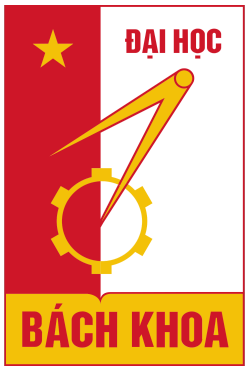
ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG ĐIỆN- ĐIỆN TỬ



BÁO CÁO MÔN HỌC
THÔNG TIN DI ĐỘNG

Đề tài: Nghiên cứu phương pháp bảo mật dữ liệu bằng hệ mật mã AES. Phát triển phần mềm truyền dữ liệu bảo mật

Nhóm 9

Nhóm sinh viên thực hiện : **Phí Mạnh Toàn** **20193142**
Nguyễn Hữu Linh **20192966**
Vũ Mạnh Hà **20192817**
Hoàng Anh Tú **20172884**

Giáo viên hướng dẫn : **PGS. TS Nguyễn Văn Đức**

Hà Nội, 1 - 2023

LỜI NÓI ĐẦU

Môn học Thông tin di động là một môn chuyên ngành quan trọng trong chương trình đào tạo kĩ sư ngành Điện tủ - Viễn thông. Môn học chứa đựng rất nhiều kiến thức từ nguồn gốc, sự phát triển cho tới nguyên lí hoạt động, ứng dụng của truyền thông tin di dộng nói riêng và lĩnh vực truyền thông nói chung. Những kiến thức đó đã giúp cho hành trang trước khi tốt nghiệp của chúng em trở nên hoàn thiện hơn, đồng thời mang đến một cái nhìn, định hướng rõ nét và sâu sắc hơn về ngành hay việc xác định mục tiêu trong tương lai. Nếu nắm chắc được những kiễn thức trong bộ môn đồng thời tích cực tìm hiểu nghiên cứu về các vấn đề liên quan thì con đường học tập và làm việc trong tương lai tất yếu sẽ thuận lợi và khởi sắc.

Một điều quý giá hơn nữa là khi chúng em được học tập dưới sự truyền thụ và chỉ bảo tận tình của PGS.TS. Nguyễn Văn Đức. Sự tận tâm với sự nghiệp giảng dạy của thầy đã khiến những kiến thức chuyên sâu về bộ môn được truyền đạt một cách dễ hiểu nhất có thể. Hơn nữa, bên cạnh những kiến thức chuyên môn, Thầy vẫn không quên chỉa sẻ về những kinh nghiệm sống quý giá, hằng mong sinh viên sẽ trưởng thành hơn, tốt hơn mỗi ngày về cả tài lẫn đức.

Bản báo cáo này là phần thực hiện Bài tập lớn của môn học Thông tin di động. Chúng em lựa chọn đề tài " Nghiên cứu phương pháp bảo mật dữ liệu bằng khóa AES, phát triền phần mềm truyền dữ liệu bảo mât". Đây là một đề tài hay, và là cơ hội để nhóm chúng em có cơ hội làm việc với nhau, giải quyết những vấn đềliên quan tới chuyên ngành mình đang học. Chúng em vẫn mong và sẽ cố gắng hoàn thành bài thật tốt để những ngày tháng cuối cùng tại trường thêm phần trọn vẹn. Tuy nhiên, rất khó tránh khỏi những thiếu sót nên nhóm chúng em rất mong sự góp ý tận tình của thầy để bài làm cũng như kĩ năng của nhóm dần cải thiện hơn.

Chúng em xin chân thành cảm ơn thầy trong suốt quá trình học .

Mục lục

Danh sách hình vẽ

1 GIỚI THIỆU CHUNG VỀ HỆ MẬT AES

1.1 Lịch sử ra đời

Năm 1997 NIST bắt đầu kêu gọi tìm kiếm một giải pháp thay thế hệ DES, đặt tên là Advanced Encryption Standard: Yêu cầu kỹ thuật đưa ra như sau: Mã khối có kích thước là 128 bits và sử dụng 3 loại khóa khác nhau có kích thước 128, 192 và 256 bits; ngoài ra, AES là giải pháp mở, phục vụ rộng rãi công khai trên thế giới. NIST tổ chức các hội thảo hàng năm trên thế giới. Lần đầu vào tháng 8/1998, họ nhận được 21 thuật toán, chọn 15 thuật toán đáp ứng yêu cầu trên. Tháng 8/1999 , tại hội thảo diễn ra ở Rome (Ý), họ chọn được 5 trong số 15 ứng viên, gọi là MARS, RC6, Rijndael, Serpent, và Twofish. Tiếp đó, tại hội thảo lần thứ 3 vào tháng 10/2000, thuật toán Rijndael (đọc là Rain Doll) – thiết kế bởi các nhà nghiên cứu người Bỉ là Joan Daemen và Vincent Rijndael. Vào tháng 2/2001, NIST công bố bản nháp (draft) về chuẩn xử lý thông tin Liên Bang và công bố rộng rãi để nhận được các đóng góp. Tháng 10/2001, AES được công bố theo chuẩn FIPS 197.

AES là một hệ mật mã khối đối xứng được tạo ra để thay thế cho hệ mật DES và được đặt làm tiêu chuẩn, được phê duyệt cho nhiều ứng dụng thực tế. So với các mật mã khóa công khai như RSA, cấu trúc của AES và các hệ mật đối xứng khá phức tạp và không thể tấn công dễ dàng như các hệ mật khác.

1.2 Cơ sở toán học

1.2.1 Trường số học hữu hạn

Trong hệ mật AES, tất cả các phép toán, phép biến đổi đều được thực hiện trên đơn vị bytes (8-bits). Đặc biệt các phép toán học như phép cộng, nhân, chia được thực hiện trên trường hữu hạn ($GF(2^8)$).

Về bản chất, một trường số học là một tập hợp mà ta có thể thực hiện các phép toán cộng trừ nhân chia mà không cần rời khỏi tập hợp đó. Phép chia được xác định theo quy tắc sau: ($a/b = a(b^{-1})$). Ví dụ về trường số học hữu hạn: Z_p bao gồm các số nguyên từ $\{0, 1, ..., p-1\}$ trong đó p là một số nguyên tố và trong đó thực hiện phép tính số học *modulo* p.

Hầu như tất cả các thuật toán mã hóa, đối với cả khóa thông thường lẫn khóa công khai, đều liên quan đến các phép toán số học trên số nguyên. Nếu một trong các phép toán được sử dụng là phép chia, thì chúng ta cần làm việc với phép toán số học được xác định trong một trường, điều này là do phép chia yêu cầu mỗi phần tử khác không phải có một phép nhân nghịch đảo.

Có một cách xác định trường hữu hạn 2^n phần tử, một trường như vậy được gọi là $GF(2^n)$. Xét tập S gồm tất cả các đa thức bậc n-1 hoặc nhỏ hơn với các hệ số nhị phân. Như vậy mỗi đa thức có dạng:

$$\text{f(x)} = \text{a}_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

Với a_i nhận giá trị 0 hoặc 1. Trong tập hợp S có tổng cộng 2^n đa thức khác

nhau. Với n=3 thì $2^3 = 8$ đa thức trong tập hợp là:

$$\begin{array}{cccc} 0 & \mathrm{x} & \mathrm{x}^2 & \mathrm{x}^2+x \\ 1 & \mathrm{x}+1 & \mathrm{x}^2+1 & \mathrm{x}^2+x+1 \end{array}$$

Với định nghĩa của các phép toán số học đã nêu ra, mỗi tập S như vậy là một trường hữu hạn. Định nghĩa một trường số học S bao gồm các yếu tố sau:

- Số học tuân theo các quy tắc thông thường của số học đa thức bằng cách sử dụng các quy tắc cơ bản của đại số với 2 tinh chỉnh sau đây:
- Phép tính số học trên các hệ số được thực hiện theo modulo 2, giống như phép toán XOR.
- Khi phép nhân tạo ra một đa thức bậc lớn hơn $n-1$, đa thức đó sẽ được rút gọn theo modulo một số đa thức bất khả quy m(x) bậc n. Nghĩa là chúng ta chia cho m(x) và giữ nguyên phần dư. Đối với đa thức f(x), phần dư được biểu diễn dưới dạng $r(x) = f(x)\ mod\ m(x)$. Một đa thức m(x) được gọi là **bất khả quy** khi và chỉ khi m(x) không thể biểu diễn dưới dạng tích của hai đa thức, cả hai đều nhỏ hơn bậc của m(x).

Ví dụ để xây dựng trường hữu hạn $GF(2^3)$, ta cần chọn một đa thức bất khả quy bậc 3. Chỉ có hai đa thức như vậy: (x^3+x^2+1) và (x^3+x+1). Phép cộng tương đương với việc lấy XOR của các số hạng giống nhau. Như vậy $(x+1)+x = 1$.

Một đa thức trong $GF(2^n)$ có thể được biểu diễn duy nhất bởi n hệ số nhị phân của nó $(a_{n-1}\ a_{n-2}\ldots\ a_0)$. Do đó mọi đa thức trong $GF(2^n)$ có thể được biểu diễn bằng một số n-bit. Phép cộng được thực hiện bằng cách lấy phép XOR theo bit của hai phần tử n-bit. Không có phép toán XOR đơn giản nào có thể thực hiện phép nhân trong $GF(2^n)$. Tuy nhiên một kĩ thuật hợp lý đơn giản, dễ thực hiện có sẵn. Về bản chất có thể chỉ ra rằng phép nhân một số trong $GF(2^n)$ với 2 bao gồm một phép dịch trái theo sau là phép XOR có điều kiện với một hằng số. Phép nhân với số lớn hơn có thể đạt được bằng áp dụng lặp đi lặp lại quy tắc này.

Ví dụ, hệ mật mã AES thực hiện các phép toán trên trường số học hữu hạn GF(28), với đa thức bất khả quy là $m(x) = x^8+x^4+x^3+x+1$. Xét hai phần tử $A(a_7\ a_6\ldots a_1\ a_0)$ và $B(b_7\ b_6\ldots b_1\ b_0)$ có tổng A+B = (c7 c6 … c1 c0) với $ci = ai \oplus bi$. Phép nhân 02 . $A(a_6\ a_5\ldots a_0\ 0)$ nếu a_7 = 0, và bằng $(a_6\ a_5\ldots a_0\ 0)$ $\oplus$ (0 0 0 1 1 0 1 1) nếu a_7 =1.

2 CẤU TRÚC CHUNG CỦA HỆ MẬT AES

2.1 Cấu trúc chung

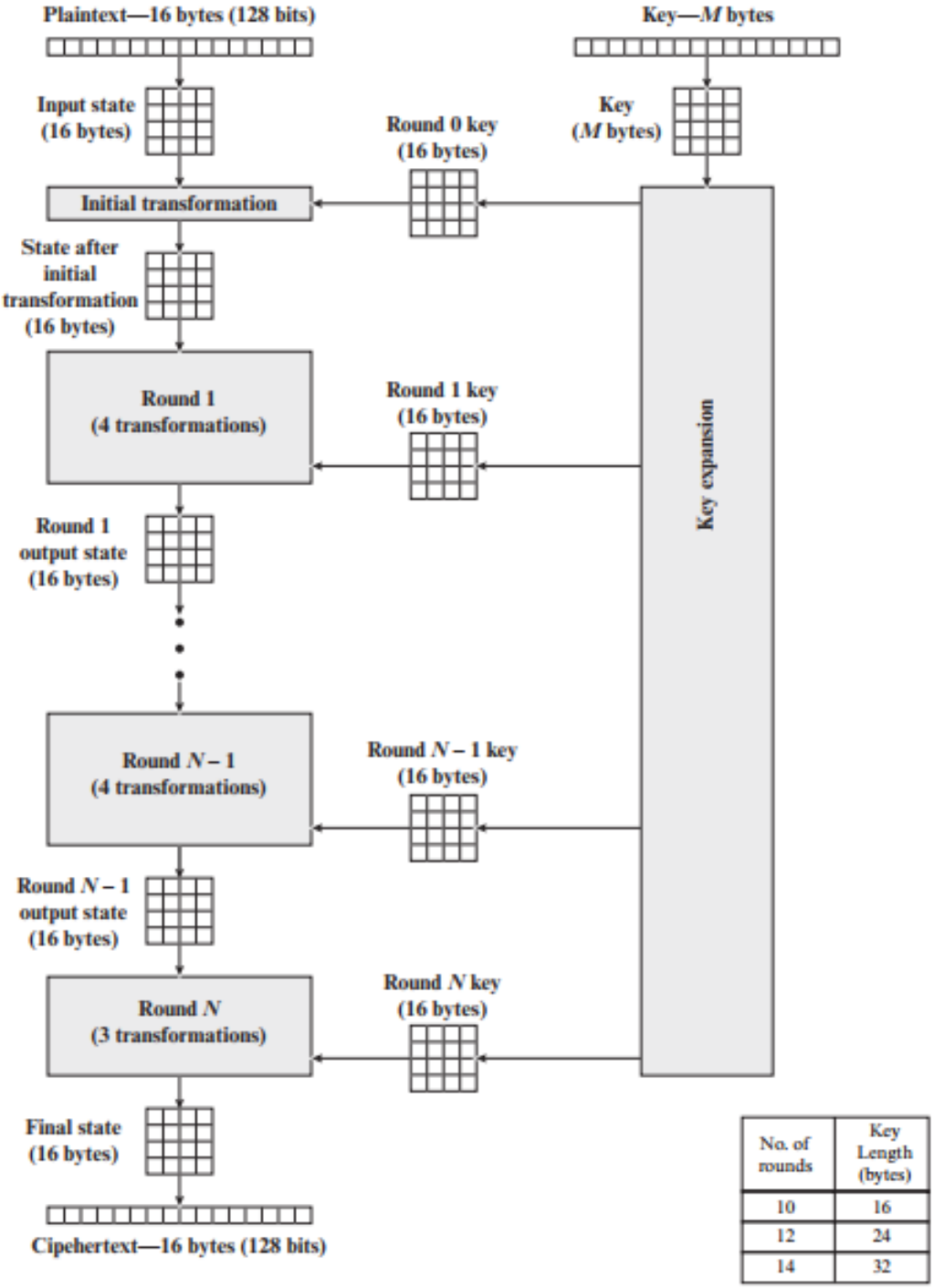
Hình 2.1 cho thấy cấu trúc tổng thể của quy trình mã hóa AES. Mật mã khối AES có kích thước bản rõ (plaintext) là 128 bits, hay 16 bytes. Độ dài của khóa (key) có thể là 16, 24, hoặc 32 bytes (128 bits, 192 bits, 256 bits). Thuật toán được gọi là AES-128, AES-192 hoặc AES-256 tùy thuộc vào độ dài của khóa.

Đầu vào của các thuật toán mã hóa và giải mã là một khối 128-bit duy nhất. Trong FIPS PUB 197, khối này được mô tả dưới dạng ma trận vuông 4x4. Khối này được sao chép vào mảng **Trạng thái** (State array), được sửa đổi ở mỗi giai đoạn mã hóa và giải mã. Sau giai đoạn cuối cùng, **Trạng thái** được sao chép vào ma trận đầu ra. Các thao tác này được mô tả trong Hình 6.2a. Tương tự, khóa được mô tả dưới dạng ma trận vuông bytes. Khóa này sau đó được mở rộng thành một mảng các khóa mở rộng. Hình 6.2b cho thấy sự mở rộng cho khóa 128-bit. Mỗi từ là bốn byte và tổng lịch trình khóa là 44 từ cho khóa 128-bit. Lưu ý rằng thứ tự của các byte trong một ma trận là theo cột. Vì vậy, ví dụ, bốn byte đầu tiên của đầu vào văn bản gốc 128-bit cho mật mã mã hóa chiếm cột đầu tiên của ma trận trong, bốn byte thứ hai chiếm cột thức hai,....Tương tự, bốn byte đầu tiên của khóa mở rộng tạo thành một từ, chiếm cột đầu tiên của ma trận **w**.

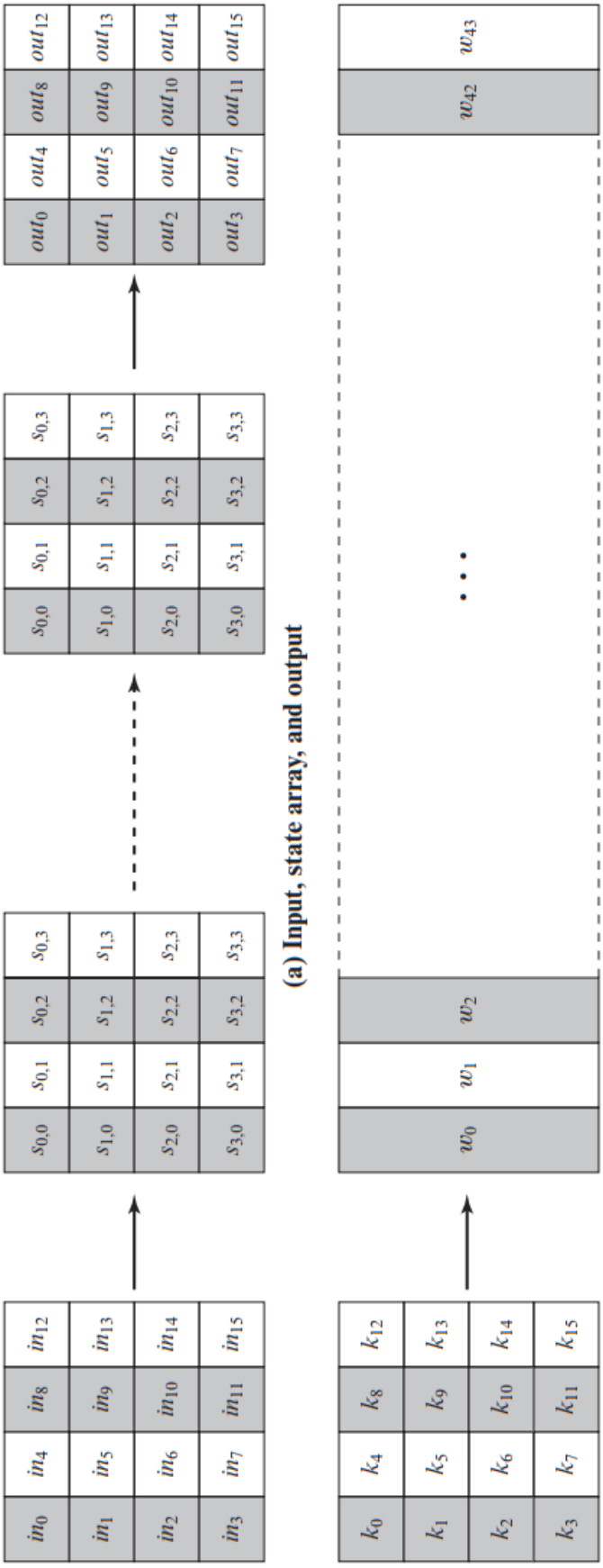
Mật mã bao gồm N vòng, trong đó số vòng phụ thuộc vào độ dài khóa: 10 vòng cho khóa 16 byte, 12 vòng cho khóa 24 byte và 14 vòng cho khóa 32 byte (Bảng 6.1). N - 1 vòng đầu tiên bao gồm bốn hàm chuyển đổi riêng biệt: **SubBytes**, **ShiftRows**, **MixColumns** và **AddRoundKey**, được mô tả sau đó. Vòng cuối cùng chỉ chứa ba phép biến đổi và có một phép biến đổi đơn đầu tiên **(AddRoundKey)** trước vòng đầu tiên, có thể được coi là Vòng 0. Mỗi phép biến đổi có một hoặc nhiều ma trận 4 * 4 làm đầu vào và tạo ma trận 4 * 4 làm đầu ra. Hình 6.1 cho thấy đầu ra của mỗi vòng là một ma trận 4 * 4, với đầu ra của vòng cuối cùng là bản mã. Ngoài ra, chức năng mở rộng khóa tạo ra N + 1 khóa vòng, mỗi khóa là một ma trận 4 * 4 riêng biệt. Mỗi phím vòng đóng vai trò là một trong các đầu vào cho phép chuyển đổi **AddRoundKey** trong mỗi vòng.

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

Bảng 2.1 Các thông số trong hệ mật AES



Hình 2.1 Quá trình mã hóa AES



Hình 2.2 Cấu trúc dữ liệu của hệ mật AES

2.2 Cấu trúc cụ thể

Hình 2.2a trình bày chi tiết hơn về mật mã AES, biểu thị trình tự các phép biến đổi trong mỗi vòng và hiển thị hàm giải mã tương ứng. Trước khi đi sâu vào chi tiết, chúng ta có thể đưa ra một số nhận xét về cấu trúc AES tổng thể.

1. Một đặc điểm đáng chú ý của cấu trúc này là nó không phải là cấu trúc **Feistel**. Nhớ lại rằng, trong cấu trúc **Feistel** cổ điển, một nửa khối dữ liệu được sử dụng để sửa đổi nửa khối dữ liệu còn lại và sau đó hai nửa được hoán đổi. Thay vào đó, AES xử lý toàn bộ khối dữ liệu dưới dạng một ma trận duy nhất trong mỗi vòng bằng cách thay thế và hoán vị.

2. Khóa được cung cấp làm đầu vào được mở rộng thành một mảng gồm bốn mươi bốn từ 32 bit, **w[i]**. Bốn từ riêng biệt (128-bit) đóng vai trò là khóa vòng cho mỗi vòng; những điều này được chỉ ra trong Hình 2.2.

3. Bốn giai đoạn khác nhau được sử dụng, một giai đoạn hoán vị và ba giai đoạn thay thế:

   - **Substitute bytes**: Sử dụng S-box để thực hiện thay thế khối theo từng byte.
   - **ShiftRows**: Một phép hoán vị đơn giản.
   - **MixColumns**: Một phép thay thế sử dụng số học trên trường $GF(2^8)$.
   - **AddRoundKey**: Phép XOR của khối hiện tại với một phần của khóa mở rộng **(expanded key)**.

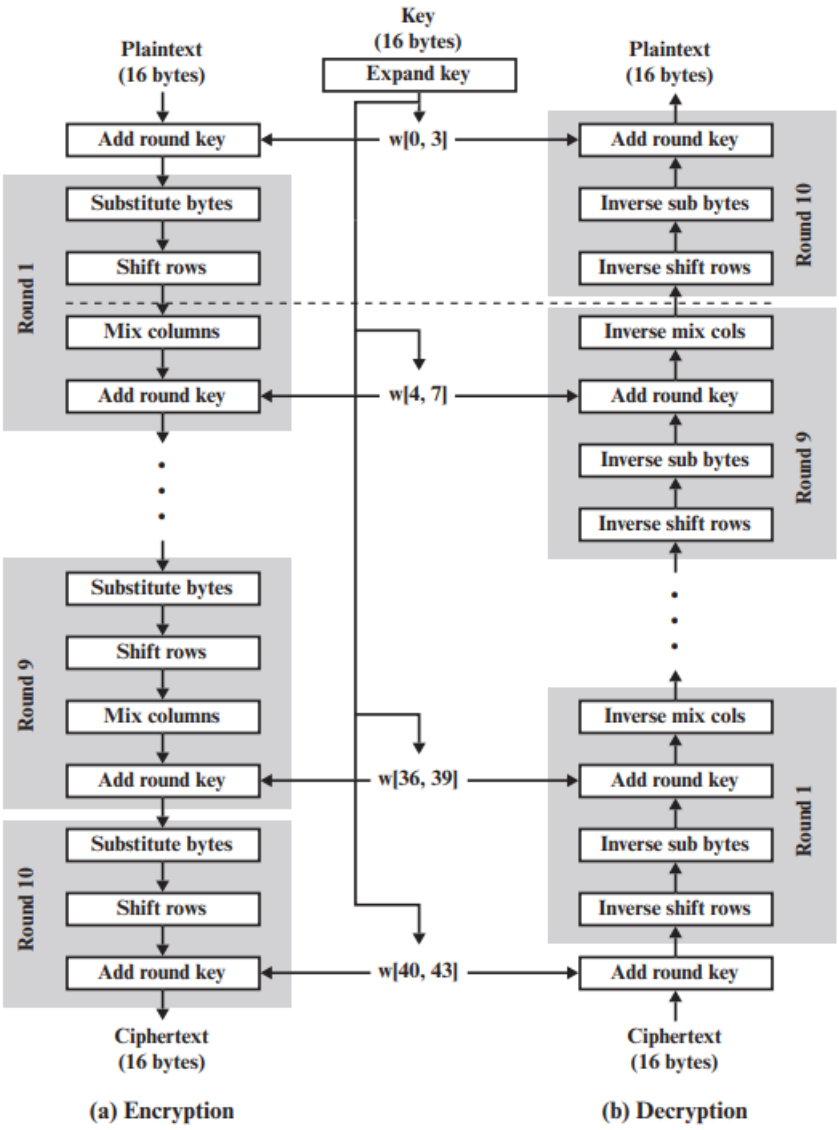
4. Cấu trúc khá đơn giản. Đối với cả mã hóa và giải mã, mật mã bắt đầu bằng một giai đoạn **AddRoundKey**, tiếp theo là chín vòng, mỗi vòng bao gồm tất cả bốn giai đoạn, tiếp theo là vòng thứ mười gồm ba giai đoạn. Hình 2.2b mô tả cấu trúc của một vòng mã hóa đầy đủ.

5. Chỉ giai đoạn **AddRoundKey** mới sử dụng khóa. Vì lý do này, mật mã bắt đầu và kết thúc bằng giai đoạn **AddRoundKey**. Bất kỳ giai đoạn nào khác, được áp dụng ở đầu hoặc cuối, đều có thể đảo ngược mà không cần biết về khóa và do đó sẽ không tăng thêm tính bảo mật.

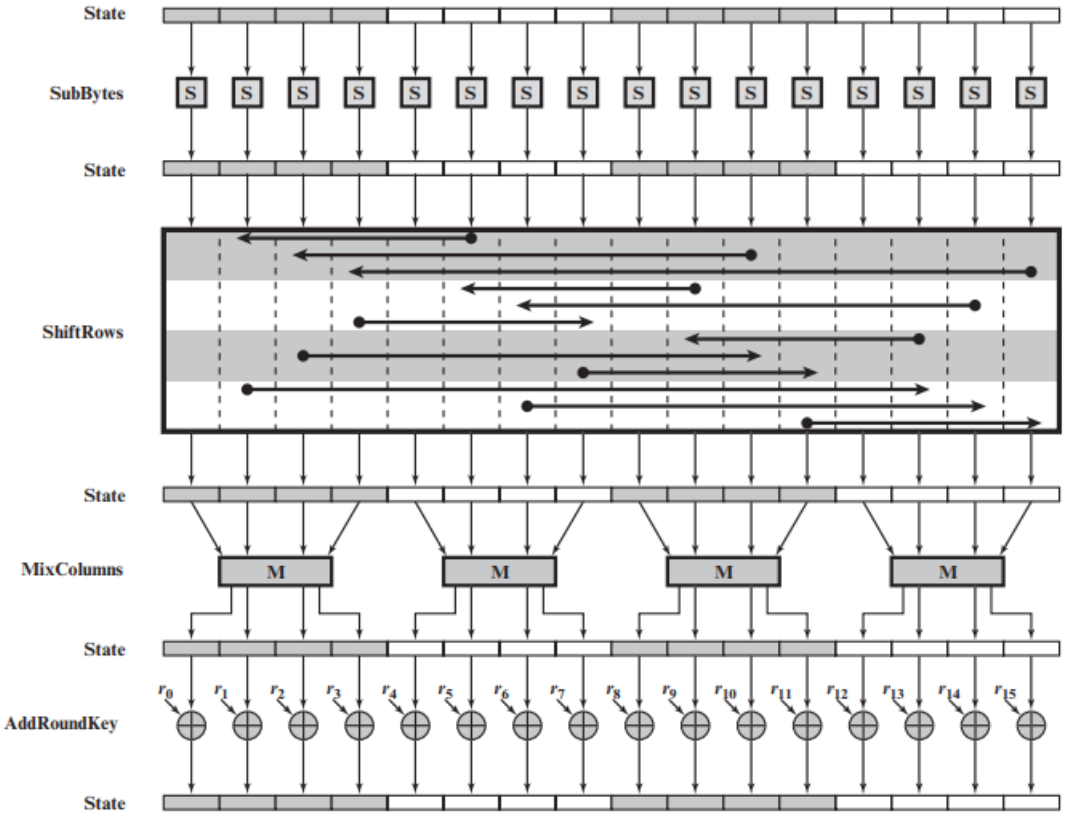
6. Trên thực tế, giai đoạn **AddRoundKey** là một dạng mật mã **Vernam** và bản thân nó sẽ không ghê gớm. Ba giai đoạn còn lại cùng nhau tạo ra sự nhầm lẫn, khuếch tán và phi tuyến tính, nhưng bản thân chúng sẽ không mang lại sự an toàn vì chúng không sử dụng khóa. Chúng ta có thể xem mật mã dưới dạng các hoạt động xen kẽ của mã hóa XOR **(AddRoundKey)** của một khối, theo sau là xáo trộn của khối (ba giai đoạn còn lại), tiếp theo là mã hóa XOR, v.v. Chương trình này vừa hiệu quả vừa có tính bảo mật cao.

7. Mỗi giai đoạn có thể dễ dàng đảo ngược. Đối với các giai đoạn Byte thay thế, **ShiftRows** và **MixColumns**, một hàm nghịch đảo được sử dụng trong thuật toán giải mã. Đối với giai đoạn **AddRoundKey**, nghịch đảo đạt được bằng cách XOR cùng một phím tròn cho khối, sử dụng kết quả là $A \oplus B \oplus B = A$.

8. Như với hầu hết các mật mã khối, thuật toán giải mã sử dụng khóa mở rộng theo thứ tự ngược lại. Tuy nhiên, thuật toán giải mã không giống với thuật toán mã hóa. Đây là hệ quả của cấu trúc đặc biệt của AES.
9. Khi đã xác định được rằng cả bốn giai đoạn đều có thể đảo ngược, thì dễ dàng xác minh rằng quá trình giải mã có khôi phục được bản rõ. Hình 2.2a trình bày quá trình mã hóa và giải mã theo các hướng ngược nhau. Tại mỗi điểm nằm ngang (ví dụ: đường đứt nét trong hình), Trạng thái giống nhau cho cả mã hóa và giải mã.
10. Vòng cuối cùng của cả mã hóa và giải mã bao gồm ba giai đoạn. Một lần nữa, đây là hệ quả của cấu trúc cụ thể của AES và được thiết kế để làm cho mật mã có thể đảo ngược.



Hình 2.3 Mã hóa và giải mã AES



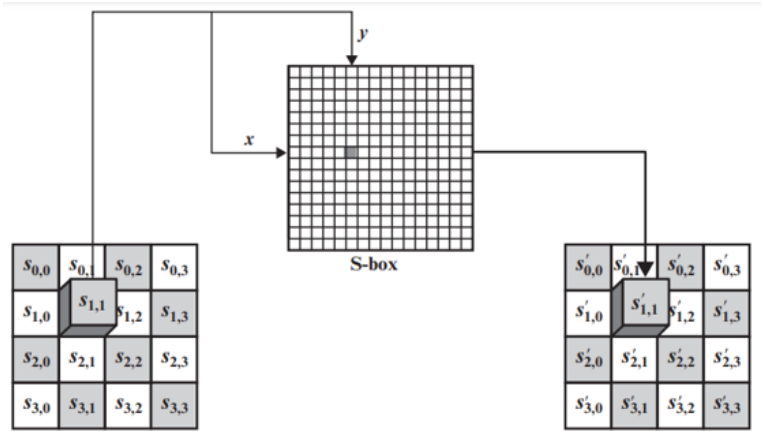
Hình 2.4 Vòng mã hóa AES

3 CÁC HÀM CHUYỂN ĐỔI

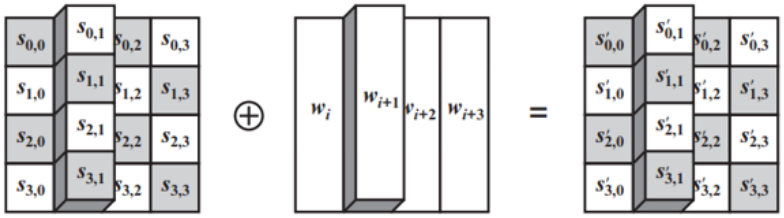
Bây giờ chúng ta chuyển sang thảo luận về bốn bước biến đổi được sử dụng trong AES. Với mỗi giai đoạn, chúng ta mô tả thuật toán chuyển tiếp **(encryption)**, và nghịch đảo **(decryption)**, và chức năng của mỗi giai đoạn.

3.1 Substitute Bytes Transformation

Biến đổi Byte thay thế thuận, được gọi là SubBytes, là một phép tra cứu bảng đơn giản. AES định nghĩa một ma trận 16x16 của các giá trị byte, được gọi là S-box, chứa một hoán vị của tất cả 256 giá trị 8 bit có thể. Mỗi byte trạng thái riêng lẻ được ánh xạ vào một byte mới theo cách sau: 4 bit ngoài cùng bên trái của byte là được sử dụng làm giá trị hàng và 4 bit ngoài cùng bên phải được sử dụng làm giá trị cột. những hàng này và các giá trị cột đóng vai trò là chỉ mục trong hộp S để chọn đầu ra 8 bit duy nhất giá trị. Ví dụ: giá trị thập lục phân 95 tham chiếu đến hàng 9, cột 5 của S-box, chứa giá trị $\{2A\}$. Theo đó, giá trị $\{95\}$ được ánh xạ vào giá trị $\{2A\}$.



Hình 3.5 Sub byte transformation



Hình 3.6 Add round key transformation

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

Hình 3.7 S-Box Table

x \ 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
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 3.8 Inv S-Box Table

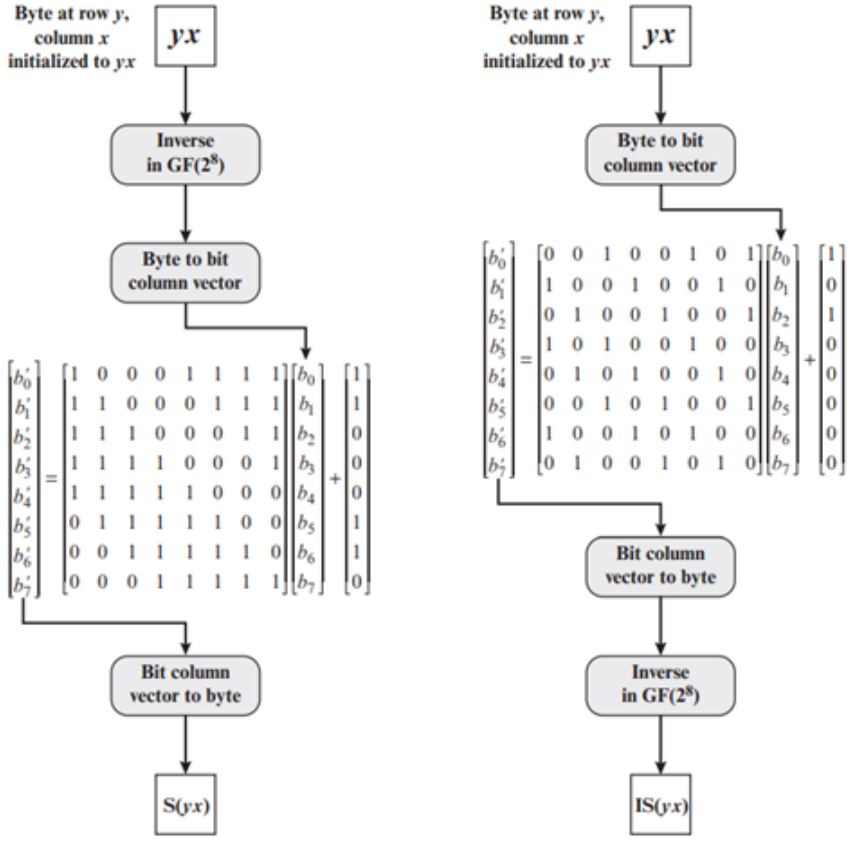
Dưới đây là một ví dụ của biến đổi **SubBytes**:

S-box được xây dựng như Hình 2.10, theo các bước như sau

1. Khởi tạo S-box với các giá trị byte theo thứ tự tăng dần theo từng hàng. Hàng đầu tiên chứa $\{00\}$, $\{01\}$, $\{02\}$, ... , $\{0F\}$; hàng thứ hai chứa $\{10\}$, $\{11\}$, v.v.. Như vậy, giá trị của byte tại hàng y, cột x là $\{yx\}$.
2. Ánh xạ từng byte trong S-box thành nghịch đảo nhân của nó trong trường hữu hạn $GF(2^8)$, giá trị $\{00\}$ được ánh xạ tới chính nó.
3. Xét rằng mỗi byte trong hộp S bao gồm 8 bit được đánh nhãn $(b_7, b_6, b_5, b_4, b_3, b_2,$ $b_1, b_0)$. Áp dụng phép biến đổi sau cho từng bit của từng byte trong hộp S: $b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$ trong đó c_i là bit thứ i của byte c có giá trị $\{63\}$; tức là $(c_7c_6c_5c_4c_3c_2c_1c_0 = (01100011)$. Số nguyên tố () chỉ ra rằng biến sẽ được cập nhật theo giá trị bên phải. Tiêu chuẩn AES mô tả sự chuyển đổi này ở dạng ma trận như Hình 2.11.

EA	04	65	85	→	87	F2	4D	97
83	45	5D	96		EC	6E	4C	90
5C	33	98	B0		4A	C3	46	E7
F0	2D	AD	C5		8C	D8	95	A6

Hình 3.9 Ví dụ về phép Sub Bytes



(a) Calculation of byte at row y, column x of S-box

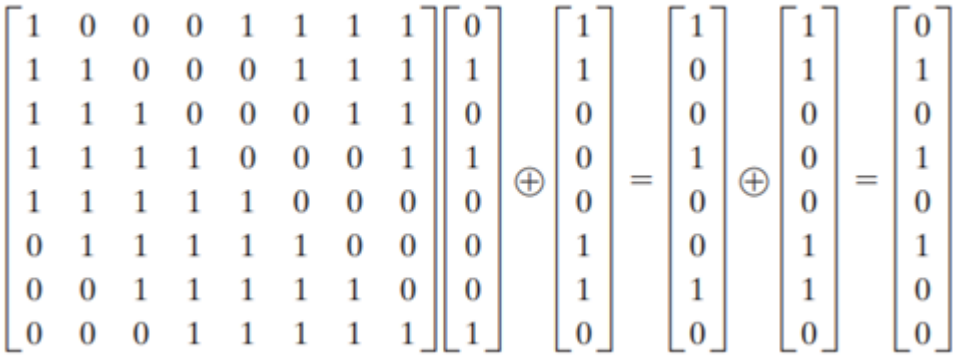
(a) Calculation of byte at row y, column x of IS-box

Hình 3.10 Cách xây dựng bảng S-box

Phương trình (Hình 2.11) phải được giải thích cẩn thận. Trong phép nhân ma trận thông thường, mỗi phần tử trong ma trận tích là tổng các tích của các phần tử của một hàng và một cột. Trong trường hợp này, mỗi yếu tố trong ma trận sản phẩm là XOR theo bit của các sản phẩm của các phần tử của một hàng và một cột. Hơn nữa, các phép cộng cuối cùng được chỉ ra trong phương trình (Hình 2.11) là phép XOR theo bit. (XOR bitwise ở phần trước).

Ví dụ, xem xét giá trị đầu vào $\{95\}$. Nghịch đảo phép nhân trong $GF(28)$ là $\{95\} - 1 = \{8A\}$, là 10001010 ở dạng nhị phân. Sử dụng phương trình (Hình 2.11).

Hình 3.11 Chuyển đổi S-box



Hình 3.12 Phép biến đổi ma trận S-box

Kết quả là $\{2A\}$, sẽ xuất hiện trong hàng $\{09\}$ cột $\{05\}$ của S-box. Điều này được xác minh bằng cách kiểm tra Bảng 2.6.

Biến đổi byte thay thế nghịch, được gọi là **InvSubBytes**, sử dụng của hộp S ngược thể hiện trong Bảng 2.7. Ví dụ: lưu ý rằng đầu vào $\{2A\}$ tạo ra đầu ra $\{95\}$ và đầu vào $\{95\}$ vào hộp S tạo ra $\{2A\}$. S-box nghịch được xây dựng (Hình2.7) bằng cách áp dụng nghịch đảo của phép biến đổi trong Phương trình (6.1) dưới đây bằng cách lấy nghịch đảo nhân trong GF($GF(2^8)$. Sự biến đổi nghịch đó là: $b'_i = b_{(i+2)mod8} \oplus b(i+5)mod8 \oplus b_{(i+7)mod8} \oplus d_i$

Tại byte d = $\{01\}$ hoặc 00000101. Chúng ta có thể mô tả phép biến đổi này như sau:

Hình 3.13 Phép biến đổi ma trận S-box

Để thấy rằng **InvSubBytes** là nghịch đảo của **SubBytes**, hãy gắn nhãn các ma trận trong **SubBytes** và **InvSubBytes** tương ứng là **X** và **Y**, và các dạng vectơ của hằng

số **c** và **d** tương ứng là **C** và **D**. Đối với một số vectơ **B** 8 bit, phương trình trở thành $B^{'} = XB \oplus C$. Ta cần chứng minh rằng $Y(XB \oplus C) \oplus D = B$. Để nhân ra, chúng ta phải chỉ ra $YXB \oplus YC \oplus D = B$ như sau:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

Chúng ta đã chứng minh rằng YX bằng ma trận đơn vị và YC = D, sao cho $YC \oplus D$ bằng với vectơ không.

(Cơ sở lí luận) S-box được thiết kế để chống lại các cuộc tấn công phân tích mật mã đã biết. Cụ thể, các nhà phát triển Rijndael đã tìm kiếm một thiết kế có mối tương quan thấp giữa các bit đầu vào và các bit đầu ra và thuộc tính mà đầu ra không phải là hàm toán học tuyến tính của đầu vào. Tính phi tuyến tính là do việc sử dụng nghịch đảo nhân. Ngoài ra, hằng số trong phương trình (6.1) đã được chọn sao cho hộp S không có điểm cố định [S-box(a) = a] và không có **"điểm cố định đối diện"** [S-box(a) = ā], trong đó ā là phần bù theo chiều bit của a.

Tất nhiên, S-box phải khả đảo, nghĩa là IS-box[S-box(a)] = a.Tuy nhiên, hộp S không tự nghịch đảo theo nghĩa là S-box(a) = IS-box(a). Ví dụ: S-box({95}) = {2*A*}, nhưng IS-Box({95}) = AD.

3.2 ShiftRows Transformation

Sự biến đổi dịch chuyển hàng, được gọi là **ShiftRows**, mô tả trong Hình 6.7a. Trạng thái của hàng đầu tiên không bị thay đổi. Từ hàng thứ hai, dịch trái 1 byte được thực hiện. Đối với hàng thứ ba, 2 byte dịch trái được thực hiện. Đối với hàng thứ tư, dịch trái 3 byte được thực hiện. Sau đây là một ví dụ về **ShiftRows**.

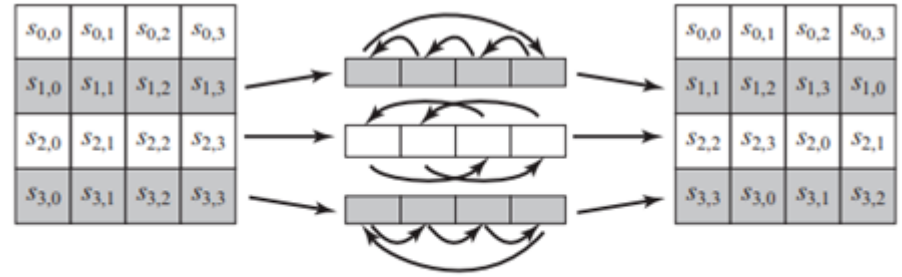
87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

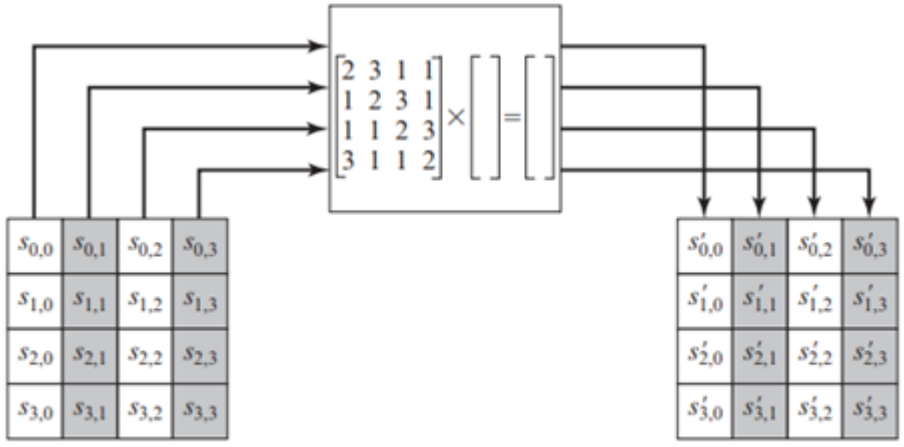
87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Hình 3.14 Phép biến ShiftRow

Phép biến đổi dịch hàng nghịch , được gọi là **InvShiftRows**, thực hiện dịch chuyển vòng tròn theo hướng ngược lại cho mỗi hàng trong số ba hàng cuối cùng, với độ dài 1 byte dịch sang phải cho hàng thứ hai, v.v.



(a) Shift row transformation



(b) Mix column transformation

Hình 3.15 Hoạt động của hàng và cột AES

Việc chuyển đổi hàng thay đổi là đáng kể hơn so với lần đầu tiên xuất hiện. Điều này là do Trạng thái, cũng như đầu vào và đầu ra của mật mã, được coi là một mảng gồm bốn cột 4 byte. Do đó, khi mã hóa, 4 byte đầu tiên của bản rõ được sao chép vào cột Trạng thái đầu tiên, v.v. Hơn nữa, như sẽ thấy, phím tròn được áp

dụng cho từng cột Trạng thái. Như vậy, một hàng shift di chuyển một byte riêng lẻ từ cột này sang cột khác, là khoảng cách tuyến tính của bội số 4 byte. Cũng lưu ý rằng việc chuyển đổi đảm bảo rằng 4 byte của một cột được trải ra cho bốn cột khác nhau. Hình 3.15 minh họa quá trình thực hiện:

3.3 MixColumns Transformation

Sự biến đổi trộn cột thuận,được gọi là **MixColumns**, hoạt động trên từng cột riêng lẻ. Mỗi byte của một cột được ánh xạ thành một giá trị mới là hàm của cả bốn byte trong cột đó. Các phép biến đổi có thể được xác định bằng phép nhân ma trận trên Trạng thái sau:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix} \quad \textbf{(6.3)}$$

Hình 3.16 Biến đổi MixColumns

Mỗi phần tử trong ma trận tích là tổng tích các phần tử của một hàng và một cột. Trong trường hợp này, các phép cộng và phép nhân riêng lẻ được thực hiện trong $GF(2^8)$. Biến đổi **MixColumns** trên một cột Trạng thái có thể được thể hiện như:

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

Hình 3.17 Trạng thái MixColumn

Sau đây là một ví dụ về **MixColumns**:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Hình 3.18 Ví dụ về MixColumn

Chúng ta cùng xác minh cột đầu tiên của ví dụ này. Nhớ lại từ Mục 1 rằng, trong $GF(2^8)$, phép cộng là phép toán XOR từng bit và phép nhân đó có thể được thực hiện theo quy tắc. Cụ thể, phép nhân một giá trị x (nghĩa là với 02) có thể được thực hiện dưới dạng dịch trái 1 bit theo sau bằng XOR bitwise có điều kiện

với (00011011) nếu bit ngoài cùng bên trái của giá trị gốc (trước khi thay đổi) là 1. Do đó, để xác minh sự chuyển đổi **MixColumns** trên cột đầu tiên, chúng ta cần chỉ ra rằng:

$$
\begin{array}{llllll}
(\{02\}\cdot\{87\}) & \oplus & (\{03\}\cdot\{6E\}) & \oplus \{46\} & \oplus \{A6\} & = \{47\} \\
\{87\} & \oplus & (\{02\}\cdot\{6E\}) & \oplus (\{03\}\cdot\{46\}) & \oplus \{A6\} & = \{37\} \\
\{87\} & \oplus & \{6E\} & \oplus (\{02\}\cdot\{46\}) & \oplus (\{03\}\cdot\{A6\}) & = \{94\} \\
(\{03\}\cdot\{87\}) & \oplus & \{6E\} & \oplus \{46\} & \oplus (\{02\}\cdot\{A6\}) & = \{ED\}
\end{array}
$$

Hình 3.19 Chứng minh ví dụ MixColumn

Với phương trình đầu tiên, chúng ta có $\{2A\} \bullet \{87\} = (0000\ 1110)\ (0001\ 1011) = (0001\ 0101)$ và $\{03\} \bullet \{6E\} = \{6E\}\ (\{02\} \bullet \{6E\}) = (0110\ 1110)\ (1101\ 1100) = (1011\ 0010)$. Và:

$$
\begin{aligned}
\{02\}\cdot\{87\} &= 0001\ 0101 \\
\{03\}\cdot\{6E\} &= 1011\ 0010 \\
\{46\} &= 0100\ 0110 \\
\{A6\} &= \underline{1010\ 0110} \\
&\quad 0100\ 0111 = \{47\}
\end{aligned}
$$

Các phương trình khác có thể thực hiện tương tự.

Biến đổi trộn cột nghịch, được gọi là **InvMixColumns**, được xác định bởi phép nhân ma trận sau:

$$
\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}
\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}
=
\begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}
$$

Tương đương với :

$$
\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}
\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}
\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}
=
\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{0,3} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}
$$

Nghĩa là, ma trận biến đổi nghịch nhân với ma trận biến đổi thuận bằng ma trận đơn vị. Để xác minh cột đầu tiên của phương trình, chúng ta cần để xét:

$$
\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}
\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}
=
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Với phương trình đầu tiên, chúng ta có $\{0E\}$ •$\{02\}$ = 00011100 và $\{09\}$ •$\{03\}$ = $\{09\}$ ($\{09\}$ •$\{02\}$) = $00001001 \oplus 00010010 = 00011011$. Khi đó:

$$(\{0E\} \bullet \{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\} \bullet \{03\}) = \{01\}$$

$$(\{09\} \bullet \{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\} \bullet \{03\}) = \{00\}$$

$$(\{0D\} \bullet \{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\} \bullet \{03\}) = \{00\}$$

$$(\{0B\} \bullet \{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\} \bullet \{03\}) = \{00\}$$

Các phương trình khác có thể được thực hiện tương tự.

Tài liệu AES mô tả một cách khác để mô tả biến đổi **MixColumns**, đó là về mặt số học đa thức. Trong tiêu chuẩn,**MixColumns** được xác định bằng cách coi mỗi cột Trạng thái là một đa thức bốn hạng với các hệ số trong $GF(2^8)$. Mỗi cột được nhân modulo $(x^4 + 1)$ với đa thức cố định a(x), được cho bởi:

$a(x) = \{03\}\mathrm{x}^3 + \{01\}\mathrm{x}^2 + \{01\}\mathrm{x} + \{02\}$

Phép nhân mỗi cột **Trạng thái** với a(x) có thể được viết dưới dạng phép nhân ma trận của phương trình . Tương tự, có thể thấy rằng phép biến đổi trong phương trình trên tương ứng với việc coi mỗi cột là một đa thức 4 hạng và nhân mỗi cột với b(x), được cho bởi

$b(x) = \{0B\}\mathrm{x}^3 + \{0D\}\mathrm{x}^2 + \{09\}\mathrm{x} + \{0E\}$

Có thể dễ dàng chứng minh rằng $b(x) = a^{-1}(x) mod (x^4 + 1)$

Các hệ số của ma trận trong phương trình (6.3) dựa trên tuyến tính mã với khoảng cách tối đa giữa các từ mã, đảm bảo trộn tốt giữa các byte của mỗi cột. Việc chuyển đổi cột hỗn hợp kết hợp với phép biến đổi hàng dịch chuyển đảm bảo rằng sau một vài vòng, tất cả các bit đầu ra đều phụ thuộc vào tất cả các bit đầu vào.

Ngoài ra, việc lựa chọn các hệ số trong MixColumns, tất cả đều là $\{01\}$, $\{02\}$, hoặc $\{03\}$, bị ảnh hưởng bởi một số ràng buộc. Như đã thảo luận, phép nhân với các hệ số này bao gồm tối đa một phép dịch chuyển và phép XOR. các hệ số trong **InvMixColumns** khó triển khai hơn. Tuy nhiên, mã hóa đã được coi là quan trọng hơn giải mã.

3.4 AddRoundKey Transformation

Trong phép biến đổi cộng khóa vòng thuận, được gọi là **AddRoundKey**, 128 bit của Trạng thái được XOR theo từng bit với 128 bit của khóa vòng. Như thể hiện trong Hình 3.20, hoạt động này được xem như là một thao tác với cột giữa 4 byte của cột Trạng thái và một từ của khóa vòng ; nó cũng có thể được xem như một hoạt động cấp byte. Sau đây là một ví dụ về **AddRoundKey**:

Ma trận đầu tiên là Trạng thái và ma trận thứ hai là khóa vòng.

Phép biến đổi cộng khóa vòng nghịch giống với phép cộng khóa vòng thuận, bởi vì phép toán XOR là nghịch đảo của chính nó.

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

$\oplus$

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

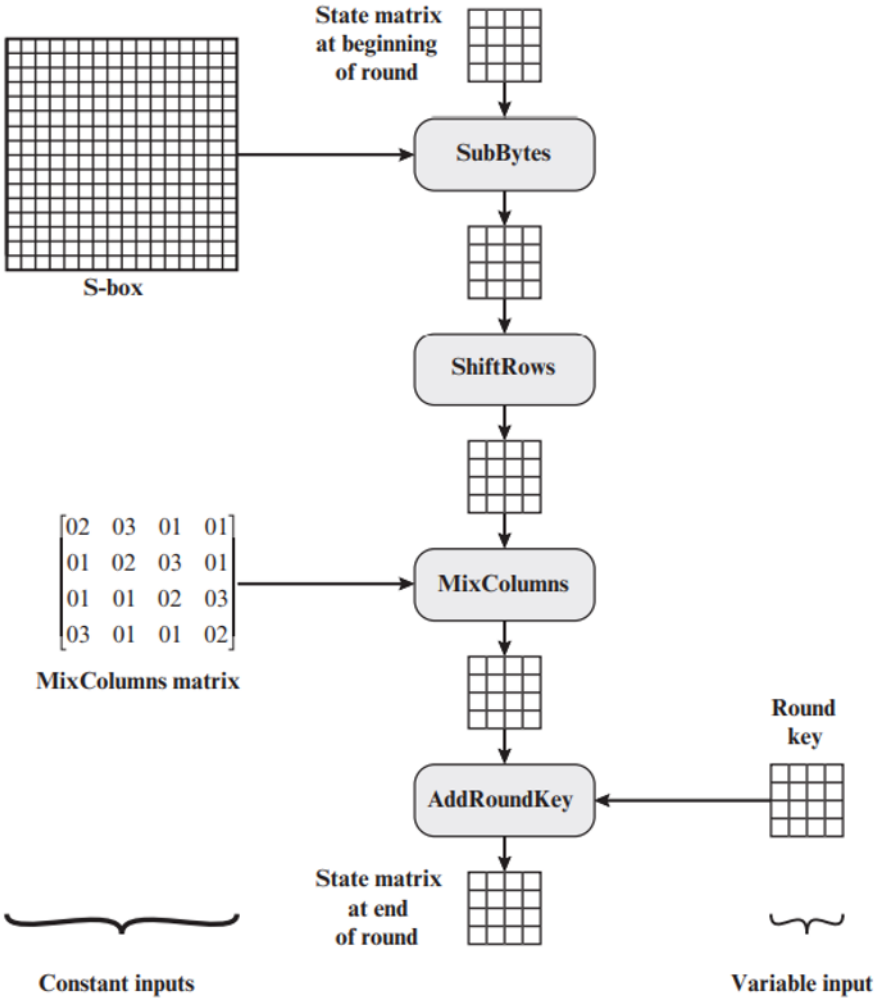
=

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

Hình 3.20 Chứng minh ví dụ AddRoundKey

Biến đổi cộng khóa vòng đơn có thể và dễ dàng ảnh hưởng đến từng phần của Trạng thái. Độ phức tạp của việc mở rộng phím tròn, cộng với độ phức tạp của các giai đoạn khác của AES, đảm bảo về bảo mật.

Hình 3.21 là một góc nhìn nữa một chu trình đơn của AES, nhấn mạnh các cơ chế và đầu vào của mỗi chuyển đổi.



Hình 3.21 Các đầu vào của chu trình AES đơn

4 MỞ RỘNG KHÓA AES

4.1 Thuật toán mở rộng khóa AES

Thuật toán mở rộng khóa AES nhận một khóa bốn từ (16 byte) làm đầu vào và tạo ra một mảng tuyến tính của 44 từ (176 byte). Điều này đủ để cung cấp một khóa vòng bốn từ cho giai đoạn **AddRoundKey** ban đầu và mỗi trong 10 vòng của mã hóa. **Pseudocode** trên trang tiếp theo mô tả việc mở rộng.

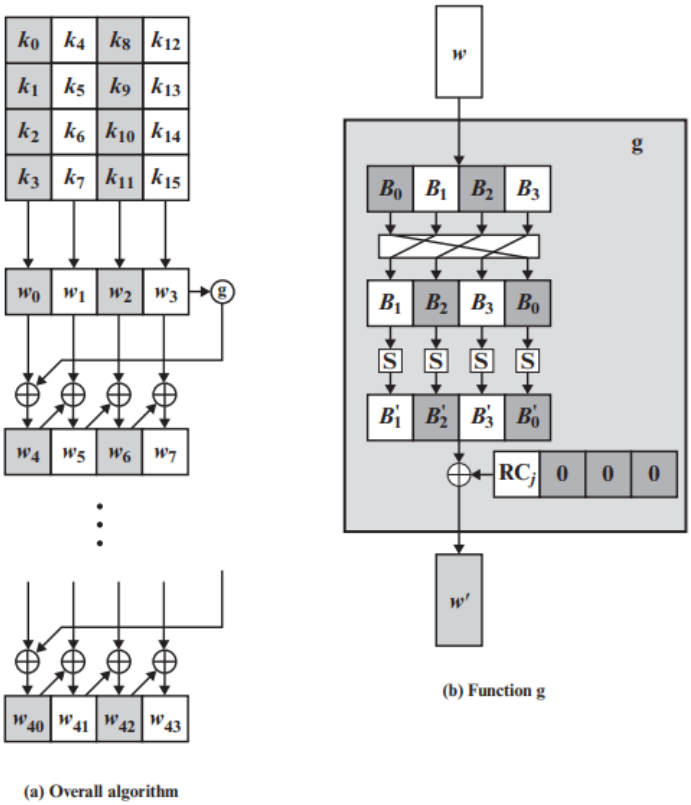
Khóa được sao chép vào bốn từ đầu tiên của khóa mở rộng. Phần còn lại của khóa mở rộng được điền vào bốn từ một lần. Mỗi từ được thêm vào w[i] phụ thuộc vào từ trước đó, w[i - 1], và từ bốn vị trí trở lại, w[i - 4]. Trong ba trong bốn trường hợp, sử dụng một XOR đơn giản. Cho một từ có vị trí trong mảng w là bội số của 4, sử dụng một hàm phức tạp hơn. Hình 6.9 minh họa việc tạo ra khóa mở rộng, sử dụng ký hiệu g để biểu thị hàm phức tạp đó. Hàm g bao gồm các hàm con sau.

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for (i = 0; i < 4; i++) w[i] = (key[4*i], key[4*i+1],
                                    key[4*i+2],
                                    key[4*i+3]);
    for (i = 4; i < 44; i++)
    {
    temp = w[i - 1];
    if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                            ⊕ Rcon[i/4];
    w[i] = w[i-4] ⊕ temp
    }
}
```

1. RotWord thực hiện một dịch trái vòng tròn một byte trên một từ. Điều này có nghĩa là một từ đầu vào [B0, B1, B2, B3] được biến đổi thành [B1, B2, B3, B0].
2. SubWord thực hiện một thay thế byte trên mỗi byte của từ đầu vào của nó, sử dụng S-box.
3. Kết quả của các bước 1 và 2 được XOR với một hằng số vòng, Rcon [j].

Hằng số vòng là một từ trong đó ba byte bên phải luôn là 0.

Do đó, tác động của XOR của một từ với Rcon chỉ là thực hiện XOR trên byte bên trái của từ. Hằng số vòng khác nhau cho mỗi vòng và được định nghĩa như Rcon [j] = (RC [j], 0, 0, 0), với RC [1] = 1, RC [j] = 2 * RC [j-1] và với nhân được định nghĩa trên lĩnh vực $GF(2^8)$. Giá trị của RC[j] theo hệ thập lục phân là



Hình 4.22 Thuật toán sinh khóa AES

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

Sau đó, 4 byte đầu tiên (cột đầu tiên) của khóa tròn cho vòng 9 được tính như sau:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i − 4]	w[i] = temp ⊕ w[i − 4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Cơ sở lý luận

Nhà phát triển Rijndael đã thiết kế thuật toán mở rộng khóa để chống lại các tấn công mã hóa đã biết. Bổ sung của một hằng số vòng phụ thuộc vòng loại bỏ đối xứng hoặc tương tự giữa cách tạo khóa vòng trong các vòng khác nhau. Các tiêu chí cụ thể được sử dụng là:

- Không có thể tính toán nhiều bit khóa vòng khác khi biết một phần của khóa hoặc khóa vòng.
- Một biến đổi đảo ngược [tức, biết bất kỳ Nk từ liên tiếp của khóa mở rộng có thể tái tạo toàn bộ khóa mở rộng (Nk = kích thước khóa theo từ)].
- Tốc độ trên một số bộ xử lý rộng rãi.

- Sử dụng hằng số vòng để loại bỏ đối xứng.
- Chuyển hoá sự khác biệt của khóa mã thành sự khác biệt của khóa vòng; tức là, mỗi bit khóa ảnh hưởng đến nhiều bit khóa vòng.
- Đủ phi tuyến để cấm việc xác định đầy đủ sự khác biệt của khóa vòng từ sự khác biệt của khóa mã.
- Mô tả đơn giản.

5 MỘT VÍ DỤ VỀ AES

5.1 Ví dụ minh họa

Chúng ta hiện đang làm việc với một ví dụ và xem một số tác động của nó. Mặc dù bạn không cần làm lại ví dụ bằng tay, bạn sẽ thấy rất hữu ích khi nghiên cứu các mẫu hex xảy ra từ một bước đến bước kế tiếp.

Với ví dụ này, văn bản mẫu là một chuỗi hexadecimal đối xứng. Văn bản mẫu, khóa và văn bản mã hoá kết quả là:

Plaintext:	`0123456789abcdeffedcba9876543210`
Key:	`0f1571c947d9e8590cb7add6af7f6798`
Ciphertext:	`ff0b844a0853bf7c6934ab4364148fb9`

Bảng 6.3 cho thấy mở rộng của khóa 16 byte thành 10 khóa vòng. Như đã giải thích trước đó, quá trình này được thực hiện từ từ, với mỗi từ 4 byte đảm nhận một cột của ma trận khóa vòng từ. Cột bên trái cho thấy bốn từ khóa vòng được tạo ra cho mỗi vòng. Cột bên phải cho thấy các bước sử dụng để tạo từ hỗ trợ trong mở rộng khóa. Chúng ta bắt đầu, tất nhiên, với khóa chính làm khóa vòng cho vòng 0.

Bảng 6.4 cho thấy tiến trình của Trạng thái qua quá trình mã hoá AES. Cột đầu tiên cho thấy giá trị của **Trạng thái** ở đầu một vòng. Cho dòng đầu tiên, Trạng thái chỉ là sắp xếp ma trận của văn bản mẫu. Cột thứ hai, ba và bốn cho thấy giá trị của Trạng thái cho vòng đó sau khi thực hiện biến đổi **SubBytes**, **ShiftRows** và **MixColumns**. Cột thứ năm hiển thị khóa vòng. Bạn có thể xác minh rằng các khóa vòng này tương đương với những gì được hiển thị trong Bảng 6.3. Cột đầu tiên hiển thị giá trị của State kết quả từ phép XOR bitwise giữa State sau khi **MixColumns** trước đó với khóa vòng của vòng trước đó.

5.2 Hiệu ứng thác

Nếu một sự thay đổi nhỏ trong khóa hoặc plaintext có thể tạo ra một sự thay đổi tương đối nhỏ trong ciphertext, điều này có thể được sử dụng để có hiệu quả giảm kích thước của không gian plaintext (hoặc khóa) để tìm kiếm. Những gì được mong muốn là hiệu ứng thác, trong đó một sự thay đổi nhỏ trong plaintext hoặc khóa sẽ tạo ra một sự thay đổi lớn trong ciphertext.

Sử dụng ví dụ từ bảng 6.4, bảng 6.5 cho thấy kết quả khi số hữu tỉ thứ tám của plaintext bị thay đổi. Cột thứ hai của bảng cho thấy giá trị của ma trận State tại cuối mỗi vòng cho hai plaintexts. Lưu ý rằng sau chỉ một vòng, 20 bit của vectơ State khác nhau.

Sau hai vòng, gần nửa bit khác nhau. Sự khác biệt này đẩy nổi qua các vòng còn lại. Một sự khác biệt bit tại khoảng gần nửa vị trí là kết quả mong muốn nhất. Rõ ràng, nếu gần tất cả các bit đều thay đổi, điều này sẽ tương đương với gần không có bit nào thay đổi. Để nói một cách khác, nếu chúng ta chọn hai plaintext ngẫu nhiên, chúng ta có thể mong đợi hai plaintext khác nhau tại khoảng gần nửa vị trí bit và hai ciphertext cũng khác nhau tại khoảng gần nửa vị trí.

Bảng 5.23 cho thấy sự thay đổi của giá trị ma trận State khi sử dụng cùng một plaintext và hai khoá khác nhau trong bit thứ tám. Để rõ hơn, trong trường hợp thứ hai, khoá là 0e1571c947d9e8590cb7add6af7f6798. Một vòng sẽ tạo ra một sự thay đổi đáng kể, và mức độ thay đổi sau tất cả các vòng tiếp theo khoảng gần nửa bit. Vì vậy, dựa trên ví dụ này, AES cho thấy hiệu ứng thác rất mạnh.

Key Words	Auxiliary Function
w0 = 0f 15 71 c9 w1 = 47 d9 e8 59 w2 = 0c b7 ad d6 w3 = af 7f 67 98	RotWord (w3) = 7f 67 98 af = x1 SubWord (x1) = d2 85 46 79 = y1 Rcon (1) = 01 00 00 00 y1 ⊕ Rcon (1) = d3 85 46 79 = z1
w4 = w0 ⊕ z1 = dc 90 37 b0 w5 = w4 ⊕ w1 = 9b 49 df e9 w6 = w5 ⊕ w2 = 97 fe 72 3f w7 = w6 ⊕ w3 = 38 81 15 a7	RotWord (w7) = 81 15 a7 38 = x2 SubWord (x2) = 0c 59 5c 07 = y2 Rcon (2) = 02 00 00 00 y2 ⊕ Rcon (2) = 0e 59 5c 07 = z2
w8 = w4 ⊕ z2 = d2 c9 6b b7 w9 = w8 ⊕ w5 = 49 80 b4 5e w10 = w9 ⊕ w6 = de 7e c6 61 w11 = w10 ⊕ w7 = e6 ff d3 c6	RotWord (w11) = ff d3 c6 e6 = x3 SubWord (x3) = 16 66 b4 83 = y3 Rcon (3) = 04 00 00 00 y3 ⊕ Rcon (3) = 12 66 b4 8e = z3
w12 = w8 ⊕ z3 = c0 af df 39 w13 = w12 ⊕ w9 = 89 2f 6b 67 w14 = w13 ⊕ w10 = 57 51 ad 06 w15 = w14 ⊕ w11 = b1 ae 7e c0 w16 = w12 ⊕ z4 = 2c 5c 65 f1 w17 = w16 ⊕ w13 = a5 73 0e 96 w18 = w17 ⊕ w14 = f2 22 a3 90 w19 = w18 ⊕ w15 = 43 8c dd 50	RotWord (w15) = ae 7e c0 b1 = x4 SubWord (x4) = e4 f3 ba c8 = y4 Rcon (4) = 08 00 00 00 y4 ⊕ Rcon (4) = ec f3 ba c8 = 4 RotWord (w19) = 8c dd 50 43 = x5 SubWord (x5) = 64 c1 53 1a = y5 Rcon(5) = 10 00 00 00 y5 ⊕ Rcon (5) = 74 c1 53 1a = z5
w20 = w16 ⊕ z5 = 58 9d 36 eb w21 = w20 ⊕ w17 = fd ee 38 7d w22 = w21 ⊕ w18 = 0f cc 9b ed w23 = w22 ⊕ w19 = 4c 40 46 bd	RotWord (w23) = 40 46 bd 4c = x6 SubWord (x6) = 09 5a 7a 29 = y6 Rcon(6) = 20 00 00 00 y6 ⊕ Rcon(6) = 29 5a 7a 29 = z6
w24 = w20 ⊕ z6 = 71 c7 4c c2 w25 = w24 ⊕ w21 = 8c 29 74 bf w26 = w25 ⊕ w22 = 83 e5 ef 52 w27 = w26 ⊕ w23 = cf a5 a9 ef	RotWord (w27) = a5 a9 ef cf = x7 SubWord (x7) = 06 d3 bf 8a = y7 Rcon (7) = 40 00 00 00 y7 ⊕ Rcon(7) = 46 d3 df 8a = z7
w28 = w24 ⊕ z7 = 37 14 93 48 w29 = w28 ⊕ w25 = bb 3d e7 f7 w30 = w29 ⊕ w26 = 38 d8 08 a5 w31 = w30 ⊕ w27 = f7 7d a1 4a	RotWord (w31) = 7d a1 4a f7 = x8 SubWord (x8) = ff 32 d6 68 = y8 Rcon (8) = 80 00 00 00 y8 ⊕ Rcon(8) = 7f 32 d6 68 = z8
w32 = w28 ⊕ z8 = 48 26 45 20 w33 = w32 ⊕ w29 = f3 1b a2 d7 w34 = w33 ⊕ w30 = cb c3 aa 72 w35 = w34 ⊕ w32 = 3c be 0b 3	RotWord (w35) = be 0b 38 3c = x9 SubWord (x9) = ae 2b 07 eb = y9 Rcon (9) = 1B 00 00 00 y9 ⊕ Rcon (9) = b5 2b 07 eb = z9
w36 = w32 ⊕ z9 = fd 0d 42 cb w37 = w36 ⊕ w33 = 0e 16 e0 1c w38 = w37 ⊕ w34 = c5 d5 4a 6e w39 = w38 ⊕ w35 = f9 6b 41 56	RotWord (w39) = 6b 41 56 f9 = x10 SubWord (x10) = 7f 83 b1 99 = y10 Rcon (10) = 36 00 00 00 y10 ⊕ Rcon (10) = 49 83 b1 99 = z10
w40 = w36 ⊕ z10 = b4 8e f3 52 w41 = w40 ⊕ w37 = ba 98 13 4e w42 = w41 ⊕ w38 = 7f 4d 59 20 w43 = w42 ⊕ w39 = 86 26 18 76	

Hình 5.23 Ví dụ về AES

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

Hình 5.24 Ví dụ về AES (tiếp)

Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0023456789abcdeffedcba9876543210	1
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c4a9ad090fc7ff3fc0e8e8ca4dd02a9c	20
2	5c7bb49a6b72349b05a2317ff46d1294 fe2ae569f7ee8bb8c1f5a2bb37ef53d5	58
3	7115262448dc747e5cdac7227da9bd9c ec093dfb7c45343d689017507d485e62	59
4	f867aee8b437a5210c24c1974cffeabc 43efdb697244df808e8d9364ee0ae6f5	61
5	721eb200ba06206dcbd4bce704fa654e 7b28a5d5ed643287e006c099bb375302	68
6	0ad9d85689f9f77bc1c5f71185e5fb14 3bc2d8b6798d8ac4fe36a1d891ac181a	64
7	db18a8ffa16d30d5f88b08d777ba4eaa 9fb8b5452023c70280e5c4bb9e555a4b	67
8	f91b4fbfe934c9bf8f2f85812b084989 20264e1126b219aef7feb3f9b2d6de40	65
9	cca104a13e678500ff59025f3bafaa34 b56a0341b2290ba7dfdfbddcd8578205	61
10	ff0b844a0853bf7c6934ab4364148fb9 612b89398d0600cde116227ce72433f0	58

Hình 5.25 Hiệu ứng thác đổ ở AES: Thay đổi ở Plaintext

Round		Number of Bits that Differ
	0123456789abcdeffedcba9876543210 0123456789abcdeffedcba9876543210	0
0	0e3634aece7225b6f26b174ed92b5588 0f3634aece7225b6f26b174ed92b5588	1
1	657470750fc7ff3fc0e8e8ca4dd02a9c c5a9ad090ec7ff3fc1e8e8ca4cd02a9c	22
2	5c7bb49a6b72349b05a2317ff46d1294 90905fa9563356d15f3760f3b8259985	58
3	7115262448dc747e5cdac7227da9bd9c 18aeb7aa794b3b66629448d575c7cebf	67
4	f867aee8b437a5210c24c1974cffeabc f81015f993c978a876ae017cb49e7eec	63
5	721eb200ba06206dcbd4bce704fa654e 5955c91b4e769f3cb4a94768e98d5267	81
6	0ad9d85689f9f77bc1c5f71185e5fb14 dc60a24d137662181e45b8d3726b2920	70
7	db18a8ffa16d30d5f88b08d777ba4eaa fe8343b8f88bef66cab7e977d005a03c	74
8	f91b4fbfe934c9bf8f2f85812b084989 da7dad581d1725c5b72fa0f9d9d1366a	67
9	cca104a13e678500ff59025f3bafaa34 0ccb4c66bbfd912f4b511d72996345e0	59
10	ff0b844a0853bf7c6934ab4364148fb9 fc8923ee501a7d207ab670686839996b	53

Hình 5.26 Hiệu ứng thác đổ ở AES: thay đổi ở Key

6 AES IMPLEMENTATION

6.1 Thuật toán mã hóa ngược

Như đã đề cập, giải mã mật mã AES không giống với mã hoá. Điều này có nghĩa là chuỗi các biến đổi cho giải mã khác với biến đổi cho mã hoá, mặc dù cấu trúc lịch trình khóa cho mã hoá và giải mã giống nhau. Nó có nhược điểm là cần có hai mô-đun phần mềm hoặc firmware riêng biệt cho các ứng dụng yêu cầu mã hoá và giải mã. Tuy nhiên, có một phiên bản tương đương của thuật toán giải mã có cùng cấu trúc như thuật toán mã hoá. Phiên bản tương đương này có cùng chuỗi các biến đổi như thuật toán mã hoá (với các biến đổi được thay bằng các biến đổi ngược). Để đạt được tính tương đương này, cần một sự thay đổi trong lịch trình khóa. Tuy nhiên, có một phiên bản tương đương của thuật toán mã hóa có cấu trúc tương tự như thuật toán giải mã.

Phiên bản tương đương này có cùng chuỗi biến đổi như thuật toán mã hóa (với các biến đổi được thay thế bằng các biến đổi đối xứng). Để đạt được tương đương này, cần sự thay đổi lịch trình trong khóa. Để đồng bộ hoá cấu trúc giải mã với cấu trúc mã hóa, cần hai thay đổi riêng biệt. Như được minh hoạ trong các hình trên, một vòng mã hóa có cấu trúc **SubBytes**, **ShiftRows**, **MixColumns**, **AddRoundKey**. Vòng giải mã tiêu chuẩn có cấu trúc **InvShiftRows**, **InvSubBytes**, **AddRoundKey**, **InvMixColumns**. Do đó, hai giai đoạn đầu tiên của vòng giải mã cần được hoán vị, và hai giai đoạn cuối của vòng giải mã cũng cần được hoán vị.

6.1.1 Hoán đổi InvShiftRows và InvSubBytes

InvShiftRows ảnh hưởng đến thứ tự các byte trong State nhưng không thay đổi nội dung của byte và không phụ thuộc vào nội dung của byte để thực hiện biến đổi của nó. **InvSubBytes** ảnh hưởng đến nội dung của byte trong **State** nhưng không thay đổi thứ tự byte và không phụ thuộc vào thứ tự byte để thực hiện biến đổi của nó. Do đó, hai hoạt động này hoán vị và có thể hoán vị. Cho một State S_i cho trước

$$\textbf{InvShiftRows}\,[\textbf{InvSubBytes}(S_i)] = \textbf{InvSubBytes}\,[\textbf{InvShiftRows}(S_i)]$$

6.1.2 Hoán đổi AddRoundKey và InvMixColumns

Các biến đổi **AddRoundKey** và **InvMixColumns** không thay đổi thứ tự các byte trong State. Nếu chúng ta xem khóa là một chuỗi từ, thì cả **AddRoundKey** và **InvMixColumns** hoạt động trên State một cột một lần. Hai hoạt động này là tuyến tính với đầu vào cột. Đó là, cho một State S_i cho trước và một khóa vòng w_j cho trước.

$$\textbf{InvMixColumns}(S_i \oplus w_j) = [\textbf{InvMixColumns}(S_i)] \oplus [\textbf{InvMixColumns}(w_j)]$$

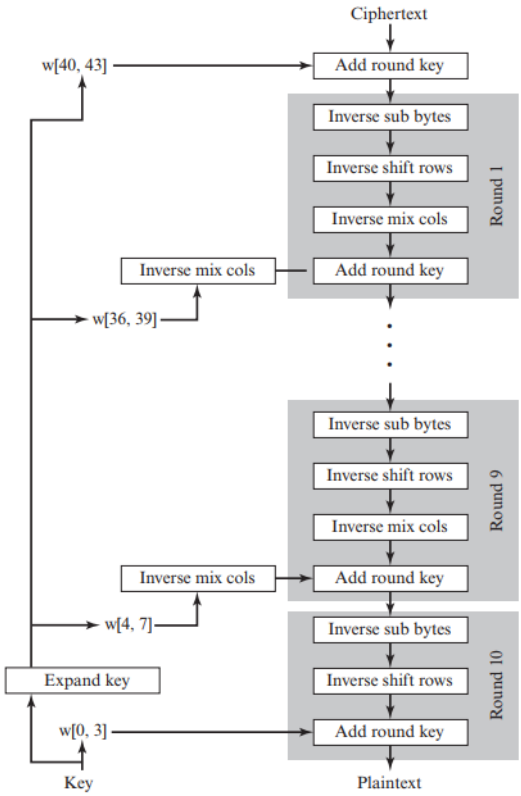
Để thấy điều này, giả sử rằng cột đầu tiên của State Si là chuỗi (y_0, y_1, y_2, y_3)và cột đầu tiên của khóa vòng $w-j$ là (k_0, k_1, k_2, k_3). Sau đó ta cần chứng minh

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \oplus k_0 \\ y_1 \oplus k_1 \\ y_2 \oplus k_2 \\ y_3 \oplus k_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \oplus \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

$$\begin{aligned} &[\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)] \\ &\quad = [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3] \oplus \\ &\quad\quad [\{0E\} \cdot k_0] \oplus [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3] \end{aligned}$$

Phương trình này hợp lệ theo quan sát. Vì vậy, chúng ta có thể hoán đổi AddRoundKey và InvMixColumns, đều đặn khi chúng ta trước tiên áp dụng InvMixColumns cho khóa vòng. Lưu ý rằng chúng ta không cần áp dụng InvMixColumns cho khóa vòng cho bản đầu vào cho biến đổi AddRoundKey đầu tiên (trước vòng đầu tiên) hoặc cho biến đổi AddRoundKey cuối cùng (trong vòng 10). Điều này là bởi vì hai biến đổi AddRoundKey này không hoán đổi với InvMixColumns để tạo ra giải mã thuật toán tương đương.

Hình 6.27 minh họa giải mã thuật toán tương đương.



Hình 6.27 Mật mã AES nghịch đảo tương đương

6.2 Các khía cạnh thực hiện

Rijndael đề xuất cung cấp một số gợi ý để thực hiện hiệu quả trên bộ xử lý 8 bit, thường được sử dụng trên thẻ tín dụng hiện đại, và trên bộ xử lý 32 bit, thường được sử dụng trên PC. BỘ XỬ LÝ 8 BIT AES có thể được thực hiện rất hiệu quả trên một bộ xử lý 8 bit. **AddRoundKey** là một hoạt động XOR từng byte. **ShiftRows** là một hoạt động dịch chuyển byte đơn giản. **SubBytes** hoạt động tại mức byte và chỉ cần một bảng 256 byte.

Chuyển đổi **MixColumns** yêu cầu nhân ma trận trong trường $GF(2^8)$, có nghĩa là tất cả các thao tác được thực hiện trên các byte. MixColumns chỉ yêu cầu nhân bởi $\{02\}$ và $\{03\}$, điều này, như chúng ta đã thấy, có liên quan đến các phép dịch chuyển đơn giản, XOR có điều kiện và XOR. Điều này có thể được thực hiện theo cách hiệu quả hơn bởi loại bỏ các dịch chuyển và XOR điều kiện. Bộ phương trình (6.4) cho thấy phương trình cho chuyển đổi MixColumns trên một cột duy nhất. Sử dụng công thức $\{03\}x=(\{02\}x) \oplus x$ ta được phương trình (6.4) như sau:

$$
\begin{aligned}
Tmp &= s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\
s'_{0,j} &= s_{0,j} \oplus \mathrm{Tmp} \oplus X2[s_{0,j} \oplus s_{1,j}] \\
s'_{1,c} &= s_{1,j} \oplus \mathrm{Tmp} \oplus X2[s_{1,j} \oplus s_{2,j}] \\
s'_{2,c} &= s_{2,j} \oplus \mathrm{Tmp} \oplus X2[s_{2,j} \oplus s_{3,j}] \\
s'_{3,j} &= s_{3,j} \oplus \mathrm{Tmp} \oplus X2[s_{3,j} \oplus s_{0,j}]
\end{aligned}
$$

Hình 6.28 Phân tích MixColumn

BỘ XỬ LÝ 32 BIT Phiên bản được mô tả trong phần trước sử dụng cho các phép tính 8 bit. Với bộ xử lý 32 bit, một phiên bản hiệu quả hơn có thể đạt được nếu các thao tác được xác định trên từ 32 bit. Để minh họa điều này, chúng ta trước tiên xác định bốn bước chuyển đổi bằng dạng toán học. Giả sử chúng ta bắt đầu với một ma trận Trạng thái bao gồm các phần tử ai, j và một ma trận khóa vòng bao gồm các phần tử ki, j. Sau đó, các chuyển đổi có thể được biểu thị như sau:

SubBytes	$b_{i,j} = \mathrm{S}[a_{i,j}]$
ShiftRows	$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$
MixColumns	$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$
AddroundKey	$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$

Hình 6.29 Chuyển đổi các trạng thái

Trong phương trình ShiftRows, chỉ số cột được lấy dưới dạng mod 4. Chúng ta có thể kết hợp tất cả các biểu thức này thành một phương trình duy nhất:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j-2}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Hình 6.30 Chuyển đổi các trạng thái

Trong phương trình thứ hai, chúng ta đang biểu diễn nhân ma trận như một tổ hợp tuyến tính của các vectơ. Chúng ta định nghĩa bốn bảng 256 từ (1024 byte) như sau:

$T_0[x] = \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[x] \right)$	$T_1[x] = \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \right)$	$T_2[x] = \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \right)$	$T_3[x] = \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x] \right)$

Do đó, mỗi bảng nhận một giá trị byte và tạo ra một vectơ cột (một từ 32 bit) là một hàm của mục S-box cho giá trị byte đó. Những bảng này có thể tính trước. Chúng ta có thể định nghĩa một hàm vòng hoạt động trên một cột theo cách sau:

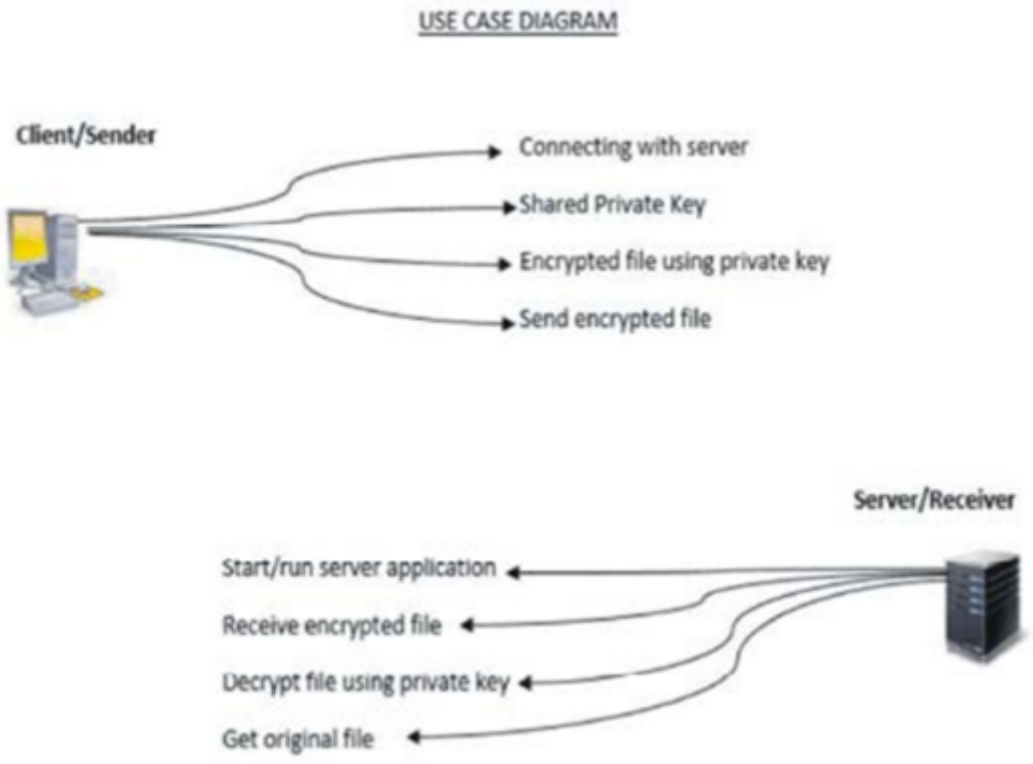
$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

Việc triển khai dựa trên phương trình trước đó chỉ cần bốn tra cứu bảng và bốn XOR cho mỗi cột mỗi vòng, cùng với 4 Kbyte để lưu trữ bảng. Những nhà phát triển của Rijndael tin rằng việc triển khai nhỏ gọn, hiệu quả này có thể là một trong những yếu tố quan trọng nhất trong việc chọn Rijndael cho AES.

7 ỨNG DỤNG XÂY DỰNG PHẦN MỀM TRUYỀN DỮ LIỆU BẰNG KHÓA AES

7.1 Đặt vấn đề

Một client/người gửi có một tệp có tính bảo mật cao cần được chuyển đến máy chủ/người nhận trên cùng một mạng. Tệp nằm trong máy tính của khách hàng. Tệp phải được chuyển mà không bị thay đổi hoặc bị bắt bởi các trình thám thính gói tin. Đầu tiên, một máy chủ cần chạy hệ thống để cho phép máy khách gửi tệp. Sau đó, máy khách chạy hệ thống duyệt tìm file trong máy tính và chọn file muốn gửi đến máy chủ/máy nhận. Tiếp theo, sau khi hoàn tất quá trình duyệt và chọn tệp, quá trình mã hóa bắt đầu khi nhấp vào nút gửi. Giờ đây, tệp đã được mã hóa và truyền qua mạng LAN một cách an toàn. Khi máy chủ nhận được tệp, quá trình giải mã bắt đầu chuyển đổi tệp được mã hóa thành tệp gốc của nó, sau đó máy chủ có thể đọc được tệp này. Tệp được lưu trữ trong đĩa cục bộ trong máy tính của máy chủ.



Hình 7.31 Sơ đồ use case

7.2 Các bước triển khai

- Tạo hai chương trình client và server side bằng ứng dụng Visual Code Studio.
- Sử dụng Lập trình Java Socket để làm giao thức truyền thông TCP.

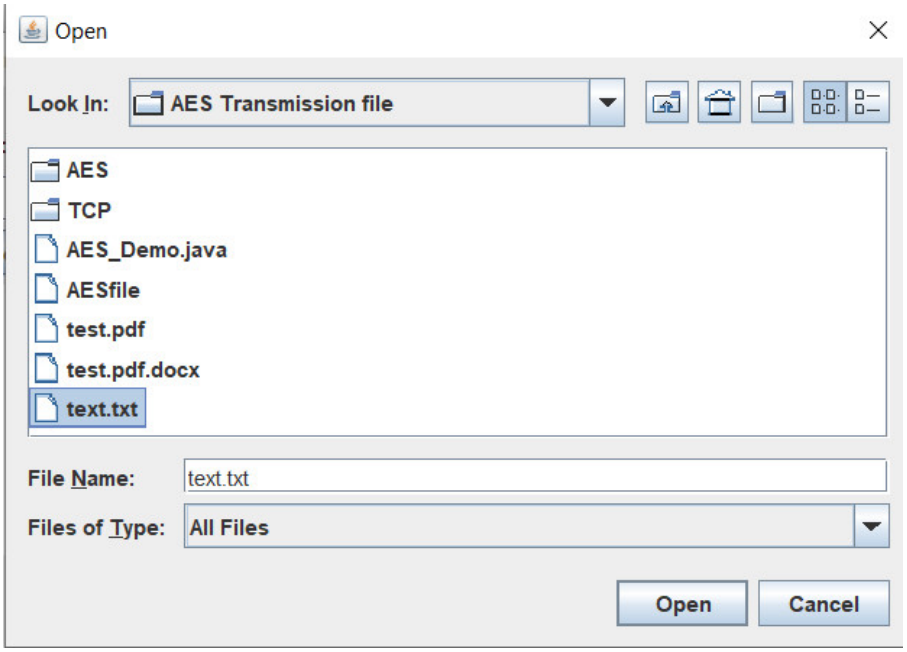
- Tạo một đối tượng Socket ở phía máy chủ và máy khách để kích hoạt giao tiếp.
- Sử dụng một phương thức của lớp java.net.Socket đại diện cho một ổ cắm ở phía Máy khách. Sử dụng một phương thức của lớp java.net.ServerSocket ở phía Máy chủ để chương trình máy chủ lắng nghe các máy khách và thiết lập kết nối với chúng.
- Máy chủ khởi tạo một đối tượng ServerSocket, biểu thị giao tiếp số cổng nào sẽ xảy ra.
- Máy chủ gọi phương thức accept() của lớp ServerSocket. Phương pháp này đợi cho đến khi máy khách kết nối với máy chủ trên cổng đã cho.
- Sau khi máy chủ đang đợi, máy khách khởi tạo một đối tượng Socket, chỉ định tên máy chủ và số cổng để kết nối.
- Hàm tạo của lớp Socket cố gắng kết nối máy khách đến máy chủ và port được chỉ định. Nếu giao tiếp được thiết lập thì hiện máy khách có một đối tượng Socket có khả năng giao tiếp với máy chủ.
- Mỗi ổ cắm có cả OutputStream và InputStream. OutputStream của máy khách được kết nối với InputStream của máy chủ và InputStream của máy khách được kết nối với OutputStream của máy chủ.
- Nhưng đối với ứng dụng này, phía Máy chủ chỉ có thể truyền hoặc gửi dữ liệu cho máy khách và phía Máy khách chỉ có thể nhận dữ liệu từ máy khách.
- InputStream được sử dụng để đọc dữ liệu từ nguồn và OutputStream được sử dụng để ghi dữ liệu đến đích.
- Sau đó, tạo đối tượng tệp sẽ được chuyển bằng phương pháp FileInputStream và FileOutputStream.
- Đối tượng tệp sẽ được chuyển đổi thành luồng byte bằng cách sử dụng cơ chế tuần tự hóa.
- Tuần tự hóa trong java là một cơ chế ghi trạng thái của một đối tượng vào một luồng byte.
- Thuật toán mã hóa AES đã được triển khai và bắt đầu hoạt động sau khi các đối tượng tệp được chuyển đổi thành luồng byte. Thuật toán AES sẽ mã hóa luồng byte và chuyển đổi nó thành bản mã trước khi được chuyển đến phía bên nhận bằng cách sử dụng lớp Mật mã cung cấp chức năng của mật mã được sử dụng để mã hóa và giải mã.
- Sau khi máy tính của người nhận nhận được luồng byte được mã hóa hoặc bản mã, thuật toán giải mã AES sẽ bắt đầu quá trình chuyển đổi bản mã thành đối tượng tệp bằng cách sử dụng phương thức lớp Cipher.

7.3 Xây dựng giao diện

Graphic User Interface (GUI): Giao diện này đã được phát triển bởi lập trình java trong ứng dụng Visual Code Studio. Người dùng (Client) cần biết địa chỉ IP của máy tính của người nhận (Server) và số cổng (Port) đã được sử dụng trong ứng dụng. Sau đó, trường địa chỉ IP cần được điền bằng địa chỉ IP của người gửi và tên cổng cần được kết nối.

Hình 7.32 Giao diện chính của chương trình

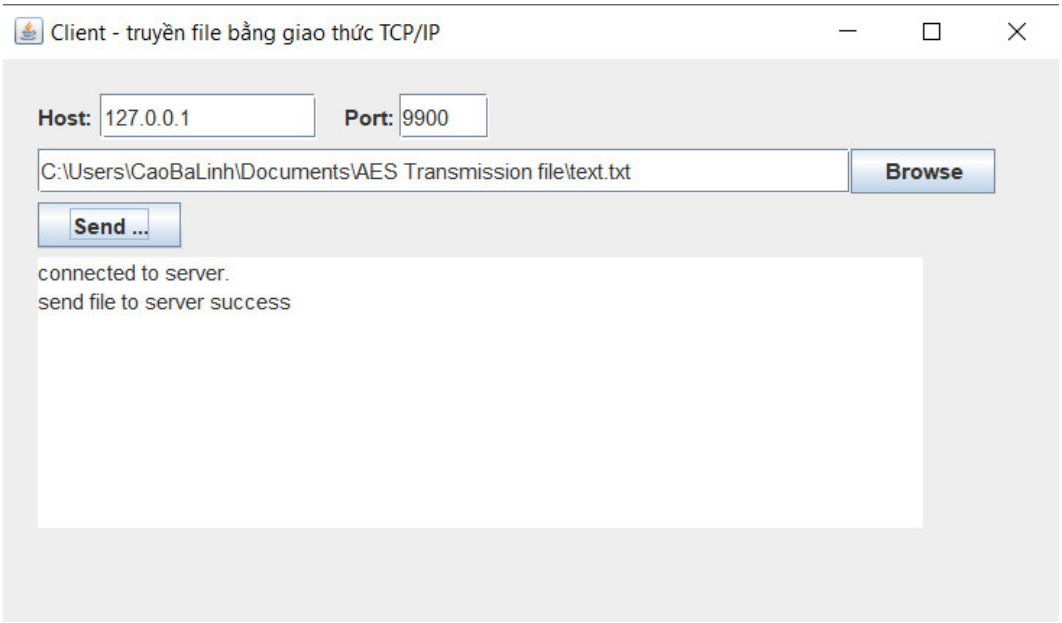
Tiếp theo, người dùng sẽ chọn thư mục cần gửi đi bằng thao tác nhấn nút Browse.



Hình 7.33 Chọn file gửi đi

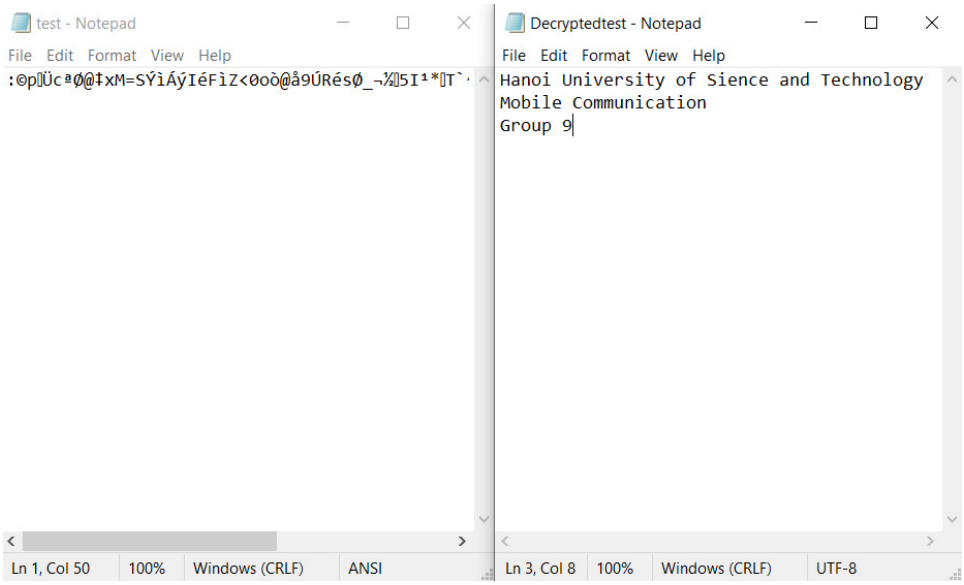
File trước khi gửi đi sẽ được mã hóa bằng hệ mật AES và khóa cho trước. File đến bên Server sẽ được giải mã và khôi phục như file gốc ban đầu.

Khi gửi file thành công, thanh trạng thái sẽ thông báo như sau:

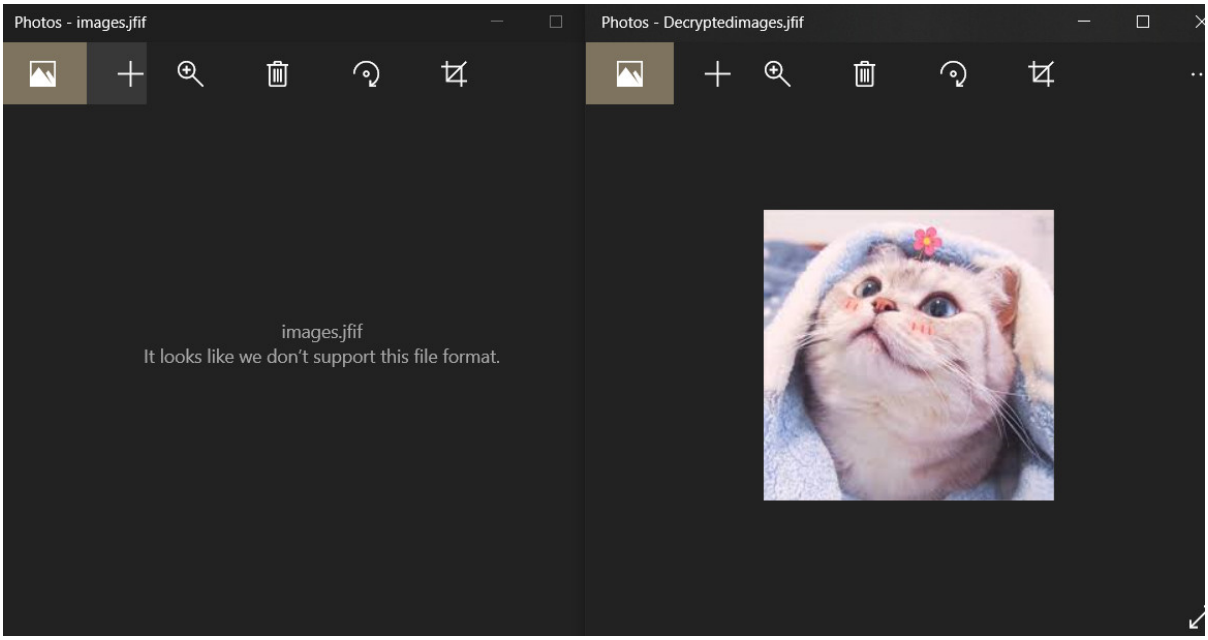


Hình 7.34 Gửi file thành công

Các trường hợp thử nghiệm: Nhóm chúng em thử với 3 trường hợp lần lượt các file gửi dưới định dạng txt, pdf, dạng ảnh png và thu được kết quả như sau:



Hình 7.35 File txt



Hình 7.36 File ảnh

8 Kết luận

Qua quá trình làm việc nhóm, chúng em đã tìm hiểu toàn bộ lý thuyết, cách mã hóa dữ liệu bằng hệ mật mã AES. Từ đó, nhóm chúng em đã xây dựng được một phần mềm truyền file dữ liệu có bảo mật với giao thức TCP/IP bằng lập trình Socket trên nền tảng ngôn ngữ lập trình Java. Tuy không phải là một đề tài mới nhưng đây sẽ là cơ sở để chúng em có thể xây dựng và phát triển những đề tài khác trong tương lai.

9 TÀI LIỆU THAM KHẢO

[1] Cryptography and Network Security Principles and Practice, Seven Edition- William Stallings

[2] Securing File Transferring System by Implementing AES Algorithm, Nor Surayati Mohamad Usop, Ahmad Faisal Abidin, Fauziah Ab. Wahab and Norlina Udin