn bản trông giống như được tạo thành từ các ký tự ngẫu nhiên. Có thể an toàn khi nói rằng AES là một loại mã hóa đối xứng, vì nó sử dụng cùng một khóa để mã hóa và giải mã dữ liệu.

Tiêu chuẩn mã hóa này sử dụng thuật toán mạng hoán vị thay thế (thuật toán SPN) để áp dụng một số vòng mã hóa để bảo vệ dữ liệu. Thực tế là nó sử dụng quá nhiều vòng khiến AES hầu như không thể xuyên thủng .

4.1.3 AES nhiều độ dài khóa

AES bao gồm ba mật mã khối và mỗi mật mã khối này có một số tổ hợp phím có thể có khác nhau, như sau:

- AES-128: Độ dài khóa 128 bit = 3,4*10^38
- AES-192: Độ dài khóa 192 bit = 6.2*10^57
- AES-256: Độ dài khóa 256 bit = 1,1*10^77

Mặc dù có ba mật mã khối, mỗi một trong số chúng mã hóa và giải mã dữ liệu ở các bit 128 khối bằng cách sử dụng các độ dài khóa khác nhau (tức là 128, 192 và 256, như được chỉ định ở trên). Vì vậy, có thể an toàn khi nói rằng mặc dù độ dài của các khóa có thể khác nhau, nhưng kích thước khối luôn giống nhau (128 bit hoặc 16 byte, chúng giống nhau).

Mặc dù thực tế là khóa AES 256 bit chỉ đơn giản là khóa mạnh nhất trong nhóm, đôi khi được gọi là "cấp quân sự", nó không phải lúc nào cũng được triển khai theo mặc định và lý do tại sao điều này xảy ra là do tài nguyên có sẵn hoặc tốt hơn là không có.

Ở trong đề tài này ứng với mỗi chế độ mã hóa thì chương trình AES đều có thể chuyển đổi độ dài khóa vào ứng với 128 bit hoặc 256 bit tùy cấu hình đầu vào ta thiết lặp.

4.1.4 Mô tả thuật toán

Mặc dù 2 tên AES và Rijndael vẫn thường được gọi thay thế cho nhau nhưng trên thực tế thì 2 thuật toán không hoàn toàn giống nhau. AES chỉ làm việc với các khối dữ liệu (đầu vào và đầu ra) 128 bít và khóa có độ dài 128, 192 hoặc 256 bít trong khi Rijndael có thể làm việc với dữ liệu và khóa có độ dài bất kỳ là bội số của 32 bít nằm trong khoảng từ 128 tới 256 bít. Các khóa con sử dụng trong các chu trình được tạo ra bởi quá trình tạo khóa con Rijndael. Mỗi khóa con cũng là một cột gồm 4 byte. Hầu hết các phép toán trong thuật toán AES đều thực hiện trong một trường hữu hạn của các byte. Mỗi khối dữ liệu 128 bit đầu vào được chia thành 16 byte (mỗi byte 8 bit), có thể xếp thành 4 cột, mỗi cột 4 phần tử hay là một ma trận 4x4 của các byte,nó được gọi là ma trận trạng thái, hay vắn tắt là trạng thái (tiếng Anh: state, trạng thái trong Rijndael có thể có thêm cột). Trong quá trình thực hiện thuật toán các toán tử tác động để biến đổi ma trận trạng thái này.

- Qúa trình mã hóa

  1)

  Khởi động vòng lặp

  1. AddRoundKey - Mỗi cột của trạng thái đầu tiên lần lượt được kết hợp với một khóa con theo thứ tự từ đầu dãy khóa.

  2)

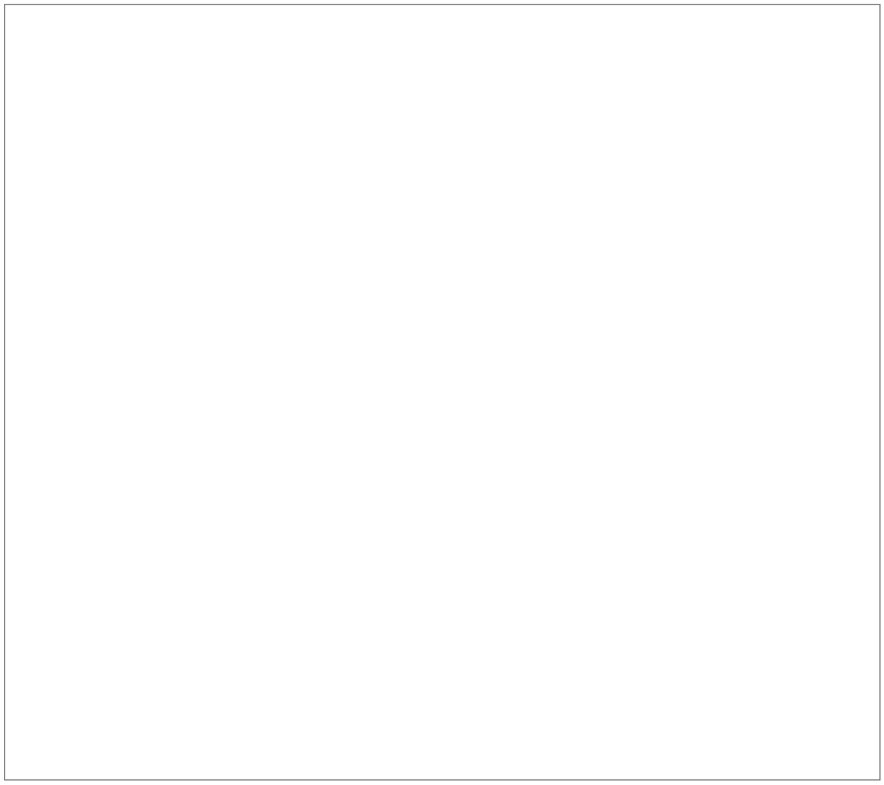
  Vòng lặp

1. SubBytes — đây là phép thế (phi tuyến) trong đó mỗi byte trong trạng thái sẽ được thế bằng một byte khác theo bảng tra (Rijndael S-box).
2. ShiftRows — dịch chuyển, các hàng trong trạng thái được dịch vòng theo số bước khác nhau.
3. MixColumns — quá trình trộn làm việc theo các cột trong khối theo một phép biến đổi tuyến tính.
4. AddRoundKey

3)

Vòng lặp cuối

1. SubBytes
2. ShiftRows
3. AddRoundKey



Hình 14 : Mô hình thuật toán mã hóa và giải mã của AES với khóa độ dài 128 bit

Tại chu trình cuối thì bước MixColums không thực hiện. Nr = 10, 12 hoặc 14 là số vòng lặp ứng với từng loại khóa 128, 192 và 256.

- Mô tả các bước:

1. Bước SubBytes: các byte được thế thông qua bảng tra S-box, đây chính là quá trình phi tuyến của thuật toán, hộp S-box này được tạo ra từ một phép biến đổi khả nghịch trong trường hữu hạn GF (28) có tính chất phi tuyến, để chống lại các tấn công dựa trên các đặc tính đại số, hộp S-box này được tạo nên bằng cách kết hợp phép nghịch đảo với một phép biến đổi affine khả nghịch, hộp S-box này cũng được chọn để tránh các điểm bất động (fixed point).
2. Bước ShiftRows: các hàng được dịch vòng một số bước nhất định, đối với AES, hàng đầu được giữ nguyên, mỗi byte của hàng thứ 2 được dịch vòng trái một vị trí, tương tự, các hàng thứ 3 và 4 được dịch vòng 2 và 3 vị trí, do vậy, mỗi cột khối đầu ra của bước này sẽ bao gồm các byte ở đủ 4 cột khối đầu vào, đối với Rijndael với độ dài khối khác nhau thì số vị trí dịch chuyển cũng khác nhau.
3. Bước MixColumns: bốn byte trong từng cột được kết hợp lại theo một phép biến đổi tuyến tính khả nghịch, mỗi khối 4 byte đầu vào sẽ cho một khối 4 byte ở đầu ra với tính chất là mỗi byte ở đầu vào đều ảnh hưởng tới cả bốn byte đầu ra, cùng với bước ShiftRows, MixColumns đã tạo ra tính chất khuếch tán cho thuật toán. Mỗi cột được xem như một đa thức trong trường hữu hạn và được nhân với đa thức c(x)=3x^{3}+x^{2}+x+2 (modul x^{4}+1, vì thế bước này có thể được xem là phép nhân ma trận trong trường hữu hạn.
4. Bước AddRoundKey: tại bước này, khóa con được kết hợp với các khối, khóa con trong mỗi chu trình được tạo ra từ khóa chính với quá trình tạo khóa con Rijndael, mỗi khóa con có độ dài giống như các khối, quá trình kết hợp được thực hiện bằng cách XOR từng bít của khóa con với khối dữ liệu.

4.2 Hàm mã hóa Encrypt AES

```
private void encryptBlock(byte[] in, int inOff, byte[] out, int outOff, int[][] KW)
  {
    int C0 = Pack.littleEndianToInt(in, inOff +  0);
    int C1 = Pack.littleEndianToInt(in, inOff +  4);
```

```
 int C2 = Pack.littleEndianToInt(in, inOff +  8);
int C3 = Pack.littleEndianToInt(in, inOff + 12);
int t0 = C0 ^ KW[0][0];
int t1 = C1 ^ KW[0][1];
int t2 = C2 ^ KW[0][2];
int r = 1, r0, r1, r2, r3 = C3 ^ KW[0][3];
while (r < ROUNDS - 1)
 {
        r0 = T0[t0&255] ^ shift(T0[(t1>>8)&255], 24) ^ shift(T0[(t2>>16)&255], 16) ^ shift(T0[(r3>>24)&255], 8) ^ KW[r][0];
        r1 = T0[t1&255] ^ shift(T0[(t2>>8)&255], 24) ^ shift(T0[(r3>>16)&255], 16) ^ shift(T0[(t0>>24)&255], 8) ^ KW[r][1];
        r2 = T0[t2&255] ^ shift(T0[(r3>>8)&255], 24) ^ shift(T0[(t0>>16)&255], 16) ^ shift(T0[(t1>>24)&255], 8) ^ KW[r][2];
        r3 = T0[r3&255] ^ shift(T0[(t0>>8)&255], 24) ^ shift(T0[(t1>>16)&255], 16) ^ shift(T0[(t2>>24)&255], 8) ^ KW[r++][3];
        t0 = T0[r0&255] ^ shift(T0[(r1>>8)&255], 24) ^ shift(T0[(r2>>16)&255], 16) ^ shift(T0[(r3>>24)&255], 8) ^ KW[r][0];
        t1 = T0[r1&255] ^ shift(T0[(r2>>8)&255], 24) ^ shift(T0[(r3>>16)&255], 16) ^ shift(T0[(r0>>24)&255], 8) ^ KW[r][1];
        t2 = T0[r2&255] ^ shift(T0[(r3>>8)&255], 24) ^ shift(T0[(r0>>16)&255], 16) ^ shift(T0[(r1>>24)&255], 8) ^ KW[r][2];
        r3 = T0[r3&255] ^ shift(T0[(r0>>8)&255], 24) ^ shift(T0[(r1>>16)&255], 16) ^ shift(T0[(r2>>24)&255], 8) ^ KW[r++][3];
 }
r0 = T0[t0&255] ^ shift(T0[(t1>>8)&255], 24) ^ shift(T0[(t2>>16)&255], 16) ^ shift(T0[(r3>>24)&255], 8) ^ KW[r][0];
r1 = T0[t1&255] ^ shift(T0[(t2>>8)&255], 24) ^ shift(T0[(r3>>16)&255], 16) ^ shift(T0[(t0>>24)&255], 8) ^ KW[r][1];
r2 = T0[t2&255] ^ shift(T0[(r3>>8)&255], 24) ^ shift(T0[(t0>>16)&255], 16) ^ shift(T0[(t1>>24)&255], 8) ^ KW[r][2];
r3 = T0[r3&255] ^ shift(T0[(t0>>8)&255], 24) ^ shift(T0[(t1>>16)&255], 16) ^ shift(T0[(t2>>24)&255],
```

```
8) ^ KW[r++][3];

            C0 = (S[r0&255]&255) ^ ((S[(r1>>8)&255]&255)<<8) ^ ((s[(r2>>16)&255]&255)<<16) ^
(s[(r3>>24)&255]<<24) ^ KW[r][0];

            C1 = (s[r1&255]&255) ^ ((S[(r2>>8)&255]&255)<<8) ^ ((S[(r3>>16)&255]&255)<<16) ^
(s[(r0>>24)&255]<<24) ^ KW[r][1];

            C2 = (s[r2&255]&255) ^ ((S[(r3>>8)&255]&255)<<8) ^ ((S[(r0>>16)&255]&255)<<16) ^
(S[(r1>>24)&255]<<24) ^ KW[r][2];

            C3 = (s[r3&255]&255) ^ ((s[(r0>>8)&255]&255)<<8) ^ ((s[(r1>>16)&255]&255)<<16) ^
(S[(r2>>24)&255]<<24) ^ KW[r][3];

    Pack.intToLittleEndian(C0, out, outOff +  0);

    Pack.intToLittleEndian(C1, out, outOff +  4);

    Pack.intToLittleEndian(C2, out, outOff +  8);

     Pack.intToLittleEndian(C3, out, outOff + 12);

  }
```

Hình 15 : encryptBlock

4.3 Hàm giải mã Decrypt AES

```
private void decryptBlock(byte[] in, int inOff, byte[] out, int outOff, int[][] KW)
   {
      int C0 = Pack.littleEndianToInt(in, inOff +  0);
      int C1 = Pack.littleEndianToInt(in, inOff +  4);
      int C2 = Pack.littleEndianToInt(in, inOff +  8);
     int C3 = Pack.littleEndianToInt(in, inOff + 12);
      int t0 = C0 ^ KW[ROUNDS][0];
      int t1 = C1 ^ KW[ROUNDS][1];
      int t2 = C2 ^ KW[ROUNDS][2];
     int r = ROUNDS - 1, r0, r1, r2, r3 = C3 ^ KW[ROUNDS][3];
      while (r > 1)
       {
                r0 = Tinv0[t0&255] ^ shift(Tinv0[(r3>>8)&255], 24) ^ shift(Tinv0[(t2>>16)&255], 16) ^
shift(Tinv0[(t1>>24)&255], 8) ^ KW[r][0];
                r1 = Tinv0[t1&255] ^ shift(Tinv0[(t0>>8)&255], 24) ^ shift(Tinv0[(r3>>16)&255], 16) ^
shift(Tinv0[(t2>>24)&255], 8) ^ KW[r][1];
                r2 = Tinv0[t2&255] ^ shift(Tinv0[(t1>>8)&255], 24) ^ shift(Tinv0[(t0>>16)&255], 16) ^
shift(Tinv0[(r3>>24)&255], 8) ^ KW[r][2];
                r3 = Tinv0[r3&255] ^ shift(Tinv0[(t2>>8)&255], 24) ^ shift(Tinv0[(t1>>16)&255], 16) ^
 shift(Tinv0[(t0>>24)&255], 8) ^ KW[r--][3];
                t0 = Tinv0[r0&255] ^ shift(Tinv0[(r3>>8)&255], 24) ^ shift(Tinv0[(r2>>16)&255], 16) ^
shift(Tinv0[(r1>>24)&255], 8) ^ KW[r][0];
                t1 = Tinv0[r1&255] ^ shift(Tinv0[(r0>>8)&255], 24) ^ shift(Tinv0[(r3>>16)&255], 16) ^
shift(Tinv0[(r2>>24)&255], 8) ^ KW[r][1];
                t2 = Tinv0[r2&255] ^ shift(Tinv0[(r1>>8)&255], 24) ^ shift(Tinv0[(r0>>16)&255], 16) ^
shift(Tinv0[(r3>>24)&255], 8) ^ KW[r][2];
                r3 = Tinv0[r3&255] ^ shift(Tinv0[(r2>>8)&255], 24) ^ shift(Tinv0[(r1>>16)&255], 16) ^
 shift(Tinv0[(r0>>24)&255], 8) ^ KW[r--][3];
       }
            r0 = Tinv0[t0&255] ^ shift(Tinv0[(r3>>8)&255], 24) ^ shift(Tinv0[(t2>>16)&255], 16) ^
```

```
shift(Tinv0[(t1>>24)&255], 8) ^ KW[r][0];

        r1 = Tinv0[t1&255] ^ shift(Tinv0[(t0>>8)&255], 24) ^ shift(Tinv0[(r3>>16)&255], 16) ^
shift(Tinv0[(t2>>24)&255], 8) ^ KW[r][1];

        r2 = Tinv0[t2&255] ^ shift(Tinv0[(t1>>8)&255], 24) ^ shift(Tinv0[(t0>>16)&255], 16) ^
shift(Tinv0[(r3>>24)&255], 8) ^ KW[r][2];

        r3 = Tinv0[r3&255] ^ shift(Tinv0[(t2>>8)&255], 24) ^ shift(Tinv0[(t1>>16)&255], 16) ^
shift(Tinv0[(t0>>24)&255], 8) ^ KW[r][3];


        C0 = (Si[r0&255]&255) ^ ((s[(r3>>8)&255]&255)<<8) ^ ((s[(r2>>16)&255]&255)<<16) ^
(Si[(r1>>24)&255]<<24) ^ KW[0][0];

        C1 = (s[r1&255]&255) ^ ((s[(r0>>8)&255]&255)<<8) ^ ((Si[(r3>>16)&255]&255)<<16) ^
(s[(r2>>24)&255]<<24) ^ KW[0][1];

       C2 = (s[r2&255]&255) ^ ((Si[(r1>>8)&255]&255)<<8) ^ ((Si[(r0>>16)&255]&255)<<16) ^
(s[(r3>>24)&255]<<24) ^ KW[0][2];

        C3 = (Si[r3&255]&255) ^ ((s[(r2>>8)&255]&255)<<8) ^ ((s[(r1>>16)&255]&255)<<16) ^
(s[(r0>>24)&255]<<24) ^ KW[0][3];

    Pack.intToLittleEndian(C0, out, outOff +  0);

    Pack.intToLittleEndian(C1, out, outOff +  4);

    Pack.intToLittleEndian(C2, out, outOff +  8);

     Pack.intToLittleEndian(C3, out, outOff + 12);

  }
```

Hình 16 : decryptBlock

4.4 Chế độ mã hóa

4.4.1 Chế độ CBC

4.4.1.1 Mã hóa CBC

```
private int encryptBlock( byte[] in, int inOff, byte[] out, int outOff) throws DataLengthException,
IllegalStateException {

    if ((inOff + blockSize) > in.length)

     {

       throw new DataLengthException("input buffer too short");

     }
```

```
    for (int i = 0; i < blockSize; i++)
     {
        cbcV[i] ^= in[inOff + i];
     }
     int length = cipher.processBlock(cbcV, 0, out, outOff);
    System.arraycopy(out, outOff, cbcV, 0, cbcV.length);
     return length;
  }
```

Hình 17 : encryptBlock CBC

4.4.1.2 Giải mã CBC

```
private int decryptBlock( byte[] in, int inOff, byte[] out, int outOff) throws DataLengthException, IllegalStateException {
    if ((inOff + blockSize) > in.length) {
        throw new DataLengthException("input buffer too short"); }
    System.arraycopy(in, inOff, cbcNextV, 0, blockSize);
    int length = cipher.processBlock(in, inOff, out, outOff);
    for (int i = 0; i < blockSize; i++) {
        out[outOff + i] ^= cbcV[i]; }
    byte[] tmp;
    tmp = cbcV;
    cbcV = cbcNextV;
    cbcNextV = tmp;
    return length;
}
```

Hình 18 : decryptBlock CBC

4.4.2 Chế độ CFB

4.4.2.1 Mã hóa CFB

```
private byte encryptByte(byte in) {
    if (byteCount == 0) {
        cipher.processBlock(cfbV, 0, cfbOutV, 0); }
    byte rv = (byte)(cfbOutV[byteCount] ^ in);
    inBuf[byteCount++] = rv;
    if (byteCount == blockSize) {
        byteCount = 0;
        System.arraycopy(cfbV, blockSize, cfbV, 0, cfbV.length - blockSize);
        System.arraycopy(inBuf, 0, cfbV, cfbV.length - blockSize, blockSize); }
    return rv;
}
```

```
public int encryptBlock( byte[] in, int inOff, byte[] out, int outOff) throws DataLengthException, IllegalStateException {
    processBytes(in, inOff, blockSize, out, outOff);
    return blockSize;
  }
```

Hình 19 : encryptByte

4.4.2.2 Giải mã CFB

```
private byte decryptByte(byte in) {
    if (byteCount == 0){
      cipher.processBlock(cfbV, 0, cfbOutV, 0)  }
    inBuf[byteCount] = in;
    byte rv = (byte)(cfbOutV[byteCount++] ^ in);
    if (byteCount == blockSize)  {
      byteCount = 0;
      System.arraycopy(cfbV, blockSize, cfbV, 0, cfbV.length - blockSize);
      System.arraycopy(inBuf, 0, cfbV, cfbV.length - blockSize, blockSize); }
    return rv; }
public int decryptBlock(byte[] in, int inOff, byte[] out, int outOff) throws DataLengthException, IllegalStateException {
    processBytes(in, inOff, blockSize, out, outOff);
    return blockSize;
  }
```

Hình 20 : decryptByte CFB

4.5 Tiến hành thực nghiệm

Chạy hàm run() với các kết quả thông số được gán trực tiếp như sau

```
void run() {
        byte[] key =  new byte[] {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
        byte[] dat =  "pham hieu dep trai vo dich".getBytes(StandardCharsets.UTF_8);
       byte[] iv =  new byte[] {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
        Mode mode = Mode.CBC;
        byte[] output = encryptMode(mode, dat, key, iv);
       decryptMode(mode, output, key, iv);
    }
```

Hình 21 : hàm run()

Kết quả thu được theo chế độ mã hóa CBC:

```
===== Mã hóa CBC =====
[-] Duration encrypt: 31700 nano second
[.] Padding data : 0x7068616d2068696575206465702074726169207 66f20646963680606060606060606
[.] Encrypt data output: 0x8c113445a33d46146293c7d916aefbcb2c5f2ad313bfb749509ee3ff5e0cc06d

===== Giải mã CBC =====
[-] Duration decrypt: 28100 nano second
[.] Decrypt data output: 0x7068616d2068696575206465702074726169207 66f2064696368
[.] String result: pham hieu dep trai vo dich
```

Hình 22: Kết quả thực nghiệm

Kết quả thực nghiệm đáp ứng đủ các yêu cầu ra của đề tài.

CHƯƠNG 5. CÁC DẠNG TẤN CÔNG VÀO AES VÀ PHƯƠNG PHÁP PHÒNG CHỐNG

5.1 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.

5.2 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.

5.3 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.

5.4 Kết luận và đánh giá thuật toán

- Thiết kế và độ dài khóa của thuật toán AES (128, 192 và 256 bit) là đủ an toàn để bảo vệ các thông tin được xếp vào loại tối mật nhưng về an ninh của AES thì các nhà khoa học đánh giá là chưa cao. Nếu các kỹ thuật tấn công được cải thiện thì AES có thể bị phá vỡ.
- Một vấn đề khác nữa là cấu trúc toán học của AES khá đơn giản.

TÀI LIỆU THAM KHẢO

[1] Trần Long, *Tìm hiểu về tiêu chuẩn mã hóa nâng cao AES*. Available: https://cyberlances.wordpress.com/2020/05/05/tim-hieu-ve-tieu-chuan-ma-hoa-nang-cao-aes/, truy cập ngày: 4/12/2021

[2] Bùi Quy Hoạt, *Tìm hiểu thuật toán mã hóa khóa đối xưng AES*. Available: https://viblo.asia/p/tim-hieu-thuat-toan-ma-hoa-khoa-doi-xung-aes-gAm5yxOqldb, truy cập ngày: 4/12/2021

[3] Đặng Thị Thu Hương, *Tiêu chuẩn mã hóa dữ liệu AES là gì và các chế độ hoạt động của AES phần 1*. Available: https://viettelidc.com.vn/tin-tuc/tieu-chuan-ma-hoa-du-lieu-aes-la-gi-va-cac-che-do-hoat-dong-cua-aes-phan-1, truy cập ngày: 4/12/2021

[4] Nguyễn Quân, *Các chế độ mã hóa và giải mã*. Available: https://nguyenquanicd.blogspot.com/2019/10/aes-bai-6-cac-che-o-ma-hoa-va-giai-ma.html, Truy cập ngày: 4/12/2021

[5] Phạm Hoàng Anh, *Cấu trúc và thuật toán Advanced Encryption Standard (Chuẩn mã hóa nâng cao)*. Available: https://viblo.asia/p/cau-truc-va-thuat-toan-advanced-encryption-standard-chuan-ma-hoa-nang-cao-924lJYe8ZPM, truy cập ngày 4/12/2021