BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC DÂN LẬP HẢI PHÒNG
---------o0o---------

TÌM HIỂU VÀ XÂY DỰNG ỨNG DỤNG MÃ HÓA KHÓA ĐỐI XỨNG BẰNG THUẬT TOÁN RIJNDAEL

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
NGÀNH CÔNG NGHỆ THÔNG TIN

Sinh viên thực hiên: Đỗ Thị Bích Thủy
Giáo viên hướng dẫn: Ths. Lê Thụy
Mã số sinh viên: 111339

LỜI CẢM ƠN

Để hoàn thành đồ án này, trước hết, em xin gửi lời cảm ơn và biết ơn sâu sắc tới thầy giáo Lê Thụy, người đã tận tình hướng dẫn, chỉ bảo và giúp đỡ em trong suốt thời gian nghiên cứu và hoàn thành đồ án.

Em xin chân thành cảm ơn tới các thầy cô trong khoa Công Nghệ Thông Tin cũng như các thầy cô trong trường Đại học dân lập Hải Phòng, những người đã tận tình giảng dậy, và tạo điều kiện cho em trong suốt quá trình học tập và nghiên cứu tại trường.

Cuối cùng, em xin cảm ơn gia đình, bạn bè, người thân đã luôn ở bên động viên và là nguồn cổ vũ lớn lao, là động lực trong suốt quá trình học tập và nghiên cứu.

Mặc dù em đã cố gắng hoàn thành đồ án trong phạm vi và khả năng có thể. Tuy nhiên sẽ không tránh khỏi những điều thiếu sót. Em rất mong nhận được sự cảm thông và tận tình chỉ bảo của quý thầy cô và toàn thể các bạn.

Một lần nữa em xin chân thành cảm ơn !

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

MỞ ĐẦU

Từ khi con người có nhu cầu trao đổi thông tin, thư từ với nhau thì nhu cầu giữ bí mật và bảo mật tính riêng tư của những thông tin, thư từ đó cũng nảy sinh. Hình thức thông tin trao đổi phổ biến sớm nhất là dưới dạng các văn bản, để giữ bí mật của thông tin người ta đã sớm nghĩ đến cách che dấu nội dung các văn bản bằng cách biến dạng các văn bản đó để người ngoài đọc nhưng không hiểu được, đồng thời có cách khôi phục lại nguyên dạng ban đầu để người trong cuộc vẫn hiểu được; theo cách gọi ngày nay thì dạng biến đổi của văn bản được gọi là *mật mã* của văn bản, cách lập mã cho một văn bản được gọi là *phép lập mã*, còn cách khôi phục lại nguyên dạng ban đầu gọi là *phép giải mã.* Phép lập mã và phép giải mã được thực hiện nhờ một chìa khóa riêng nào đó mà chỉ những người trong cuộc được biết và nó được gọi là *khóa lập mã*. Người ngoài dù có lấy được bản mật mã trên đường truyền mà không có khóa mật mã thì cũng không thể hiểu được nội dung của văn bản truyền đi.

Có rất nhiều thuật toán đã được đưa ra nhằm mục đích bảo mật thông tin với độ an toàn cao. Và một trong số các thuật toán đó có thuật toán mã hóa đối xứng Rijndael được Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (National Institute of Standards and Technology – NIST) chọn làm chuẩn mã hóa nâng cao (Advanced Encryption Standard) từ 02 tháng 10 năm 2000. Vì đó mà em chọn đề tài "Tìm hiểu và xây dựng ứng dụng mã hóa đối xứng bằng thuật toán Rijndael" để hiểu rõ các bước thực hiện trong thuật toán mới này khi mã hóa thông tin.

.Luận văn gồm phần mở đầu, kết luận và 3 chương với các nội dung chính sau:

- Chương 1: Cơ sở lý thuyết về toán học.

- Chương 2: Nói về vấn đề mã hóa bao gồm giới thiệu về mật mã, các khái niệm về mã hóa, các phương pháp mã hóa, chữ ký số và hàm băm.

- Chương 3: Tìm hiểu thuật toán Rijndael và mô phỏng chương trình ứng dụng.

CHƯƠNG 1: CƠ SỞ TOÁN HỌC

1.1 Các khái niệm toán học

1.1.1. Số nguyên tố và số nguyên tố cùng nhau.

- Số nguyên tố là số nguyên dương lớn hơn 1chỉ chia hết cho 1 và chính nó.

Ví dụ: 2, 3, 5, 7, 11, … là những số nguyên tố.

- Hệ mật mã thường sử dụng các số nguyên tố ít nhất là lớn hơn 10^{150}.

- Hai số m và n được gọi là nguyên tố cùng nhau nếu ước số chung lớn nhất của chúng bằng 1. Ký hiệu: gcd(m, n) = 1.

Ví dụ: 11 và 13 là nguyên tố cùng nhau.

Định lý số nguyên tố: Với mọi n>=2 đều có thể phân tích thành lũy thừa cơ số nguyên tố $n = p_1^{e1}p_2^{e2}p_3^{e3}...$, với p_i : số nguyên tố, $e_i \in Z^+$.

Hệ quả: Giả sử $a = p_1^{e1}.p_2^{e2}p_3^{e3}\ldots p_k^{ek}$

$$b = p_1^{f1}.p_2^{f2}.p_3^{f3}...p_k^{fk}$$

thì $\gcd(a,b) = p_1^{\min(e1,f1)}.p_2^{\min(e2,f2)}\ldots p_k^{\min(ek,fk)}$ (1.1)

$$\text{lcm}(a,b) = p_1^{\max(e1,f1)}.p_2^{\max(e2,f2)}\ldots p_k^{\max(ek,\,fk)}$$ (1.2)

Ví dụ: $a = 4864 = 2^8.19$ và $b = 3458 = 2.7.13.19$

ta được : $\gcd(a,b) = 2.19$ và $\text{lcm}(a,b) = 2^8.19.7.13$

1.1.1 Khái niệm đồng dư

Cho n là một số nguyên dương. Nếu a và b là hai số nguyên, khi đó a được gọi là đồng dư với b theo modulo n, được viết $a \equiv b \pmod n$ nếu $n \mid (a - b)$, và n được gọi là modulo của đồng dư.

Ví dụ: $24 \equiv 9 \pmod 5$, $17 \equiv 5 \pmod 3$

Tính chất:

(i) $a \equiv b \pmod n$, nếu và chỉ nếu a và b đều trả số dư như nhau khi đem chia chúng cho n.

(ii) $a \equiv a \pmod n$(tính phản xạ).

(iii) Nếu $a \equiv b \pmod n$ thì $b \equiv a \pmod n$.

(iv) Nếu a ≡ b (mod n) và b ≡ c (mod n) thì a ≡ c (mod n).

(v) Nếu a ≡ a_1 (mod n) và b ≡ b_1 (mod n) thì a + b ≡ (a_1 + b_1) (mod n) và a.b ≡ a_1.b_1 (mod n).

1.1.2 Định nghĩa Phi Euler

Với n ≥ *1*, đặt **ϕ(*n*)** là số các số nguyên trong khoảng [1, n] và nguyên tố cùng nhau với n. Hàm **ϕ** như thế được gọi là hàm ***phi-Euler***.

Tính chất:

- Nếu p là số nguyên tố thì **ϕ**(p) = p-1 (1.3)

- Nếu gcd(n.m) = 1, thì **ϕ**(m.n) = **ϕ**(m) . (n) (1.4)

- Nếu n = p_1^{e1} .p_2^{e2}…p_k^{ek}, dạng khai triển chính tắc của n, thì

$$\mathbf{(n)} = n\left(1 - \frac{1}{p1}\right)\left(1 - \frac{1}{p2}\right)...\left(1 - \frac{1}{pk}\right) \quad (1.5)$$

Ví dụ: **ϕ**(*11*) = 11-1 = 10

1.1.3 Thuật toán Euclide

Thuật toán: Thuật toán Euclide, tính ước số chung lớn nhất của hai số.

INPUT: Hai số nguyên không âm *a* và *b* sao cho *a* ≥ *b*.

OUTPUT: Ước số chung lớn nhất của *a* và *b*.

1. Trong khi *b* ≠ *0*, thực hiện

   Đặt *r* ← *a mod b*, *a* ← *b*, *b* ← *r*.

2. Kết_quả(*a*)

*Ví dụ***:** Tính gcd(*4864, 3458*) = *38*:

4864 = 1.3458 + 1406
3458 = 2.1406 + 646
1406 = 2.646 + 114
646 = 5.114 + 76
114 = 1.76 + 38
76 = 2.38 + 0.

Thuật toán Euclidean có thể được mở rộng để không chỉ tính được ước số chung *d* của hai số nguyên *a* và *b*, mà còn có thể tính được hai số nguyên *x*, *y* thoả mãn: $ax + by = d$

***Thuật toán Euclidean mở rộng**

INPUT: Hai số nguyên không âm *a* và *b* với $a \geq b$.

OUTPUT: $d = \gcd(a, b)$ và hai số *x*, *y* thoả mãn $ax + by = d$.

1. Nếu $b = 0$, đặt $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, Kết_quả(d, x, y).

2. Đặt $x2 \leftarrow 1$, $x1 \leftarrow 0$, $y2 \leftarrow 0$, $y1 \leftarrow 1$.

3. Trong khi còn $b > 0$, thực hiện:

3.1 $q \leftarrow \lfloor a/b \rfloor$, $r \leftarrow a - q.b$, $x \leftarrow x2 - q.x1$, $y \leftarrow y2 - q.y1$

3.2 $a \leftarrow b$, $b \leftarrow r$, $x2 \leftarrow x1$, $x1 \leftarrow x$, $y2 \leftarrow y1$, $y1 \leftarrow y$

4. Đặt $d \leftarrow a$, $x \leftarrow x2$, $y \leftarrow y2$, Kết_quả(d, x, y).

1.1.4 Không gian Z_n và Z_n^*

1.1.4.1 Không gian Z_n (các số nguyên theo modulo n)

Là tập hợp các số nguyên {0, 1, 2, …, n-1}. Các phép toán trong Z_n như cộng, trừ, nhân, chia đều được thực hiện theo module n.

Ví dụ: Z_{21} = {0, 1, 2, 3, …,20}

1.1.4.2 Không gian Z_n^*

Là tập hợp các số nguyên $a \in Z_n$, nguyên tố cùng n. Tức là: $Z_n^* = \{a \in Z_n \mid \gcd(n, a) = 1\}$, **(n)** là số phần tử của Z_n^*.

Nếu n là một số nguyên tố thì: $Z_n^* = \{a \in Zn \mid 1 \leq a \leq n-1\}$

(1.6)

Ví dụ: $Z_3 = \{0, 1,2\}$ thì $Z_3^* = \{1,2\}$ vì gcd(1, 3) = 1và gcd(2,3) = 1.

1.1.5 Định nghĩa cấp của một số $a \in Z_n^*$

Cho $\alpha \in Z_n^*$, khi đó cấp của a, kí hiệu ord(a) là số nguyên dương nhỏ nhất sao cho $a^t \equiv 1 (\text{mod } n)$ trong Z_n^*.

1.1.6 Khái niệm Nhóm, Nhóm con, Nhóm Cyclic

1.1.6.1 Khái niệm Nhóm

Nhóm là một bội (G, *), trong đó G ≠ ∅, * là *phép toán hai ngôi* trên G thỏa mãn ba tính chất sau:

+ Phép toán có tính kết hợp: (x*y)*z = x*(y*z) với mọi x, y, z ∈ G. (1.7)

+ Có phần tử *trung lập* e ∈ G: x*e = e*x = x với mọi x ∈ G. (1.8)

+ Với mọi x ∈ G, có phần tử nghịch đảo x’ ∈ G: x*x’ = x’*x = e. (1.9)

Cấp của nhóm G được hiểu là số phần tử của nhóm, ký hiệu là |G|. Cấp của nhóm có thể là ∞ nếu G có vô hạn phần tử.

Nhóm Abel là nhóm (G, *), trong đó phép toán hai ngôi * có tính giao hoán.

Tính chất: Nếu a*b = a*c, thì b = c.

Nếu a*c = b*c, thì a = b.

Ví dụ:

+) Tập hợp các số nguyên Z cùng với phép cộng (+) thông thường là nhóm giao hoán, có phần tử đơn vị là số 0. Gọi là ***nhóm cộng*** các số nguyên.

+)Tập Q^* các số hữu tỷ khác 0 (hay tập R^* các số thực khác 0), cùng với phép nhân (*) thông thường là nhóm giao hoán. Gọi là ***nhóm nhân*** các số hữu tỷ (số thực).

+)Tập các vectơ trong không gian với phép toán cộng vectơ là nhóm giao hoán.

1.1.6.2 Nhóm con của nhóm (G, *)

Nhóm con của G là tập S ⊂ G, S ≠ ϕ, và thỏa mãn các tính chất sau:

+ Phần tử trung lập e của G nằm trong S.

+ S khép kín đối với phép tính (*) trong G, tức là x*y ∈ S với mọi x, y ∈ S.

+ S khép kín đối với phép lấy nghịch đảo trong G, tức x^{-1} ∈ S với mọi x ∈ S.

1.1.6.3 Nhóm Cyclic

Cho $\alpha \in Z_n^*$, nếu cấp của α là (n), khi đó α được gọi là phần tử sinh hay phần tử nguyên thủy của Z_n^*. Nếu Z_n^* có một phần tử sinh, thì Z_n^* được gọi là nhóm Cyclic. (chú ý nếu n là số nguyên tố thì $\phi(n) = n-1$)

Tính chất:

- Nếu α là phần tử sinh của Z_n^* thì $Z_n^* = \{\alpha^i \bmod n \mid 0 \le i \le (n) - 1\}$

- Giả sử α là một phần tử sinh của Z_n^*, khi đó $b = \alpha^i \bmod n$ cũng là một phần tử sinh của Z_n^* khi và chỉ khi gcd(i, $\phi(n)$) = 1. Và sau đó nếu Z_n^* là nhóm Cyclic thì số phàn tử sinh sẽ là ((n)).

- $\alpha \in Z_n^*$ là phần tử sinh của Z_n^* khi và chỉ khi $\alpha^{\phi(n)/p} \not\equiv 1 \pmod n$ với mỗi số chia nguyên tố của $\phi(n)$.

- Z_n^* có phần tử sinh khi và chỉ khi n = 2, 4, p^k hay $2p^k$ khi p là số nguyên tố lẻ và $k \ge 1$. Còn nếu p là số nguyên tố thì chắc chắn Z_p^* có phần tử sinh.

Ví dụ: Z_{21}^* không phải là nhóm Cyclic vì không phần tử nào của Z_{21}^* có cấp là $\varphi(21) = 12$, *chú ý là 21* không thỏa mãn điều kiện nào theo tính chất của phần tử sinh trên. Trong khi đó Z_{13}^* là nhóm Cyclic và có phần tử sinh α = 2

Thật vậy:

$2^0 \bmod 13 = 1$	$2^1 \bmod 13 = 2$	$2^2 \bmod 13 = 4$
$2^3 \bmod 13 = 8$	$2^4 \bmod 13 = 3$	$2^5 \bmod 13 = 6$
$2^6 \bmod 13 = 12$	$2^7 \bmod 13 = 11$	$2^8 \bmod 13 = 9$
$2^9 \bmod 13 = 5$	$2^{10} \bmod 13 = 10$	$2^{11} \bmod 13 = 7$

Phần tử 2^i là sinh khi và chỉ khi gcd(i, 12) = 1 nghĩa là khi và chỉ khi i =1, 5, 7 hoặc 11. Vậy các phần tử sinh của Z_{13}^* là 2, 6, 7 và 11.

1.1.7 Tập thặng dư bậc hai theo modulo

Định nghĩa: Cho $a \in Z_n^*$, a được gọi là thặng dư bậc hai theo modulo n nếu tồn tại một $x \in Z_n^*$ sao cho $x^2 \equiv a \pmod n$, và nếu không tồn tại x như vậy thì a được gọi là bất thặng dư bậc hai theo modulo n. Tập hợp các thặng dư bậc hai được kí hiệu là Q_n và tập các bất thặng dư bậc hai ký hiệu là $\overline{Qn}$.

Ví dụ: α = 6 là phần tử sinh của Z_{13}^* ta có:

i	0	1	2	3	4	5	6	7	8	9	10	11
$\alpha^i \bmod 13$	1	6	10	8	9	2	12	7	3	5	4	11

Do đó Q_{13} = {1, 3, 4, 9, 10, 12} và $\overline{Qn}$ = {2, 5, 6, 7, 8, 11}

1.1.8 Phần tử nghịch đảo

Định nghĩa: Cho a ∈ Z_n, số nghịch đảo của a theo modulo n là một số nguyên x ∈ Z_n, nếu a.x ≡ 1 (mod n). Nếu tồn tại x như vậy, thì nó là duy nhất và a được gọi là khả nghịch, nghịch đảo của a được kí hiệu là a^{-1}.

Tính chất: a ∈ Z_n, a là khả nghịch khi và chỉ khi gcd(a, n) = 1.

Ví dụ: Các phần tử khả nghịch trong Z_9 là 1, 2, 4, 5, 7 và 8. Cho ví dụ, 4^{-1}=7 vì 4.7≡ 1(mod 9).

Thuật toán tính nghịch đảo trên Z_n

Input: a ∈ Z_n .

Output: a^{-1} mod n, nếu tồn tại,

1. Sử dụng thuật toán Euclidean mở rộng, tìm x và y để ax + ny = d, trong đó d = gcd(a, n).
2. Nếu d >1, thì a^{-1} mod n không tồn tại, Ngược lại, kết quả(x).

1.2 Khái niệm Độ phức tạp của thuật toán

1.2.1 Khái niệm Thuật toán

Thuật toán là một dãy hữu hạn các quy tắc (chỉ thị, mệnh lệnh) mô tả chính xác một quá trình tính toán. Theo đó với mỗi bộ dữ liệu vào sẽ cho một kết quả (Yêu cầu của bài toán).[1]

Các đặc trưng của Thuật toán đơn định:

- Tính đơn định: Thực hiện đúng các bước của thuật toán với một dữ liệu vào thì chỉ cho duy nhất một kết quả nghĩa là ở mỗi bước của thuật toán, các thao tác phải hết sức rõ ràng, không gây nên sự nhập nhằng, lộn xộn, đa nghĩa....

- Tính dừng: Thuật toán phải dừng và cho ra kết quả sau một số hữu hạn các bước.

- Tính đúng: Cho ra kết quả phù hợp yêu cầu bài toán với những dữ liệu vào đúng đắn.

- Tính phổ dụng: Thuật toán phải giải quyết được một lớp rộng các bài toán.

- Tính khả thi: Thuật toán phải được máy tính thực hiện trong khoảng thời gian và điều kiện (bộ nhớ) cho phép.

1.2.2 Độ phức tạp của thuật toán

Thông thường để đánh giá thuật toán người ta dựa trên hai tiêu chuẩn sau:

Tiêu chuẩn 1: Độ đơn giản, dễ hiểu, dễ cài đặt (viết chương trình).

Tiêu chuẩn 2: Sử dụng tiết kiệm tài nguyên hệ thống và với thời gian ngắn nhất.

Độ phức tạp của thuật toán là phương pháp đánh giá thuật toán theo hướng xấp xỉ tiệm cận qua các khái niệm toán học O lớn →O(); o nhỏ → o(); Ω(); ≡().

Hầu hết tất cả các thuật toán có thời gian chạy tiệm cận tới một trong các hàm sau:

a. **Hằng số**: Hầu hết các chỉ thị của các chương trình đều được thực hiện một lần hay nhiều nhất chỉ một vài lần. Nếu tất cả các chỉ thị của cùng một chương trình có tính chất này thì chúng ta sẽ nói rằng thời gian chạy của nó là hằng số. Điều này hiển nhiên là điều mà ta phấn đấu để đạt được trong việc thiết kế thuật toán.

b. **LogN**: Khi thời gian chạy của chương trình là **logarit** tức là thời gian chạy chương trình tiến chậm khi N lớn dần. Thời gian chạy thuộc loại này xuất hiện trong các chương trình mà giải một bài toán lớn bằng cách chuyển nó thành một bài toán nhỏ hơn, bằng cách cắt bớt kích thước một hằng số nào đó. Với mục đích của chúng ta, thời gian chạy có được xem như nhỏ hơn một hằng số "lớn". Cơ số của logarit làm thay đổi hằng số đó nhưng không nhiều: Khi N là 1000 thì logN là 3 nếu cơ số là 10, là 10 nếu cơ số là 2; khi N là một triệu, logN được nhân gấp đôi. bất cứ khi nào N được nhân đôi, logN tăng lên thêm một hằng số.

c. **N**: Khi thời gian chạy của một chương trình là tuyến tính, nói chung đây là trường hợp mà một số lượng nhỏ các xử lý được làm cho mỗi phần tử dữ liệu nhập. Khi N là một triệu thì thời gian chạy cũng cỡ như vậy. Khi N được nhân gấp đôi thì thời gian chạy cũng được nhân gấp đôi. Đây là tình huống tối ưu cho một thuật toán mà phải xử lý N dữ liệu nhập (hay sản sinh ra N dữ liệu xuất).

d. **NlogN**: Đây là thời gian chạy tăng dần lên cho các thuật toán mà giải một bài toán bằng cách tách nó thành các bài toán con nhỏ hơn, kế đến giải quyết chúng một cách độc lập và sau đó tổ hợp các lời giải. Chúng ta nói rằng thời gian chạy của

thuật toán như thế là "NlogN". Khi N là một triệu, NlogN khoảng 20 triệu. khi N được nhân gấp đôi, thời gian chạy bị nhân lên nhiều hơn gấp nhiều đôi.

e. **N^2**: Khi thời gian chạy của một thuật toán là bậc hai, trường hợp này chỉ có ý nghĩa thực tế cho các bài toán tương đối nhỏ. Thời gian bình phương thường tăng dần lên trong các thuật toán mà xử lý tất cả các phần tử dữ liệu (có thể là hai vòng lặp lồng nhau). Khi n là một ngàn thì thời gian chạy là 1 triệu. khi N được nhân đôi thì thời gian chạy tăng lên gấp 4 lần.

f. **N^3**: Tương tự, một thuật toán mà xử lý các bộ ba của các phần tử dữ liệu (có thể là 3 vòng lặp lồng nhau) có thời gian chạy bậc ba và cũng chí ý nghĩa thực tế trong các bài toán nhỏ. Khi N là một trăm thì thời gian chạy là một triệu. Khi N được nhân đôi thì thời gian chạy tăng lên gấp 8 lần.

g. **2N**: Một số ít thuật toán có thời gian chạy lũy thừa lại thích hợp trong một số trường hợp thực tế. Khi N là hai mươi thì thời gian chạy là 1 triệu. khi N tăng gấp đôi thì thời gian chạy được nâng lên luỹ thừa hai!

Thời gian chạy của một chương trình cụ thể đôi khi là một hệ số hằng nhân với các số hạng nói trên ("số hạng dẫn đầu") cộng thêm một số hạng nhỏ hơn. Giá trị của hệ số hằng và các số hạng phụ thuộc vào kết quả của sự phân tích và các chi tiết cài đặt. Hệ số của số hạng dẫn đầu liên quan tới số chỉ thị bên trong vòng lặp: Ở một tầng tùy ý của thiết kế thuật toán thì phải cẩn thận giới hạn số chỉ thị như thế. Với N lớn thì các số hạng dẫn đầu đóng vai trò chủ chốt; với N nhỏ thì các số hạng cùng đóng góp vào sự so sánh các thuật toán sẽ khó khăn hơn. Trong hầu hết các trường hợp, chúng ta sẽ gặp các chương trình có thời gian chạy là "tuyến tính", "NlogN", "bậc ba",… với hiểu ngầm là các phân tích hay nghiên cứu thực tế phải được làm trong trường hợp mà tính hiệu quả là rất quan trọng.

1.2.3 Ví dụ về việc xác định độ phức tạp của thuật toán:

Ví dụ với bài toán tính tổng các số nguyên dương từ 1 đến n ta có thể tính theo thuật toán sau:

```
Input n;
Tong:=0;
For i:= 1 to n do Tong := Tong + i;
Output Tong;
```

Với thuật toán này thời gian tính toán tỷ lệ thuận với n, khi n càng lớn thì thời gian càng tốn hay độ phức tạp tính toán là O(n)

Nếu tính tổng các số nguyên dương từ 1 đến n theo thuật toán:

```
Input n;
Tong := n*(n+1)/2;
Output Tong;
```

Thì độ phức tạp tính toán này là O(1) không phụ thuộc vào giá trị n

CHƯƠNG 2: VẤN ĐỀ MÃ HÓA

2.1 Mật mã học

2.1.1 Giới thiệu chung

Mật mã học là ngành khoa học ứng dụng toán học vào việc biến đổi thông tin thành một dạng khác với mục đích che dấu nội dung, ý nghĩa thông tin cần mã hóa. Đây là một ngành quan trọng và có nhiều ứng dụng trong đời sống xã hội. Ngày nay, các ứng dụng mã hóa và bảo mật thông tin đang được sử dụng ngày càng phổ biến hơn trong các lĩnh vực khác nhau trên thế giới, từ các lĩnh vực an ninh, quân sự, quốc phòng…, cho đến các lĩnh vực dân sự như thương mại điện tử, ngân hàng…

Cùng với sự phát triển của khoa học máy tính và Internet, các nghiên cứu và ứng dụng của khoa học mật mã ngày càng trở nên đa dạng hơn, mở ra nhiều hướng nghiên cứu chuyên sâu vào từng lĩnh vực ứng dụng đặc thù với những đặc trưng riêng.

Ứng dụng của khoa học mật mã không chỉ đơn thuần là mã hóa và giải mã thông tin mà còn bao gồm nhiều vấn đề khác nhau cần được nghiên cứu và giải quyết: chứng thực nguồn gốc nội dung thông tin (kỹ thuật chữ ký điện tử), chứng nhận tính xác thực về người sở hữu mã khóa (chứng nhận khóa công cộng), các quy trình giúp trao đổi thông tin và thực hiện giao dịch điện tử an toàn trên mạng... Những kết quả nghiên cứu về mật mã cũng đã được đưa vào trong các hệ thống phức tạp hơn, kết hợp với những kỹ thuật khác để đáp ứng yêu cầu đa dạng của các hệ thống ứng dụng khác nhau trong thực tế, ví dụ như hệ thống bỏ phiếu bầu cử qua mạng, hệ thống đào tạo từ xa, hệ thống quản lý an ninh của các đơn vị với hướng tiếp cận sinh trắc học, hệ thống cung cấp dịch vụ multimedia trên mạng với yêu cầu cung cấp dịch vụ và bảo vệ bản quyền sở hữu trí tuệ đối với thông tin số...

2.1.2 Định nghĩa

Mật mã học là sự nghiên cứu các phương pháp toán học liên quan đến một số khía cạnh của thông tin như sự an toàn, sự toàn vẹn dữ liệu, sự xác nhận tồn tại và sự xác nhận tính nguyên bản của thông tin.[2]

2.2 Khái niệm hệ mật mã

Một sơ đồ hệ thống mật mã là bộ năm S = (P, C, K, E, D) thỏa mãn các điều kiện:

P: là một tập hữu hạn các ký tự bản rõ.

C: là một tập hữu hạn các ký tự bản mã.

K: là một tập hữu hạn các khóa.

E: là một ánh xạ từ KxP vào C, được gọi là phép lập mật mã.

D: là một ánh xạ từ KxC vào P, được gọi là phép giải mã.

Với $k \in K$ ta định nghĩa $e_k \in E$, $e_k: P \rightarrow C$; $d_k \in D$, $d_k: C \rightarrow P$; e_k, d_k được gọi là hàm lập mã và hàm giải mã tương ứng với khóa mật mã k. Các hàm đó phải thỏa mãn hệ thức: $d_k(e_k(x)) = x$ với $\forall x \in P$. Tính chất này bảo đảm một mẩu tin $x \in P$ được mã hóa bằng luật mã hóa $e_k \in E$ có thể được giải mã chính xác bằng luật $d_k \in D$.

2.3 Khái niệm mã hóa (Encryption), giải mã (Decryption)

Mã hóa: là quá trình chuyển thông tin có thể đọc được (gọi là bản rõ) thành thông tin "khó" thể đọc được theo cách thông thường (gọi là bản mã). Đó là một trong những kỹ thuật để bảo mật thông tin.

Giải mã: là quá trình chuyển thông tin ngược lại từ bản mã thành bản rõ.

Thuật toán mã hóa hay giải mã là thủ tục để thực hiện mã hóa hay giải mã.

Khóa mã hóa là một giá trị làm cho thuật toán mã hóa thực hiện theo cách riêng biệt và sinh ra bản rõ riêng. Thông thường khóa càng lớn thì bản mã càng an toàn. Phạm vi các giá trị có thể có của khóa được gọi là Không gian khóa.

Hệ mã hóa là tập các thuật toán, các khóa nhằm che giấu thông tin, cũng như làm rõ nó.

2.4 Những tính năng của hệ mã hóa

+ Tính bảo mật: Bảo đảm bí mật cho các thông báo và dữ liệu bằng việc che giấu thông tin nhờ các kỹ thuật mã hóa.

+ Tính toàn vẹn: Bảo đảm với các bên rằng bản tin không bị thay đổi trên đường truyền tin.

+ Chống chối bỏ: Có thể xác nhận rằng tài liệu đã đến từ ai đó, ngay cả khi họ cố gắng từ chối nó.

+ Tính xác thực: Cung cấp hai dịch vụ:

- Nhận dạng nguồn gốc của một thông báo, đảm bảo rằng nó là đúng sự thực.

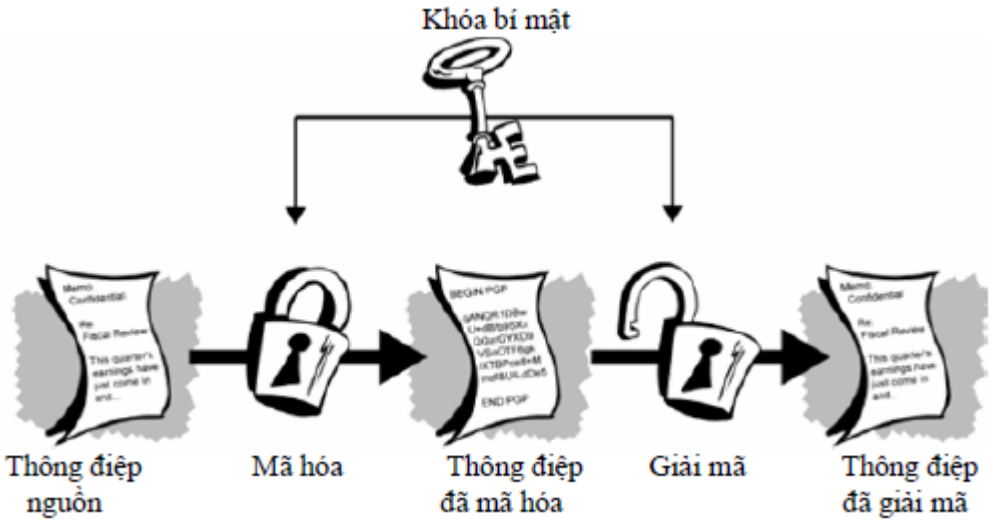
- Kiểm tra định danh của người đang đăng nhập hệ thống, tiếp tục kiểm tra đặc điểm của họ trong trường hợp ai đó cố gắng kết nối và giả danh là người sử dụng hợp pháp.

2.5 Các phương pháp mã hóa

2.5.1 Phương pháp mã hóa đối xứng

Khái niệm: *Hệ mã hóa khóa đối xứng* là hệ mã hóa mà biết được khóa lập mã thì có thể "dễ" tính được khóa giải mã và ngược lại. Đặc biệt một số hệ mã hóa có khóa lập mã và khóa giải mã trùng nhau (ke = kd), như hệ mã hóa "dịch chuyển" hay DES. Hệ mã hóa khóa đối xứng còn gọi là *Hệ mã hóa khóa bí mật*, hay *khóa riêng*, vì phải giữ bí mật cả 2 khóa. Trước khi dùng hệ mã hóa khóa đối xứng, người gửi và người nhận phải thỏa thuận thuật toán mã hóa và *khóa chung* (lập mã hay giải mã), khóa phải được bí mật.

Độ an toàn của Hệ mã hóa loại này ***phụ thuộc vào khóa***, nếu để lộ ra khóa này nghĩa là bất kỳ người nào cũng có thể mã hóa và giải mã thông báo trong hệ thống mã hóa.Sự mã hóa và giải mã của hệ thống mã hóa khóa đối xứng biểu thị bởi: E_k: P → C và D_k: C → P



Hình 2.1: Mô hình hệ thống mã hõa đối xứng

Ví dụ:

+ *Hệ mã hóa cổ điển* là Mã hóa khóa đối xứng: dễ hiểu, dễ thực thi, nhưng có độ an toàn không cao. Vì giới hạn tính toán chỉ trông phạm vi bảng chữ cái, sử dụng trong bản tin cần mã, ví dụ Z_{26} nếu dùng các chữ cái tiếng anh. Với hệ mã hóa cổ điển, nếu biết khóa lập mã hay thuật toán lập mã, có thể "dễ" xác định được bản rõ, vì "dễ" tìm được khóa giải mã.

+ *Hệ mã hóa* DES (1973) là Mã hóa khóa đối xứng *hiện đại*, có độ an toàn cao.

Ưu điểm: Hệ mã hóa khóa đối xứng mã hóa và giải mã nhanh hơn Hệ mã hóa khóa công khai.

Nhược điểm:

(i). Mã hóa khóa đối xứng chưa thật an toàn với lý do sau: Người mã hóa và người giải mã có "chung" một khóa. Khóa phải được giữ bí mật tuyệt đối, vì biết khóa này "dễ" xác định được khóa kia và ngược lại.

(ii). Vấn đề thỏa thuận khóa và quản lý khóa chung là khó khăn và phức tạp. Người gủi và người nhận phải luôn thống nhất với nhau về khóa. Việc thay đổi khóa là rất khó và dễ bị lộ. Khóa chung phải được gửi cho nhau trên kênh an toàn.

Mặt khác khi hai người (lập mã, giải mã) cùng biết "chung" một bí mật, thì càng khó giữ được bí mật!

Nơi sử dụng hệ mã hóa khóa đối xứng.

Hệ mã hóa khóa đối xứng thường được sử dụng trong môi trường mà khóa chung có thể dễ dàng trao chuyển bí mật, chẳng hạn trong cùng một mạng nội bộ. Hệ mã hóa khóa đối xứng thường dùng để mã hóa những bản tin lớn, vì tốc độ mã hóa và giải mã nhanh hơn hệ mã hóa công khai.

2.5.1.1 Mã khối (Block cipher)

Mật mã khối được cấu trúc trên nguyên tắc là bản tin được chia thành các khối có độ dài bằng nhau và việc mã hoá tiến hành theo từng khối độc lập nhau. Trong môi trường máy tính độ dài của khối được tính bằng bit.

Độ bảo mật của mã trong trường hợp này phụ thuộc vào độ dài của khối và độ phức tạp của thuật toán mã. Nếu kích cỡ của khối quá bé thì việc giải mã không mấy khó khăn do dò tìm được đặc tính cấu trúc thống kê của bản tin rõ. Nếu tăng

kích thước khối thì mức độ cấu trúc thống kê cũng tăng theo số mũ và nếu kích cỡ khối tiến đến đoạn tin thì tác dụng mã khối sẽ giảm.[2]

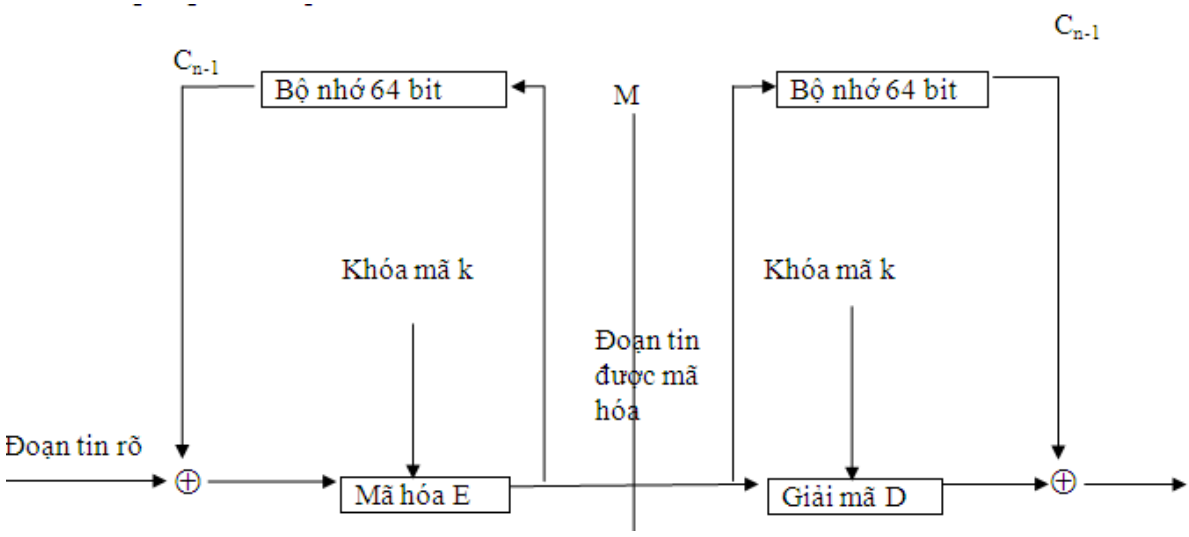
Các phương pháp ứng dụng của mật mã khối : Mật mã khối xử lý các khối dữ liệu có độ dài cố định và độ dài bản tin có thể bất kỳ. Có bốn phương pháp ứng dụng mã khối thường gặp trong hệ thống truyền tin và số liệu truyền tin:[3]

a. Phương pháp dùng từ điển điện tử, còn gọi là mật mã ECB (*Electronic CodeBook*)

Mật mã ECB sử dụng trực tiếp thuật toán để mã hóa từng khối tin một. Mật mã ECB sử dụng từ điển điện tử có một số giới hạn nhất định bởi vì trong các ứng dụng thực tế có những mẩu đoạn tin có xu hướng lặp lại. Ngay các đoạn tin từ các máy tính cũng có tính chất lặp lại, nó có thể chứa những dãy 0 hoặc khoảng trống. Các định nghĩa về thể thức cho các tham số là các giá trị hằng số, đoạn tin có khuôn dạng cố định với các dữ liệu quan trọng luôn cùng vị trí. Đối phương có thể sử dụng các dạng dữ liệu tương tự để phát hiện các tính quy luật. Mặt khác sự yếu kém của mã còn ở chỗ các khối tin được mã hóa riêng rẽ, chúng không có quan hệ với nhau.

b. Phương pháp móc xích các khối đã được mã hoá, còn gọi là mật mã CBC (*Cipher Block Chaining*)

Phương pháp mật mã CBC sử dụng đầu ra của phép toán mã hóa để biến đổi đầu vào tiếp theo. Như vậy mỗi một khối được mã hóa không những chỉ phụ thuộc vào đoạn tin rõ tương ứng mà còn phụ thuộc vào tất cả các khối của đoạn tin rõ trước nó.



Hình 2.2: Mô tả sơ đồ chức năng của mật mã CBC(mã hóa và giải mã).

Tất cả các phép toán được thực hiện với 64 bit song song. Phần bên trái của hình vẽ đặc trưng cho sự móc xích khối dữ liệu đã được mã hóa ở đầu ra với khối dữ liệu rõ ở đầu vào. Trừ khối đầu tiên, mỗi một khối trước khi mã hóa sẽ được cộng modul -2 với khối đã được mã hóa trước đó, tức khối thứ n được mã hóa thành C_n phụ thuộc vào tất cả các khối dữ liệu rõ $P_{1...}P_n$. Phần bên phải của hình vẽ là quá trình giải mã theo phương pháp ngược lại. Ở đây, sau khi giải mã sẽ thực hiện phép công modul-2 với khối đã được mã hóa trước đó để có dữ liệu rõ ban đầu. Trong quá trình thực hiện, mỗi một bit của khối dữ liệu vào, P_n được cộng modul-2 với 1 bit tương ứng của khối dữ liệu đã được mã hóa trước đó, C_{n-1}. Trong quá trình thực hiện phép toán có thể thực hiện theo phương pháp nối tiếp hoặc song song từng byte một.

Có thể giải thích phương pháp mật mã CBC như sau:

Với phép mã hóa: $C_n = Ek(P_n \oplus C_{n-1})$

Với phép giải mã: $Q_r = Dk(C_n) \oplus C_{n-1.}$

Phương pháp mật mã CBC có thể được đặc trưng như một sự phản hồi bản tin đã mã hóa về phía phát và một sự phản hồi về phía thu. Quá trình phản hồi đó dẫn đến 1 điều cần lưu ý là nếu có 1 bit lỗi trong bản tin sẽ dẫn đến tất cả các khối dữ liệu có quan hệ với khối đó đều bị lỗi.

Mật mã CBC được xây dựng trên cơ sở các khối dữ liệu có quan hệ móc xích với nhau, nhằm khắc phục nhược điểm của phương pháp dùng từ điển. Điều đó chỉ đúng bắt đầu từ khối dữ liệu thứ hai, còn khối dữ liệu đầu tiên lại phụ thuộc vào giá trị khởi đầu (IV). Nếu như giá trị khởi đầu luôn là hằng số với tất cả các bản tin thì điều đó tạo tính quy luật đối phương dễ phát hiện. Trong thực tế truyền tin các giá trị khởi đầu luôn được thay đổi và người ta cũng tránh việc dùng cùng giá trị khởi đầu và cùng khóa mã nhiều lần cho các bản tin được truyền.

c. Phương pháp phản hồi bản tin đã mã hoá, còn gọi là mật mã CFB (*Cipher feedblack*)

Dữ liệu được xử lý và được truyền dưới nhiều dạng khác nhau, chúng có thể dưới dạng các bản tin hoàn chỉnh, các khối dữ liệu, các gói, các ký tự riêng rẽ, theo byte hoặc theo bit. Khi phải xử lý các đoạn tin theo byte hoặc theo bit, người ta thường sử dụng một phương pháp mật mã dưới dạng phản hồi đoạn tin đã mã hóa, được gọi là mật mã CFB. Yêu cầu ở đây là các byte dữ liệu khi xuất hiện ra đường truyền cần được mã hóa ngay và việc mã hóa đó không ảnh hưởng đến sự chậm tốc độ truyền. Ở phía giải mã cũng có yêu cầu giải mã ngay khi một byte dữ liệu đến.

Việc mã hóa chuỗi các ký tự được thực hiện theo phương pháp cộng modul-2 ký tự lấy ở đầu ra của thuật toán DES với ký tự bản tin rõ để tạo thành một ký tự được mã hóa. Ở phía thu cùng thực hiện phép cộng modul-2 của cùng ký tự lấy từ đầu ra của DES với ký tự đã được mã hóa để có ký tự của bản tin rõ. Cấu trúc hệ thống như vậy đảm bảo rằng, dữ liệu được bổ sung thêm là hoàn toàn giả ngẫu nhiên.

Trong khi phương pháp mật mã CBC thực hiện trên các khối dữ liệu hoàn chỉnh thì phương pháp CFB thực hiện mỗi lần theo một ký tự và chiều dài m có thể được chọn như một thông số trước kia. Chiều dài m nhỏ nhất là 1 bit và trường hợp này là mật mã CFC một bit. Về nguyên lý, giá trị m có thể từ 1 đến 64. Hiện nay trên các hệ truyền tin phổ biến nhất là sử dụng m=8. Cũng giống như mật mã CBC, phương pháp mật mã CFB liên kết các ký tự với nhau và làm cho bản tin được mã hóa vào toàn bộ bản tin rõ.

Khởi đầu thực hiện CFB thì thanh ghi dịch chuyển cũng phải có một giá trị khởi đầu, Thường sử dụng các giá trị khác nhau IV cho mỗi đoạn tin và trong thời gian thực hiện một vòng khóa mã để tránh đặc tính chu kỳ. Các giá trị IV có thể truyền tin công khai bởi chúng đã trải qua phép toán mã hóa.

Mật mã CFB cũng được sử dụng để mã hóa một chuỗi các ký tự mà mỗi ký tự được biểu thị bởi m bit. Trong một tập như thế, mỗi ký tự là một số nằm trong khoảng 0 và n-1 với $n = 2^m$. Các ký tự rõ cũng như ký tự đã mã hóa đều nằm trong khoảng đó. Cũng có một số mã kí hiệu không nằm trong khoảng 2m giá trị. Ví dụ trong trường hợp chỉ sử dụng các số thập phân 0, 1,…, 9. Một tập các giá trị như vậy cần lưu ý tránh các giá trị nhị phân tương ứng với các kí tự điều khiển. Ví dụ mã ASCII có các giá trị kí tự điều khiển như: khởi đầu bản tin, kết thúc bản tin, bắt buộc thu, phát, thoát khỏi…mà điều đó dẫn đến hiểu nhầm là thể thức truyền dẫn mạng.

Nếu $n=2^m$ thì có thể sử dụng mật mã CFB bình thường. Trong trường hợp phải mã hóa các ký tự m bit với m là số nguyên khá nhỏ và $n<2^m$ thì việc mã hóa và giải mã có hơi khác một ít.

d. Phương pháp phản hồi đầu ra, còn gọi là OFB (*Output Feedblack)*

Trong số 3 phương pháp mật mã được giới thiệu trên thì phương pháp mật mã ECB sử dụng hạn chế, còn 2 phương pháp mật mã CBC và CFB được sử dụng tương đối thông dụng. Có một phương pháp được dành riêng cho các ứng dụng mà quá trình truyền tin chấp nhận một sai số nào đó. Đó là phương pháp mật mã phản hồi từ đầu ra, OFB. Mật mã OFB thường được sử dụng trong truyền tín hiệu số hóa

âm thanh hoặc số hóa hình ảnh dưới dạng điều chế mã xung PCM, trên nền nhiễu bị sai số. Mã OFB sử dụng phương thức mã hóa kiểu Vernam, chỉ có nguồn giả ngẫu nhiên là mới. Cấu trúc của mã gần giống như mã CFB vì nó có thể được sử dụng với các dãy ký tự m bit với m trong khoảng 1 đến 64.

Mật mã OFB có phương thức thực hiện phản hồi. Chuỗi giả ngẫu nhiên được lấy từ mã DES và được phản hồi về chính nó. Việc đồng bộ cho các bộ tạo giả ngẫu nhiên ở hai đầu đường truyền ở đây cần được chú ý. Nếu như việc đồng bộ không đảm bảo thì bản tin rõ sau khi mã hóa ở phía đầu thu cũng sẽ có dạng ngẫu nhiên. Để tạo lại đồng bộ ở phương pháp mã OFB thì các thanh ghi dịch chuyển phải được nạp các giá trị giống nhau. Các giá trị khởi đầu có thể gửi dưới dạng dữ liệu rõ bởi vì nếu đối phương nếu biết thì cũng không thể tạo được dãy giả ngẫu nhiên. Dãy giả ngẫu nhiên ở đây được cộng modul-2 với đoạn tin trong phép toán mã hóa và giải mã còn được gọi là dòng khóa (key stream).

Các phương pháp ứng dụng của mật mã khối trên được phát triển mạnh sau khi xuất hiện mã DES. Trên thực tế có thể có những phương pháp khác, nhưng bốn phương pháp trên được ứng dụng phổ biến và cũng khá đầy đủ.

2.5.1.2 Mã dòng

Một hệ mã dòng là một bộ (**P**, **C**, **K**, **L**, **F**, **E**, **D**) thoả mãn các điều kiện sau đây:

+ P là một tập hữu hạn các bản rõ.

+ C là một tập hữu hạn các bản mã.

+ K là một tập hữu hạn các khoá chính.

+ L là một tập hữu hạn các khoá dòng.

+ F = {*f1, f2, ...*} là bộ sinh khoá dòng; $fi = \mathbf{K} \times \mathbf{L}^{i-1} \times \mathbf{P}^{i-1} \rightarrow \mathbf{L}$

Với mỗi $z \in \mathbf{L}$ có một hàm lập mã $e_z \in \mathbf{E}$: $\mathbf{P} \rightarrow \mathbf{C}$ và một hàm giải mã $d_z \in \mathbf{D} : \mathbf{C} \rightarrow \mathbf{P}$ sao cho $d_z(e_z(x)) = x$ với mọi $x \in \mathbf{P}$. Nếu bộ sinh dòng khoá không phụ thuộc vào bản rõ, thì ta gọi mã dòng đó là đồng bộ, trong trường hợp đó, **K** như là hạt giống để sinh ra dòng khoá $z_1, z_2, \ldots$. Nếu $z_{i+d} = z_i$ thì mã dòng được gọi là tuần hoàn chu kỳ *d*.

Trong thực tế, mã dòng thường được dùng với các văn bản nhị phân, tức là:

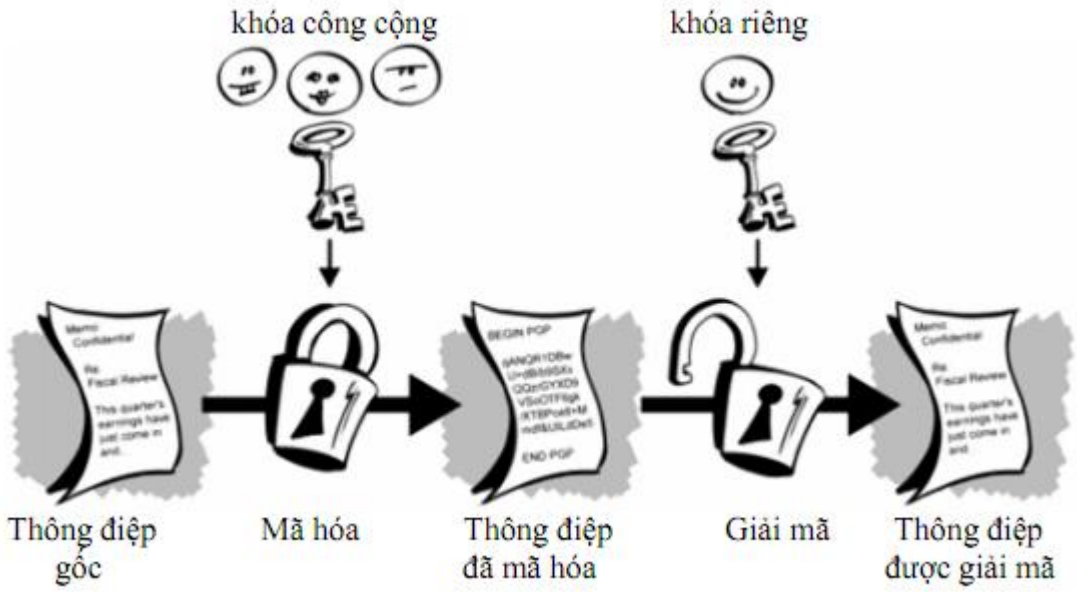
$\mathbf{P} = \mathbf{C} = Z_2 = \{0, 1\}$, các phép lập mã và giải mã là:

$e_z(x) = x + z \bmod 2$

$d_z(y) = y + z \bmod 2$

2.5.2 Phương pháp mã hóa công khai

Khái niệm: Mã hoá bằng khoá công khai (MHKCK - public-key) dùng để gửi dữ liệu một cách an toàn qua các mạng không an toàn như Internet. Cách mã hoá này làm cho những cặp mắt tò mò không thể đọc được dữ liệu. Mỗi người dùng đều có hai khoá, một khoá công khai, một khoá riêng. Khoá công khai được giữ trong một thư mục. Ai cũng có thể truy cập khoá này để mã hoá một thông điệp trước khi gửi tới người có khoá riêng tương ứng. Còn khoá riêng thì chỉ người nhận mới có thể truy cập được và dùng nó để giải mã thông điệp.



Hình 2.3: Mô hình hệ thống mã hóa công khai

Ví dụ:

+) Hệ mật mã công khai RSA được đưa ra năm 1977 là công trình nghiên cứu của ba đồng tác giả Ronald Linn Revest, Adi Shamir, Leonard Aldeman. Hệ mật mã được xây dựng dựa trên tính khó giải của bài toán phân tích một số thành thừa số nguyên tố hay còn gọi là bài toán RSA.

+)Hệ mật mã công khai ElGamal được đưa ra năm 1978. Hệ mật mã này được xây dựng dựa trên bài toán khó giải của bài toán logarit rời rạc

+) Hệ mật mã công khai Merkle-Helman (xếp ba lô) được xây dựng trên cơ sở của bài toán tổng hợp con.

Mã hóa công khai dựa trên nguyên tắc hoạt động của 2 loại hàm:

+) Hàm một phía

Hàm f(x) được gọi là *hàm một phía* nếu tính *"xuôi"* y = f(x) thì *"dễ"*, nhưng tính *"ngược"* $x = f^{-1}(y)$ lại rất *"khó"*.

Ví dụ: Hàm $f(x) = g^x \pmod p$, với p là số nguyên tố lớn, (g là phần tử nguyên thủy mod p) là hàm một phía.

+) Hàm một phía cửa sập

Hàm f(x) được gọi là *hàm của sập một phía* nếu tính y = f(x) thì *"dễ"*, tính $x = f^{-1}(y)$ lại rất *"khó"* . Tuy nhiên có cửa sổ sập z để tính $x = f^{-1}(y)$ là *"dễ"*.

Ví dụ: Hàm $f(x) = x^a \pmod n$ (với n là tích của hai số nguyên tố lớn n = p*q) là hàm một phía. Nếu chỉ biết a và n thì tính $x = f^{-1}(y)$ rất *"khó"* , nhưng nếu biết *cửa sập* p và q,, thì tính được $f^{-1}(y)$ là khá *"dễ"*.

Ưu điểm:

Khi áp dụng hệ thống mã hóa công cộng, người A sẽ sử dụng mã hóa công cộng để mã hóa thông điệp gửi cho người B. Nếu người C phát hiện thông điệp mà A gửi cho B, kết hợp thông tin về mã khóa công cộng đã được công bố, cũng rất khó có khả năng giải mã được thông điệp này do không nắm được mã khóa riêng của B.

Các phương pháp mã hóa công cộng giúp cho việc trao đổi mã khóa trở nên dễ dàng hơn. Nội dung của khóa công cộng không cần phải giữ bí mật trong các phương pháp mã hóa quy ước.

Người mã hóa dùng khóa công khai, người giải mã dùng khóa bí mật. Khả năng lộ khóa bí mật khó hơn vì chỉ có một người giữ. Nếu thám mã biết khóa công khai, cố gắng tím khóa bí mật thì sẽ phải đương đầu với bài toán "khó".

Thuật toán được viết một lần, công khai cho nhiều lần dùng, cho nhiều người dùng, họ chỉ cần giữ bí mật cho khóa riêng của mình.

Nhược điểm:

Tốc độ xử lý chậm hơn mã hóa đối xứng

Để có mức an toàn tương đương với một phương pháp mã hóa quy ước, một phương pháp mã hóa khóa công cộng phải sử dụng mã khóa có độ dài lớn hơn nhiều lần mã khóa bí mật được sử dụng trong mã hóa quy ước.

Nơi sử dụng hệ mã hóa khóa công khai.

Hệ mã hóa khóa công khai thường được sử dụng chủ yếu trên các mạng công khai như Internet, khi mà việc trao đổi chuyển khóa bí mật tương đối khó khăn.

Đặc trưng nổi bật của hệ mã hóa công khai là khóa công khai (public key) và bản mã (ciphertext) đều có thể gửi đi trên một kênh truyền tin ***không an toàn.***

Có biết cả khóa công khai và bản mã, thám mã cũng không dễ khám phá được bản rõ.

Nhưng vì có tốc độ mã hóa và giải mã ***chậm,*** nên hệ mã hóa khóa công khai chỉ dùng để mã hóa những bản tin ngắn, ví dụ như mã hóa khóa bí mật gửi đi.

Hệ mã hóa khóa công khai thường được sử dụng cho cặp người dùng thỏa thuận khóa bí mật của hệ mã hóa khóa riêng.

2.6 Chữ ký điện tử

2.6.1 Giới thiệu

Chữ ký điện tử (chữ ký số) không được sử dụng nhằm bảo mật thông tin mà nhằm bảo vệ thông tin không bị người khác cố tình thay đổi để tạo ra thông tin sai lệch. Nói cách khác, chữ ký điện tử giúp xác định được người đã tạo ra hay chịu trách nhiệm đối với một thông điệp.

Như vậy *"ký số"* trên *"tài liệu số"* là *"ký"* trên từng bít tài liệu. Kẻ gian khó thể giả mạo "chữ ký số" nếu nó không biết "khóa lập mã".

Để kiểm tra một *"chữ ký số"* thuộc về một *"tài liệu số"*, người ta giải mã *"chữ ký số"* bằng "khóa giải mã", và so sánh với tài liệu gốc

Một phương pháp chữ ký điện tử bao gồm hai thành phần chính: thuật toán dùng để tạo ra chữ ký điện tử và thuật toán tương ứng để xác nhận chữ ký điện tử. Áp dụng cho các thông điệp có độ dài cố định và thường tương đối ngắn.

2.6.2 Định nghĩa

Một phương pháp chữ ký điện tử được định nghĩa là một bộ năm (P, A, K, S, V) thỏa mãn các điều kiện sau:

1. P là tập hợp hữu hạn các thông điệp.
2. A là tập hợp hữu hạn các chữ ký có thể được sử dụng.
3. K là tập hợp hữu hạn các khóa có thể sử dụng.
4. S là tập các thuật toán ký.
5. V là tập các thuật toán kiểm thử.

Với mỗi khóa k ∈ K, tồn tại thuật toán chữ kí $sig_k \in S$ và thuật toán xác nhận chữ ký tương ứng $ver_k \in V$. Mỗi thuật toán $sig_k : P \to A$ và $ver_k : P \times A \to \{true, false\}$ là các hàm thỏa điều kiện:

$$\forall x \in P, \forall y \in A : ver(x, y) = \begin{cases} true & \text{nếu } y = sig(x) \\ false & \text{nếu } y \neq sig(x) \end{cases} \quad (2.1)$$

Ví dụ:

+) Phương pháp chữ ký điện tử RSA được xây dựng dựa theo phương pháp mã hóa công cộng RSA.

+) Phương pháp chữ ký điện tử ElGamal: được giới thiệu vào năm 1985, sau đó được sửa đổi bổ sung thành chuẩn chữ ký điện tử (Digital Signature Standard - DSS). Phương pháp này được xây dựng nhằm giải quyết bài toán chữ ký điện tử. Sử dụng chữ ký 320 bit trên thông điệp 160 bit.

2.6.3 Phân loại chữ ký số

2.6.3.1 Phân loại chữ ký theo đặc trưng kiểm tra chữ ký

+). Chữ ký khôi phục thông điệp:

Là loại chữ ký, trong đó người gửi chỉ cần gửi ***"chữ ký"*** , người nhận có thể khôi phục lại được thông điệp, đã được ***"ký"*** bởi ***"chữ ký"*** này.

+). Chữ ký đi kèm thông điệp:

Là loại chữ ký, trong đó người gửi chỉ cần gửi ***"chữ ký"***, phải gửi kèm cả thông điệp đã được ***"ký"*** bởi ***"chữ ký"*** này. Ngược lại, sẽ không có được thông điệp gốc.

Ví dụ:Chữ ký Elgamal là chữ ký đi kèm thông điệp, sẽ trình bày trong mục sau.

2.6.3.2 Phân loại chữ ký theo mức an toàn

+). Chữ ký "không thể phủ nhận":

Nhằm tránh việc nhân bản chữ ký để sử dụng nhiều lần, tốt nhất là người gửi tham gia trực tiếp vào việc kiểm thử chữ ký. Điều đó được thực hiện bằng một giao thức kiểm thử, dưới dạng một giao thức mời hỏi và trả lời.

Ví dụ: Chữ ký không phủ định (Chaum- van Antverpen), trình bày trong mục sau.

+). Chữ ký "một lần":

Để bảo đảm an toàn, "Khóa ký" chỉ dùng 1 lần (one - time) trên 1 tài liệu.

Ví dụ: Chữ ký một lần Lamport. Chữ ký Fail - Stop (Van Heyst & Pedersen).

2.6.3.3 Phân loại chữ ký theo ứng dụng đặc trưng

Chữ ký "mù" (Blind Signature).

Chữ ký "nhóm" (Group Signature).

Chữ ký "bội" (Multy Signature).

Chữ ký "mù nhóm" (Blind Group Signature).

Chữ ký "mù bội" (Blind Multy Signature).

2.7 Hàm băm mật mã

2.7.1 Giới thiệu về hàm băm

Hàm băm mật mã là hàm toán học chuyển đổi một thông điệp có độ dài bất kỳ thành một dãy bit có độ dài cố định (tùy thuộc vào thuật toán băm). Dãy bit này được gọi là thông điệp rút gọn (message digest) hay giá trị băm (hash value), đại diện cho thông điệp ban đầu.

Hàm băm h không phải là một song ánh. Do đó với thông điệp x bất kỳ, tồn tại thông điệp $x' \neq x$ sao cho $h(x) = h(x')$. Lúc này, ta nói rằng "có sự đụng độ xảy

ra". Hàm băm h được gọi là an toàn (hay "ít bị đụng độ") khi không thể xác định được (bằng cách tính toán) cặp thông điệp x và x' thỏa mãn $x \neq x'$ và $h(x) = h(x')$.

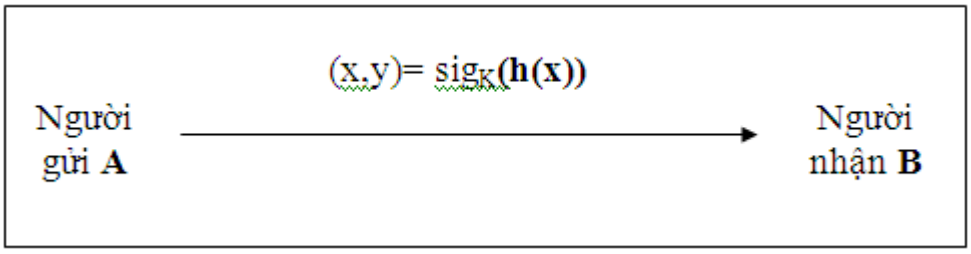
Trên thực tế, các thuật toán băm là hàm một chiều, do đó rất khó để xây dựng lại thông điệp ban đầu từ thông điệp rút gọn.

Hàm băm giúp xác định được tính toàn vẹn dữ liệu của thông tin: mọi thay đổi, dù là rất nhỏ, trên thông điệp cho trước , ví dụ như đổi giả trị 1 bit, đều làm thay đổi thông điệp rút gọn tương ứng. Tính chất này hữu ích trong việc phát sinh, kiểm tra chữ ký điện tử, các đoạn mã chứng nhận thông điệp, phát sinh số ngẫu nhiên, tạo ra khóa cho quá trình mã hóa...

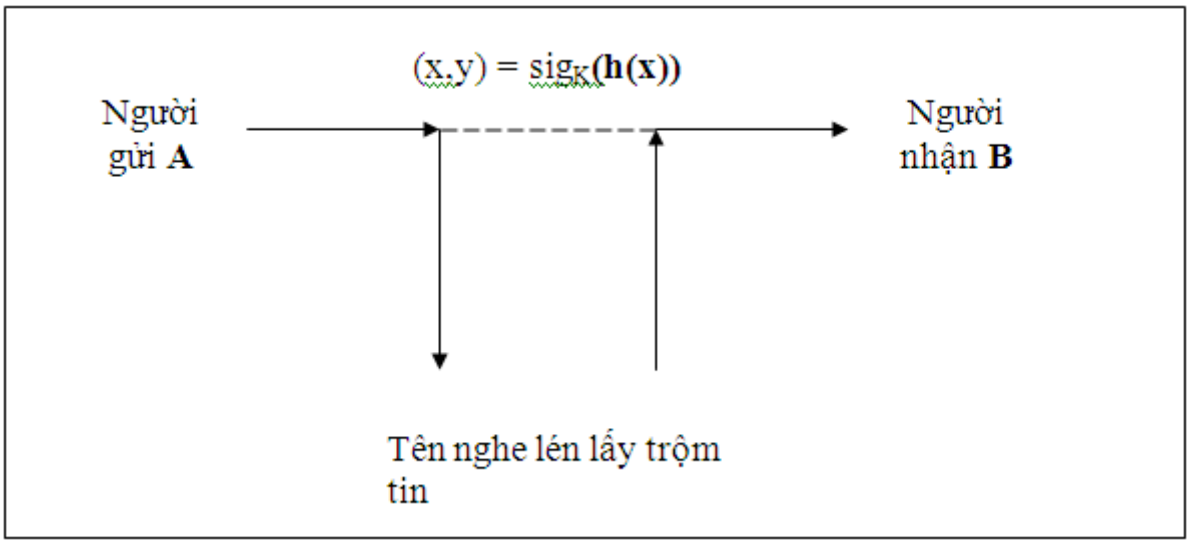
2.7.2 Tính chất hàm băm

+) Tính chất 1:* Hàm băm *h* là *không va chạm yếu.

Ví dụ: Xét kiểu tấn công như sau: *Kiểu tấn công theo tính chất 1*.



Hình 2.4: Cách đi đúng của thông tin : thông tin được truyền đúng từ A đến B



Hình 2.5: Thông tin bị lấy trộm và bị thay đổi trên đường truyền

**** Kiểu tấn công theo tính chất 1:***

+ Người A gửi cho người B bản tin **(x, y)** với **y**= sig_K (**h(x)**). B không nhận được **(x, y)** vì: trên đường truyền, tin bị lấy trộm. Tên trộm, bằng cách nào đó tìm được một bản tin **x'≠ x** nhưng lại có **h(x') = h(x).** Hắn thay thế **x** bằng **x'**, và chuyển tiếp **(x', y)** cho B.

+ Người B nhận được **(x', y)** và vẫn xác thực được thông tin đúng đắn. Do đó, để tránh kiểu tấn công như trên, hàm **h** phải thỏa mãn tính chất: **không va chạm yếu.**

**** Khái niệm:*** Hàm băm không va chạm yếu**.**

Hàm băm **h** được gọi là không va chạm yếu, nếu cho trước bức điện **x**, "khó" thể tính toán để tìm ra bức điện **x'≠ x** mà **h(x') = h(x).**

+) Tính chất 2:* Hàm băm *h* là *không va chạm mạnh.

Ví dụ: Xét kiểu tấn công như sau: *Kiểu tấn công theo tính chất 2.*

+ Đầu tiên, tên giả mạo tìm được hai thông điệp khác nhau **x'** và **x (x' ≠ x)** mà có **h(x) = h(x').** (Coi bức thông điệp **x** là hợp lệ, **x'** là giả mạo).

+ Tiếp theo, hắn thuyết phục ông A ký vào bản tóm lược **h(x)** để nhận được **y**.

Khi đó **(x', y)** là bức điện giả mạo nhưng hợp lệ vì **h(x) = h(x')**.

Để tránh tấn công kiểu này, hàm **h** phải thỏa mãn tính chất: **không va chạm mạnh.**

**** Khái niệm:*** Hàm băm không va chạm mạnh.

Hàm băm b được gọi là **không va chạm mạnh** nếu "khó" thể tính toán để tìm ra hai bức thông điệp khác nhau **x'** và **x (x' ≠ x)** mà có **h(x') = h(x)**.

+) Tính chất 3:* Hàm băm *h* là *hàm một chiều.

Ví dụ: Xét kiểu tấn công như sau: *Kiểu tấn công theo tính chất 3.*

+ Người A gửi cho B thông tin **(x, z, y)** với **z = h(x), y = sig_K(z)**

+ Giả sử tên giả mạo tìm được bản tin **x'**, được tính ngược từ bản tóm lược **z = h(x).**

+ Tên trộm thay thế bản tin **x** hợp lệ bằng bản tin **x'** giả mạo nhưng lại có **z = h(x').** Hắn ta ký số trên bản tóm lược **z** của **x'** bằng đúng chữ ký hợp lệ. Nếu làm như vậy thì **(x', z, y)** là bức điện giả mạo nhưng hợp lệ.

Để tránh được kiểu tấn công này, hàm băm h cần thỏa mãn tính chất một chiều.

*** Khái niệm**: Hàm băm một chiều.

Hàm băm **h** được gọi là **hàm một chiều** nếu khi cho trước một bản tóm lược thông báo **z** thì "khó thể" tính toán để tìm ra thông điệp ban đầu **x** sao cho **h(x) = z**.

2.7.3 Cấu trúc của hàm băm

Cho trước một thông điệp M có độ dài bất kỳ. Tùy theo thuật toán được sử dụng, chúng ta có thể cần bổ sung một số bit vào thông điệp này để nhận được thông điệp có độ dài là bội số của một hằng số cho trước. Chia nhỏ thông điệp thành từng khối có kích thước bằng nhau: $M_1, M_2,..., M_s$.

Gọi H là trạng thái có kích thước n bit, f là "hàm nén" thực hiện thao tác trộn khối dữ liệu với trạng thái hiện hành

+) Khởi gán H_0 bằng một vector khởi tạo nào đó

+) $H_i = f(H_{i-1}, M_i)$ với i = 1,2,3,...,s

H_s chính là thông điệp rút gọn của thông điệp M ban đầu

2.7.4 Một số phương pháp băm

2.7.4.1 Hàm băm MD4

- Hàm băm MD4 được giáo sư Ron Rivest đề nghị vào năm 1990. Vào năm 1992, phiên bản cải tiến MD5 của thuật toán này ra đời.

- Thông điệp rút gọn có độ dài 128 bit.

- Năm 2004, nhóm tác giả Xiaoyu Wang, Dengguo Feng, Xuejia Lai và Hongbo Yu đã công bố kết quả về việc phá vỡ thuật toán MD4 và MD5 bằng phương pháp tấn công đụng độ .

- INPUT: Thông điệp có độ dài tùy ý

- OUTPUT: Bản băm, đại diện cho thông điệp gốc, độ dài cố định 128 bit.

Mô tả thuật toán: Giả sử đầu vào là một xâu **a** có độ dài **b** bit (**b** có thể bằng 0)

Bước 1: Khởi tạo các thanh ghi

Có 4 thanh ghi được sử dụng để tính toán nhằm đưa ra các đoạn mã: A, B, C, D. Bản tóm lược của thông điệp được xây dựng như sự kết nối của các thanh ghi. Mỗi thanh ghi có độ dài 32 bit. Các thanh ghi này được khởi tạo giá trị hecxa.

word A := **67 45 23 01**

word B := **ef cd ab 89**

word C := **98 ba dc fe**

word D := **10 32 54 76**

Bước 2:

Xử lý thông điệp **a** trong 16 khối *word*, có nghĩa là xử lý cùng một lúc 16 *word* = 512 bit (chia mảng M thành các khối 512 bit, đưa từng khối 512 bit đó vào mảng T[j]). Mỗi lần xử lý một khối 512 bit. Lặp lại N/16 lần.

2.7.4.2 Hàm băm MD5

MD5 được công bố năm 1991. MD5 dùng bốn vòng thay cho ba và châm hơn 30% so với MD4 (khoảng 0.9 MB/giây trên cùng một máy).

INPUT: Thông điệp (văn bản có độ dài tùy ý).

OUTPUT: Bản băm, đại diện cho thông điệp gốc, độ dài cố định 128 bit

Mô tả thuật toán:

Các bước 1, 2 của MD5 tương tự như của MD4.

Bước 3: Thực hiện bốn vòng băm

Đầu tiên, bốn biến A, B, C, D được khởi tạo. Những biến này được gọi là chaining variables. Bốn chu kỳ biến đổi trong MD5 hoàn toàn khác nhau và lần lượt sử dụng các hàm F, G, H và I. Mỗi tham số X, Y, Z là các từ 32 bit và kết quả là một từ 32 bit.

$F(X, Y, Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$

$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge (\neg Z))$

$H(X, Y, Z) = X \oplus Y \oplus Z$

$I(X, Y, Z) = Y \oplus (X \vee (\neg Z))$

với quy ước:

$X \wedge Y$	Phép toán AND trên bit giữa X và Y
$X \vee Y$	Phép toán OR trên bit giữa X và Y
$X \oplus Y$	Phép toán XOR trên bit giữa X và Y
$\neg X$	Phép toán NOT trên bit của X
$X + Y$	Phép cộng (modulo 2^{32})
$X <<< s$	Các bit của X được dịch chuyển xoay vòng sang trái s vị trí ($0 \leq s < 32$)

Thông điệp ban đầu x sẽ được mở rộng thành dãy bit X có độ dài là bội số của 512. Một bit 1 được thêm vào sau dãy bit x, tiếp đến là dãy gồm d bit 0 và cuối cùng là dãy 64 bit *l* biểu diễn độ dài thông điệp x. Dãy gồm d bit 0 được thêm vào sao cho dãy X có độ dài là bội số của 512. Quy trình này được thể hiện trong thuật toán xây dựng dãy bit X từ dãy bit x:

$d = (447 - |x|) \bmod 512$

Gọi dãy 64 bit l là biểu diễn nhị phân của giá trị $|x| \bmod 2^{64}$.

$X = x \,||\, 1 \,||\, 0^d \,||\, l$

Đơn vị xử lý trong MD5 là các từ 32-bit nên dãy X sẽ được biểu diễn thành dãy các từ X[i] 32 bit: X = X[0] X[1]...X[N-1] với N là bội số của 16.

Định nghĩa các hàm:

```
FF(a,b,c,d,Mj,s,ti):
        a = b + ((a + F(b,c,d) + Mj + ti) <<< s)
GG(a,b,c,d,Mj,s,ti):
        a = b + ((a + G(b,c,d) + Mj + ti) <<< s)
HH(a,b,c,d,Mj,s,ti):
        a = b + ((a + H(b,c,d) + Mj + ti) <<< s)
II(a,b,c,d,Mj,s,ti):
        a = b + ((a + I(b,c,d) + Mj + ti) <<< s)
```

với M_j là M[j] và hằng số ti xác định theo công thức:

$ti = \lfloor 2^{32} |\sin(i)| \rfloor$, i tính bằng radian.

Ưu điểm:

- Thêm chu kỳ thứ 4 để tăng mức độ an toàn.

- Mỗi bước trong từng chu kỳ chịu ảnh hưởng kết quả bước biến đổi trước đó nhằm tăng nhanh tốc độ của hiệu ứng lan truyền.

- Các hệ số dịch chuyển xoay vòng trong mỗi chu kỳ được tối ưu hóa nhằm tăng tốc độ hiệu ứng lan truyền. Ngoài ra, mỗi chu kỳ sử dụng bốn hệ số dịch chuyển khác nhau.

- Hàm G ở chu kỳ 2 của MD4 : G(X, Y, Z) = ((X $\wedge$ Y)$\vee$(X $\wedge$ Z)$\vee$(Y $\wedge$ Z)) được thay thế bằng ((X $\wedge$ Z) $\vee$ (Y $\wedge$ Z)) nhằm giảm tính đối xứng.

2.7.4.3 Hàm băm Chuẩn SHA

Chuẩn hàm băm SHA phức tạp và chậm hơn dòng MD. SHA được thiết kế để chạy trên máy kiến trúc endian lớn hơn là trên máy endian nhỏ. SHA tạo ra bản tóm lược thông điệp có kích thước 160 bit, sử dụng 5 thanh ghi 32 bit.

INPUT : Thông điệp (văn bản) có độ dài tùy ý

OUTPUT: Bản băm, đại diện cho thông điệp gốc, độ dài cố định 160 bit.

Các thuật toán hàm băm SHA gồm 2 bước: tiền xử lý và tính toán giá trị băm.

+ Bước tiền xử lý bao gồm các thao tác:

- Mở rộng thông điệp

- Phân tích thông điệp đã mở rộng thành các khối *m* bit

- Khởi tạo giá trị băm ban đầu

+ Bước tính toán giá trị băm bao gồm các thao tác:

- Làm *N* lần các công việc sau:

+) Tạo bảng phân bố thông điệp (message schedule) từ khối thứ *i*.

+) Dùng bảng phân bố thông điệp cùng với các hàm, hằng số, các thao tác trên từ để tạo ra giá trị băm *i*.

- Sử dụng giá trị băm cuối cùng để tạo thông điệp rút gọn.

Thông điệp *M* được mở rộng trước khi thực hiện băm,nhằm đảm bảo thông điệp mở rộng có độ dài là bội số của 512 hoặc 1024 bit tùy thuộc vào thuật toán. Sau khi thông điệp đã mở rộng, thông điệp cần được phân tích thành *N* khối *m*-bit trước khi thực hiện băm.

Đối với SHA-1 và SHA-256, thông điệp mở rộng được phân tích thành N khối 512-bit $M^{(1)}$, $M^{(2)}$,..., $M^{(N)}$. Do đó 512 bit của khối dữ liệu đầu vào có thể được thể hiện bằng 16 từ 32-bit, $M_0(i)$ chứa 32 bit đầu của khối thông điệp i, $M_1^{(i)}$ chứa 32 bit kế tiếp,…

Đối với SHA-384 và SHA-512, thông điệp mở rộng được phân tích thành N khối 1024-bit $M^{(1)}$, $M^{(2)}$,..., $M^{(N)}$. Do đó 1024 bit của khối dữ liệu đầu vào có thể được thể hiện bằng 16 từ 64-bit, $M_0^{(i)}$ chứa 64 bit đầu của khối thông điệp i, $M_1^{(i)}$ chứa 64 bit kế tiếp,...

Với mỗi thuật toán băm an toàn, giá trị băm ban đầu $H^{(0)}$ phải được thiết lập. Kích thước và số lượng từ trong $H^{(0)}$ tùy thuộc vào kích thước thông điệp rút gọn.

Các cặp thuật toán SHA-224 và SHA-256; SHA-384 và SHA-512 có các thao tác thực hiện giống nhau, chỉ khác nhau về số lượng bit kết quả của thông điệp rút gọn. Nói cách khác, SHA-224 sử dụng 224 bit đầu tiên trong kết quả thông điệp rút gọn sau khi áp dụng thuật toán SHA256. Tương tự SHA-384 sử dụng 384 bit đầu tiên trong kết quả thông điệp rút gọn sau khi áp dụng thuật toán SHA-512.

Trong hàm băm SHA, chúng ta cần sử dụng thao tác quay phải một từ, ký hiệu là ROTR, và thao tác dịch phải một từ, ký hiệu là SHR.

Mỗi thuật toán có bảng hằng số phân bố thông điệp tương ứng. Kích thước bảng hằng số thông điệp (scheduleRound) của SHA-224 và SHA-256 là 64, kích thước bảng hằng số thông điệp của SHA-384 và SHA-512 là 80.

Trong hàm băm SHA-224 và SHA-256, chúng ta cần sử dụng các hàm:

$$\mathbf{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$
$$\mathbf{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$
$$\sum_0(x) = \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x)$$
$$\sum_1(x) = \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x)$$
$$\sigma_0(x) = \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x)$$
$$\sigma_1(x) = \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x)$$

Trong hàm băm SHA-384 và SHA-512, chúng ta cần sử dụng các hàm sau:

$$\mathrm{Ch}(x,y,z)=(x\wedge y)\oplus(\neg x\wedge z)$$
$$\mathrm{Maj}(x,y,z)=(x\wedge y)\oplus(x\wedge z)\oplus(y\wedge z)$$
$$\sum_0(x)=\mathrm{ROTR}^{28}(x)\oplus\mathrm{ROTR}^{34}(x)\oplus\mathrm{ROTR}^{29}(x)$$
$$\sum_1(x)=\mathrm{ROTR}^{14}(x)\oplus\mathrm{ROTR}^{18}(x)\oplus\mathrm{ROTR}^{41}(x)$$
$$\sigma_0(x)=\mathrm{ROTR}^{1}(x)\oplus\mathrm{ROTR}^{8}(x)\oplus\mathrm{SHR}^{7}(x)$$
$$\sigma_1(x)=\mathrm{ROTR}^{19}(x)\oplus\mathrm{ROTR}^{61}(x)\oplus\mathrm{SHR}^{6}(x)$$

Nhận xét

Chuẩn SHS đặc tả 5 thuật toán băm an toàn SHA-1, SHA-224[3], SHA-256, SHA-84 và SHA-512. Bảng 2.1 thể hiện các tính chất cơ bản của bốn thuật toán băm an toàn.

Sự khác biệt chính của các thuật toán là số lượng bit bảo mật của dữ liệu được băm – điều này có ảnh hưởng trực tiếp đến chiều dài của thông điệp rút gọn. Khi một thuật toán băm được sử dụng kết hợp với thuật toán khác đòi hỏi phải cho kết quả số lượng bit tương ứng. Ví dụ, nếu một thông điệp được ký với thuật toán chữ ký điện tử cung cấp 128 bit thì thuật toán chữ ký đó có thể đòi hỏi sử dụng một thuật toán băm an toàn cung cấp 128 bit như SHA-256.

Ngoài ra, các thuật toán khác nhau về kích thước khối và kích thước từ được sử dụng.

Bảng 2.1: *Các tính chất của các thuật toán băm an toàn*

Thuật toán	**Kích thước (bit)**				**Độ an toàn[4] (đơn vị: bit)**
	Thông điệp	**Khối**	**Từ**	**Thông điệp rút gọn**	
SHA-1	$<2^{64}$	512	32	160	80
SHA-224	$<2^{64}$	512	32	224	112
SHA-256	$<2^{64}$	512	32	256	128
SHA-384	$<2^{128}$	1024	64	384	192
SHA-512	$<2^{128}$	1024	64	512	256

CHƯƠNG 3: THUẬT TOÁN MÃ HÓA RIJNDAEL VÀ ỨNG DỤNG

3.1 Giới thiệu

Phương pháp Rijndael do Vincent Rijmen và Joan Daeman đưa ra. Thuật toán được đặt tên là "Rijndael" khi tham gia cuộc thi thiết kế AES(Tiêu chuẩn mã hóa tiên tiến). Được Viện Tiêu chuẩn và Công nghệ Hoa Kỳ (National Institute of Standards and Technology – NIST) chọn làm chuẩn mã hóa nâng cao (Advanced Encryption Standard) từ 02 tháng 10 năm 2000.

Là phương pháp mã hóa theo khối (block cipher) có kích thước khối và mã khóa thay đổi linh hoạt với các giá trị 128, 192 hay 256 bit. Phương pháp này thích hợp ứng dụng trên nhiều hệ thống khác nhau từ các thẻ thông minh cho đến các máy tính cá nhân.

3.2 Tham số, ký hiệu, thuật ngữ và hàm

- AddroundKey: Phép biến đổi sử dụng trong mã hóa và giải mã, thực hiện việc cộng mã khóa của chu kỳ vào trạng thái hiện hành. Độ dài của mã khóa của chu kỳ bằng với kích thước của trạng thái.

- SubBytes: Phép biến đổi sử dụng trong mã hóa, thực hành việc thay thế phi tuyến từng byte trong trạng thái hiện hành thông qua bảng thay thế (S-box).

- InvSubBytes: Phép biến đổi sử dụng trong giải mã. Đây là phép biến đổi ngược của phép biến đổi SubBytes.

- Mixcolumns: Phép biến đổi sử dụng trong mã hóa, thực hiện thao tác trộn thông tin của từng cột trong trạng thái hiện hành. Mỗi cột được xử lý độc lập.

- InvMixcolumns: Phép biến đổi sử dụng giải mã. Đây là phép biến đổi ngược của phép biến đổi Mixcolumns.

- ShiftRows: Phép biến đổi sử dụng trong mã hóa, thực hiện việc dịch chuyển xoay vòng của trạng thái hiện hành với di số tương ứng khác nhau.

- InvShiftRows: Phép biến đổi sử dụng trong giải mã. Đây là phép biến đổi ngược của phép biến đổi ShiftRows.

- Nw: Số lượng byte trong một đơn vị dữ liệu “từ”. Trong thuật toán Rijndael, thuật toán mở rộng 256/384/512 bit và thuật toán mở rộng 512/768/1024 bit, giá trị Nw lần lượt là 4, 8 và 16.

- K: Khóa chính.

- Nb: Số lượng cột (số lượng các từ $8\times Nw\ bit$) trong trạng thái. Giá trị Nb = 4, 6, hay 8. Chuẩn AES giới hạn lại giá trị của Nb =4.

- Nk: Số lượng các từ($8\times Nw\ bit$) trong khóa chính. Giá trị Nk = 4, 6, hay 8.

- Nr: Số lượng các chu kỳ, phụ thuộc vào giá trị Nk hay Nb theo công thức:

Nr = max(Nb, Nk)+6

- Rcon[]: hằng của chu kỳ.

-RotWord: Hàm được sử dụng trong quá trình mở rộng mã khóa, thực hiện thao tác dịch chuyển xoay vòng Nw byte thành phần của một từ.

- SubWord: Hàm được sử dụng trong quá trình mở rộng mã khóa. Nhận vào một từ(Nw byte), áp dụng phép thay thế dựa vào S-box đối với từng byte thành phần và trả về từ gồm Nw byte thành phần đã được thay thế.

- XOR: Phép toán Exclusive-OR.

- $\oplus$: Phép toán Exclusive-OR.

- $\otimes$: Phép nhân hai đa thức(mỗi đa thức có bậc < Nw) modulo cho đa thức $x^{Nw}+1$.

- • : Phép nhân trên trường hữu hạn.

3.3 Một số khái niệm toán học

Các khóa con sử dụng trong các chu trình được tạo ra bởi quá trình tạo khóa con Rijndael.

Đơn vị thông tin được xử lý trong thuật toán Rijndael là byte . Mỗi byte xem như một phần tử của trường Galois GF(2^8) được trang bị phép cộng (ký hiệu $\oplus$) và phép nhân (ký hiệu $\bullet$)

Mỗi byte được biểu diễn theo nhiều cách khác nhau:

Dạng nhị phân: $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$

Dạng thập lục phân: $\{h_1h_0\}$

Dạng đa thức có các hệ số nhị phân $\sum_{i=0}^{7} b_i x^i$

3.3.1 Phép cộng

Phép cộng hai phần tử trên GF(2^8) được thực hiện bằng cách "cộng" (thực chất là phép toán XOR, ký hiệu $\oplus$) các hệ số của các đơn thức đồng dạng của hai đa thức tương ứng với hai toán hạng đang xét. Như vậy, phép cộng và phép trừ hai phần tử bất kỳ trên GF(2^8) là hoàn toàn tương đương nhau.

Nếu biểu diễn lại các phần tử thuộc GF(2^8) dưới hình thức nhị phân thì phép cộng giữa $\{a_7a_6a_5a_4a_3a_2a_1a_0\} \oplus \{b_7b_6b_5b_4b_3b_2b_1b_0\} = \{c_7c_6c_5c_4c_3c_2c_1c_0\}$ với $c_i = a_i \oplus b_i$, $0 \le i \le 7$

3.3.2 Phép nhân trên GF(2^8)

Khi xét trong biểu diễn đa thức, phép nhân trên GF(28) (ký hiệu •) tương ứng với phép nhân thông thường của hai đa thức đem chia lấy dư (modulo) cho một đa thức tối giản (irreducible polynomial) bậc 8. Đa thức được gọi là tối giản khi và chỉ khi đa thức này chỉ chia hết cho 1 và chính mình. Trong thuật toán Rijndael, đa thức tối giản được chọn là:

$$m(x) = x^8 + x^4 + x^3 + x + 1 \qquad (3.1)$$

hay 1{1b} trong biểu diễn dạng thập lục phân.

Kết quả nhận được là một đa thức bậc nhỏ hơn 8 nên có thể được biểu diễn dưới dạng 1 byte. Phép nhân trên GF(2^8) không thể được biểu diễn bằng một phép toán đơn giản ở mức độ byte.

Phép nhân được định nghĩa trên đây có tính kết hợp, tính phân phối đối với phép cộng và có phần tử đơn vị là {01}.Với mọi đa thức $b(x)$ có hệ số nhị phân với bậc nhỏ hơn 8 tồn tại phần tử nghịch đảo của $b(x)$, ký hiệu $b^{-1}(x)$ (được thực hiện bằng cách sử dụng thuật toán Euclide mở rộng).

Nhận xét: Tập hợp 256 giá trị từ 0 đến 255 được trang bị phép toán cộng (được định nghĩa là phép toán XOR) và phép nhân định nghĩa như trên tạo thành trường hữu hạn GF(2^8).

3.3.2.1 Phép nhân với x

Phép nhân (thông thường) đa thức

$$b(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \qquad (3.2)$$

với đa thức x cho kết quả là đa thức:

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0 x \quad (3.3)$$

Kết quả x•b(x) được xác định bằng cách modulo kết quả này cho đa thức m(x).

1.Trường hợp $b_7 = 0$

$$x \bullet b(x) = b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0 x \quad (3.4)$$

2.Trường hợp $b_7 = 1$

$$x \bullet b(x) = (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0 x) \bmod m(x)$$

$$= (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0 x) - m(x) \quad (3.5)$$

Như vậy, phép nhân với đa thức *x* (hay phần tử {00000010} ∈ GF(2^8)) có thể được thực hiện ở mức độ byte bằng một phép shift trái và sau đó thực hiện tiếp phép toán XOR với giá trị {1b}nếu $b_7 = 1$ .Thao tác này được ký hiệu là xtime(). Phép nhân với các lũy thừa của *x* có thể được thực hiện bằng cách áp dụng nhiều lần thao tác xtime(). Kết quả của phép nhân với một giá trị bất kỳ được xác định bằng cách cộng (⊕) các kết quả trung gian này lại với nhau.

Khi đó, việc thực hiện phép nhân giữa hai phần tử *a*, *b* bất kỳ thuộc GF(2^8) có thể được tiến hành theo các bước sau:

1. Phân tích một phần tử (giả sử là *a*) ra thành tổng của các lũy thừa của 2.

2. Tính tổng các kết quả trung gian của phép nhân giữa phần tử còn lại (là *b*) với các thành phần là lũy thừa của 2 được phân tích từ *a*.

Ví dụ:

{57} • {13} = {fe} vì

{57} • {02} = xtime({57}) = {ae}

{57} • {04} = xtime({ae}) = {47}

{57} • {08} = xtime({47}) = {8e}

{57} • {10} = xtime({8e}) = {07},

Như vậy:

{57} • {13} = {57} • ({01} ⊕ {02} ⊕ {10})

= {57} ⊕ {ae} ⊕ {07}

= {fe}

3.3.2.2 Đa thức với hệ số trên GF(2^8)

Xét đa thức $a(x)$ và $b(x)$ bậc 4 với các hệ số thuộc GF(28):

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \quad \text{và} \quad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0 \qquad (3.6)$$

Hai đa thức này có thể được biểu diễn lại dưới dạng từ gồm 4 byte [$a0$, $a1$, $a2$, $a3$] và [$b0$, $b1$, $b2$, $b3$]. Phép cộng đa thức được thực hiện bằng cách cộng (chính là phép toán XOR trên byte) các hệ số của các đơn thức đồng dạng với nhau.

Phép nhân giữa $a(x)$ với $b(x)$ được thực hiện thông qua hai bước. Trước tiên, thực

hiện phép nhân thông thường $c(x) = a(x)b(x)$.

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0 \qquad (3.7)$$

với

$$\begin{aligned} c_0 &= a_0 \bullet b_0 & c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 & c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 & c_6 &= a_3 \bullet b_3 \\ c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 . \end{aligned} \qquad (3.8)$$

Rõ ràng là $c(x)$ không thể được biểu diễn bằng một từ gồm 4 byte. Đa thức $c(x)$ có thể được đưa về một đa thức có bậc nhỏ hơn 4 bằng cách lấy $c(x)$ modulo cho một đa thức bậc 4. Trong thuật toán Rijndael, đa thức bậc 4 được chọn là

$M(x) = x^4 + 1$.

Do $x^j \bmod (x^4 + 1) = x^{j \bmod 4}$ nên kết quả $d(x) = a(x) \otimes b(x)$ được xác định bằng

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0 \qquad (3.9)$$

với

$$\begin{aligned} d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\ d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \end{aligned} \qquad (3.10)$$

Trong trường hợp đa thức $a(x)$ cố định, phép nhân $d(x) = a(x) \otimes b(x)$ có thể được biểu diễn dưới dạng ma trận như sau:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad (3.11)$$

Do $x^4 + 1$ không phải là một đa thức tối giản trên GF(2^8) nên phép nhân với một đa thức $a(x)$ cố định được chọn bất kỳ không đảm bảo tính khả nghịch. Vì vậy, trong phương pháp Rijndael đã chọn đa thức $a(x)$ có phần tử nghịch đảo (modulo $M(x)$)

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (3.12)$$

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \quad (3.13)$$

3.4 Phương pháp Rijndael

Phương pháp mã hóa Rijndael bao gồm nhiều bước biến đổi được thực hiện tuần tự, kết quả đầu ra của bước biến đổi trước là đầu vào của bước biến đổi tiếp theo.

Kết quả trung gian giữa các bước biến đổi được gọi là trạng thái (state).Một trạng thái được biểu diễn dưới dạng ma trận gồm 4 dòng và *Nb* cột với *Nb* bằng độ dài khối chia cho 32. Mã khóa chính (Cipher Key) được biểu diễn dưới dạng ma trận gồm 4 dòng và *Nk* cột với *Nk* bằng độ dài khóa chia cho 32.

Số lượng chu kỳ phụ thuộc vào giá trị của Nb và Nk theo công thức $Nr = \max\{Nb, Nk\}+6$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

Hình 3.1: Biểu diễn dạng ma trận của trạng thái (Nb=6) và mã khóa (Nk=4)

AES làm việc với từng khối dữ liệu 4×4 byte (tiếng Anh: state, khối trong Rijndael có thể có thêm cột).

3.4.1 Quá trình mã hóa bao gồm 4 bước:

1. AddRoundKey — mỗi byte của khối được kết hợp với khóa con, các khóa con này được tạo ra từ quá trình tạo khóa con Rijndael.
2. SubBytes — đây là phép thế (phi tuyến) trong đó mỗi byte sẽ được thế bằng một byte khác theo bảng tra (Rijndael S-box).
3. ShiftRows — đổi chỗ, các hàng trong khối được dịch vòng.
4. MixColumns — quá trình trộn làm việc theo các cột trong khối theo một phép biến đổi tuyến tính.

Tại chu trình cuối thì bước MixColumns được thay thế bằng bước AddRoundKey

Mỗi phép biến đổi thao tác trên trạng thái hiện hành *S*. Kết quả *S'* của mỗi phép biến đổi sẽ trở thành đầu vào của phép biến đổi kế tiếp trong quy trình mã hóa.

Trước tiên, toàn bộ dữ liệu đầu vào được chép vào mảng trạng thái hiện hành. Sau khi thực hiện thao tác cộng mã khóa đầu tiên, mảng trạng thái sẽ được trải qua *Nr* = 10, 12 hay 14 chu kỳ biến đổi (tùy thuộc vào độ dài của mã khóa chính cũng như độ dài của khối được xử lý). *Nr* −1 chu kỳ đầu tiên là các chu kỳ biến đổi bình thường và hoàn toàn tương tự nhau, riêng chu kỳ biến đổi cuối cùng có sự khác biệt so với *Nr* −1 chu kỳ trước đó. Cuối cùng, nội dung của mảng trạng thái sẽ được chép lại vào mảng chứa dữ liệu đầu ra.

Quy trình mã hóa Rijndael được tóm tắt lại như sau:

1. Thực hiện thao tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ mã hóa.
2. *Nr* – 1 chu kỳ mã hóa bình thường: mỗi chu kỳ bao gồm bốn bước biến đổi liên tiếp nhau: SubBytes, ShiftRows, MixColumns, và AddRoundKey.
3. Thực hiện chu kỳ mã hóa cuối cùng: trong chu kỳ này thao tác MixColumns được bỏ qua

Trong thuật toán dưới đây, mảng w[] chứa bảng mã khóa mở rộng; mảng in[] và out[] lần lượt chứa dữ liệu vào và kết quả ra của thuật toán mã hóa.

```
Cipher( byte in[4 * Nb],
        byte out[4 * Nb],
        word w[Nb * (Nr + 1)])
begin
   byte   state[4,Nb]
   state =   in
   AddRoundKey(state, w)                    // Xem phần 3.4.6
   for round = 1 to Nr - 1
      SubBytes(state)                       // Xem phần 3.4.2
      ShiftRows(state)                      // Xem phần 3.4.4
      MixColumns(state)                     // Xem phần 3.4.5
      AddRoundKey(state, w + round * Nb)
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w + Nr * Nb)
   out = state
end
```

3.4.1.1 Bước SubBytes

Các byte được thế thông qua bảng tra S-box (bảng 3.1). Đây chính là quá trình phi tuyến của thuật toán. Hộp S-box này được tạo ra từ một phép nghịch đảo trong trường hữu hạn GF (2^8) 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. Gồm 2 bước:

1. Xác định phần tử nghịch đảo $x^{-1} \in GF(2^8)$. Quy ước $\{00\}^{-1} = \{00\}$.

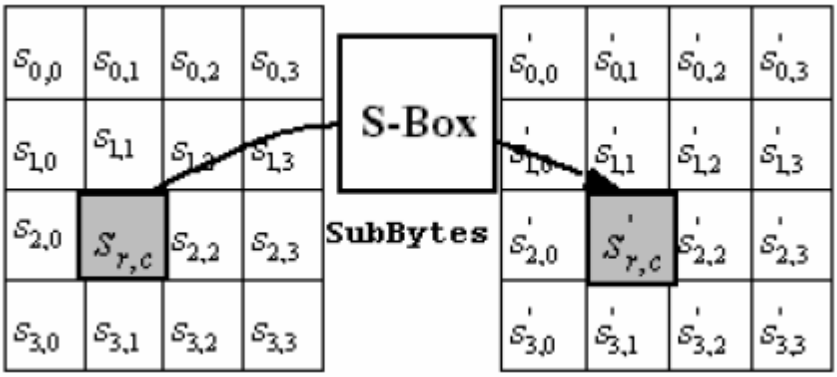
2. Áp dụng phép biến đổi affine (trên GF(2)) đối với x^{-1} (giả sử x^{-1} có biểu diễn nhị phân là $\{x_7\ x_6\ x_5x_4\ x_3x_2x_1x_0\}$):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \quad (3.14)$$

hay

$$y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8\ 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \quad (3.15)$$

với c_i là bit thứ i của {63}, $0 \leq i \leq 7$.



Hình 3.2: Thao tác SubBytes tác động trên từng byte của trạng thái.

Bảng 3.1: Bảng thay thế S-box cho giá trị {xy} ở dạng thập lục phân

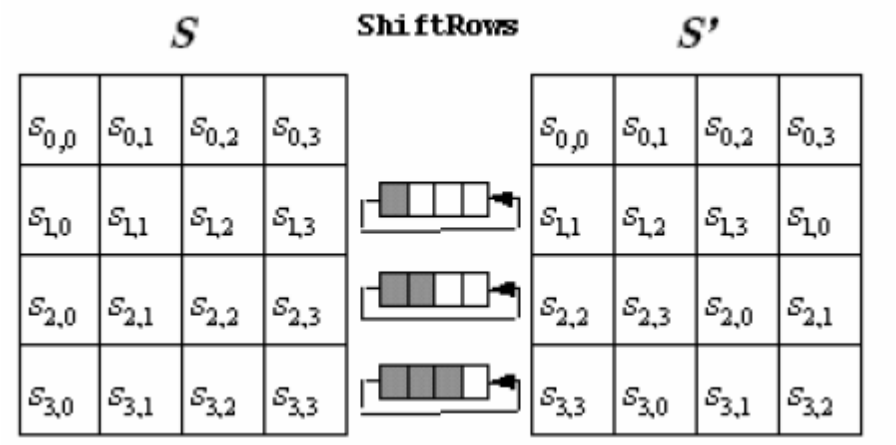
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	B5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	Df
	f	8c	a1	89	0d	Bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Đoạn mã giả cho phép biến đổi SubBytes

```
SubBytes(byte state[4,Nb])
begin
    for r = 0 to 3
        for c = 0 to Nb - 1
            state[r,c] = Sbox[state[r,c]]
        end for
    end for
end
```

3.4.1.2 Bước ShiftRows

Các hàng được dịch vòng một số vị trí nhất định. Đối với AES, hàng đầu được giữ nguyên. Mỗi byte của hàng thứ 2 được dịch trái một vị trí. Tương tự, các hàng thứ 3 và 4 được dịch 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.



Hình 3.3: Thao tác ShiftRows tác động trên từng dòng của trạng thái

Byte $S_{r,c}$ tại dòng r cột c sẽ dịch chuyển đến cột (c - *shift*(r, Nb)) mod Nb hay:

$$S'_{r,c} = S_{r.(c + shift(r,Nb) \bmod Nb} \text{ với } 0 < r < 8 \text{ và } 0 \le c < Nb \qquad (3.16)$$

Giá trị di số *shift*(r, Nb) phụ thuộc vào chỉ số dòng r và kích thước Nb của khối dữ liệu.

Bảng 3.2: Giá trị di số shift(r,Nb)

shift(r, Nb)		*r*		
		1	2	3
Nb	4	1	2	3
	6	1	2	3
	8	1	3	4

Mã giả cho phép biến đổi ShiftRows:

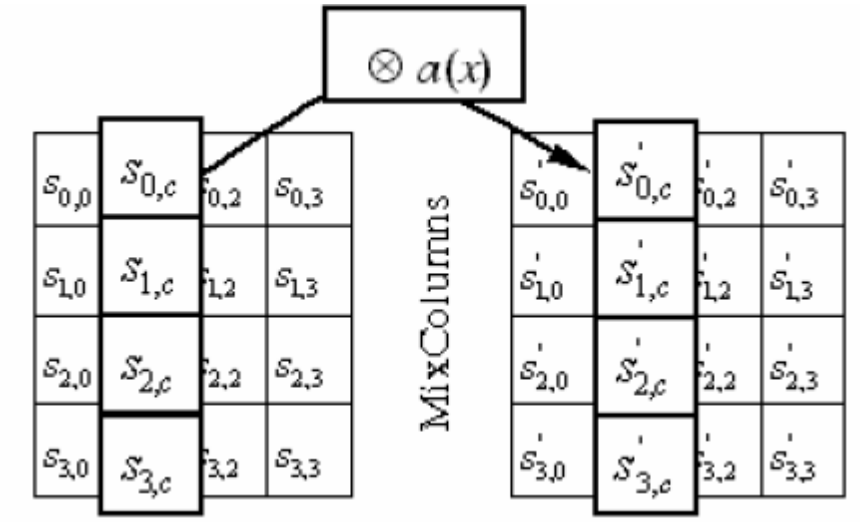
```
ShiftRows(byte state[4,Nb])
begin
    byte t[Nb]
    for r = 1 to 3
        for c = 0 to Nb - 1
            t[c] = state[r, (c + h[r,Nb]) mod Nb]
        end for
        for c = 0 to Nb - 1
            state[r,c] = t[c]
        end for
    end for
end
```

3.4.1.3 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 4 byte đầu ra. Cùng với bước ShiftRows, MixColumns đã tạo ra tính chất khuyếch tán cho thuật toán. Mỗi cột được xem như một đa thức trong trường hữu hạn và được nhân với đa thức c(x) = {03}x^3 +{01} x^2 +{01} x +{02} (modulo x^4 + 1). Vì thế, bước này có thể được xem là phép nhân ma trận trong trường hữu hạn. Hình 3.4 mô tả thao tác MixColums tác động lên cột của mỗi trạng thái.

Thao tác này được thể hiện ở dạng ma trận như sau:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (3.17)$$



Hình 3.4: Thao tác MixColumns tác động lên mỗi cột của trạng thái

3.4.1.4 Bước AddRoundKey

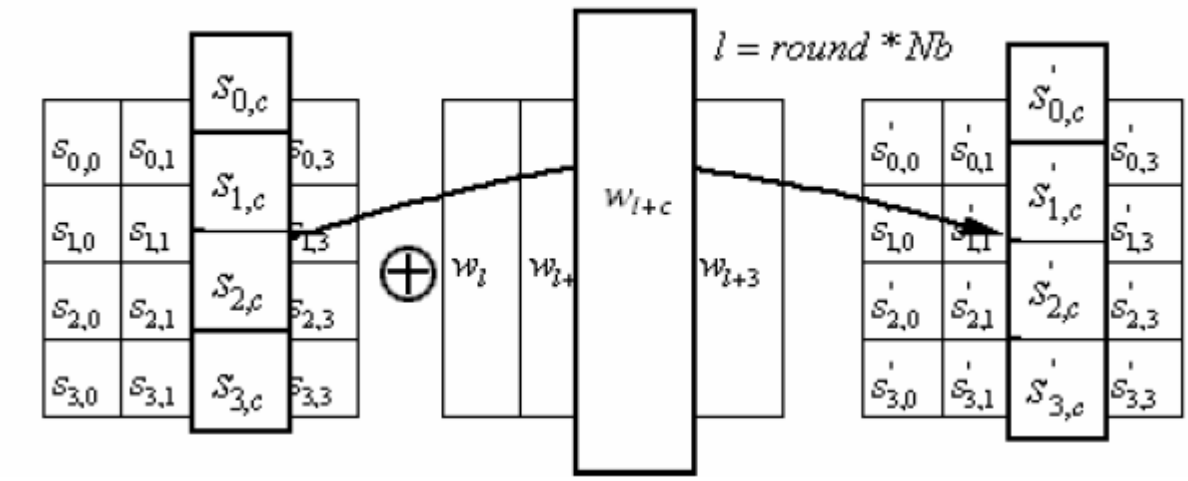
Tại bước này, khóa con được kết hợp với các khối. Khóa con trong mỗi chu trình được tạo ra từ khóa chính với quá trình tạo khóa con Rijndael; mỗi khóa con có độ dài giống như các khối. Quá trình kết hợp được thực hiện bằng cách **XOR** từng bít của khóa con với khối dữ liệu đang xét:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round*Nb+c}],$$

với $0 \le c < Nb$.

(3.18)

Thao tác biến đổi ngược của AddRoundKey cũng chính là thao tác AddRoundKey.



Hình 3.5: Thao tác AddRoundKey tác động lên mỗi cột của trạng thái.

Đoạn mã giả:

```
AddRoundKey(byte state[4,Nb], word rk[])
// rk = w + round * Nb
begin
    for c = 0 to Nb - 1
        for r = 0 to 3
            state[r,c] = state[r,c] xor xbyte(r, rk[c])
        end for
    end for
end
```

3.4.1.5 Phát sinh khóa của mỗi chu kỳ

Các khóa của mỗi chu kỳ (RoundKey) được phát sinh từ khóa chính. Quy trình phát sinh khóa cho mỗi chu kỳ gồm 2 giai đoạn::

1. Mở rộng khóa chính thành bảng khóa mở rộng,

2. Chọn khóa cho mỗi chu kỳ từ bảng khóa mở rộng.

a) Xây dựng bảng khóa mở rộng

Bảng khóa mở rộng là mảng 1 chiều chứa các từ (có độ dài 4 byte), được ký hiệu là *w*[*Nb**(*Nr* + 1)]. Hàm phát sinh bảng khóa mở rộng phụ thuộc vào giá trị *Nk*, tức là phụ thuộc vào độ dài của mã khóa chính

Hàm SubWord(W) thực hiện việc thay thế (sử dụng S-box) từng byte thành phần của từ 4 byte được đưa vào và trả kết quả về là một từ bao gồm 4 byte kết quả sau khi thực hiệc việc thay thế.

Hàm RotWord(W) thực hiện việc dịch chuyển xoay vòng 4 byte thành phần (*a*, *b*, *c*, *d*) của từ được đưa vào. Kết quả trả về của hàm RotWord là một từ gồm 4 byte thành phần là (*b*, *c*, *d*, *a*)

```
KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
begin
    i=0
   while (i < Nk)
        w[i] = word[key[4*i],key[4*i+1],
                    key[4*i+2],key[4*i+3]]
        i = i + 1
    end while
    i = Nk
   while (i < Nb * (Nr + 1))
        word temp = w[i - 1]
        if (i mod Nk = 0) then
            temp = SubWord(RotWord(temp))  xor Rcon[i / Nk]
        else
            if (Nk = 8) and (i mod Nk = 4) then
                temp = SubWord(temp)
            end if
     w[i] = w[i - Nk] xor temp
    i = i + 1
    end while
end
```

Các hằng số của mỗi chu kỳ hoàn toàn độc lập với giá trị *Nk* và được xác định bằng Rcon[*i*] = (RC[*i*], {00}, {00}, {00}) với RC[*i*] $\in$ GF(2^8) và thỏa:

RC[1]=1 ({01})

RC[i] =x ({02})•(RC[i-1]) = $x^{(i-1)}$

(3.19)

b) Xác định khóa của chu kỳ

Khóa của chu kỳ thứ i được xác định bao gồm các từ (4 byte) có chỉ số từ $Nb * i$

đến $Nb * (i +1) -1$ của bảng mã khóa mở rộng. Như vậy, mã khóa của chu kỳ thứ i bao gồm các phần tử $w[Nb* i]$, $w[Nb* i +1]$,…, $w[Nb*(i +1) -1]$.

Bảng 3.3: Mã khóa mở rộng và cách xác định mã khóa của chu kỳ

(Nb = 6 và Nk = 4)

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	…
Mã khóa chu kỳ 0						Mã khóa chu kỳ 1						Mã khóa chu kỳ 2						…

Việc phát sinh mã khóa cho các chu kỳ có thể được thực hiện mà không nhất thiết phải sử dụng đến mảng $w[Nb*(Nr +1)]$. Trong trường hợp dung lượng bộ nhớ hạn chế như ở các thẻ thông minh, các mã khóa cho từng chu kỳ có thể được xác định khi cần thiết ngay trong quá trình xử lý mà chỉ cần sử dụng max(Nk,Nb)* 4 byte trong bộ nhớ.

Bảng khóa mở rộng luôn được tự động phát sinh từ khóa chính mà không cần phải được xác định trực tiếp từ người dùng hay chương trình ứng dụng. Việc chọn lựa khóa chính (Cipher Key) là hoàn toàn tự do và không có một điều kiện ràng buộc hay hạn chế nào.

3.4.2 Quy trình giải mã

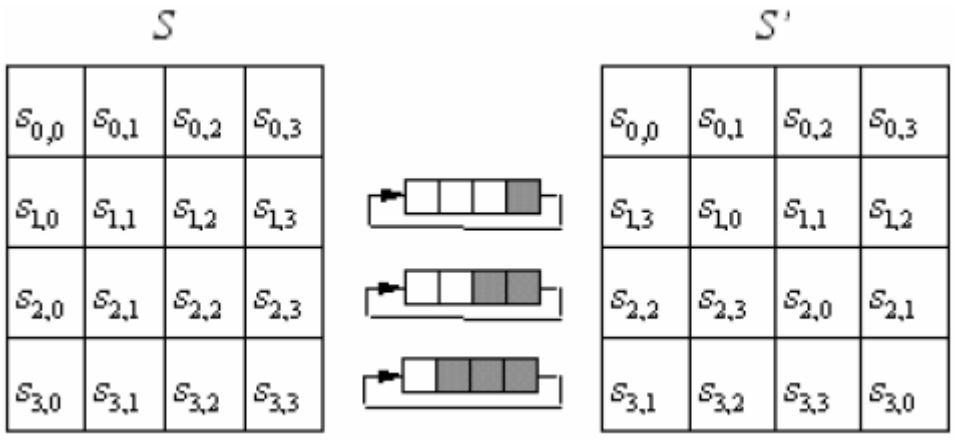
Quy trình giải mã được thực hiện qua các giai đoạn sau:

1. Thực hiện thao tác AddRoundKey đầu tiên trước khi thực hiện các chu kỳ giải mã.
2. $Nr -1$ chu kỳ giải mã bình thường: mỗi chu kỳ bao gồm bốn bước biến đổi

liên tiếp nhau: InvShiftRows, InvSubBytes, AddRoundKey,InvMixColumns.

3. Thực hiện chu kỳ giải mã cuối cùng. Trong chu kỳ này, thao tác InvMixColumns được bỏ qua.

3.4.2.1 Phép biến đổi InvShiftRows



Hình 3.6: Thao tác InvShiftRows tác động lên từng dòng của trạng thái hiện hành.

InvShiftRows chính là phép biến đổi ngược của phép biến đổi ShiftRows. Dòng đầu tiên của trạng thái sẽ vẫn được giữ nguyên trong khác ba dòng cuối của trạng thái sẽ được dịch chuyển xoay vòng theo chiều ngược với phép biến đổi ShiftRows với các di số *Nb–shift* (*r*, *Nb*) khác nhau. Các byte ở cuối dòng được đưa vòng lên đầu dòng trong khi các byte còn lại có khuynh hướng di chuyển về cuối dòng.

$$S'_{r,(c+shift(r,\ Nb))\ mod\ Nb} = S_{r,c} \text{ với } 0<r<4 \text{ và } 0\leq c<Nb \qquad (3.20)$$

Giá trị của di số *shift(r,Nb)* phụ thuộc vào chỉ số dòng *r* và kích thước *Nb* của khối và được thể hiện trong Bảng 3.1.

Đoạn mã giả cho phép biến đổi này:

```
InvShiftRows(byte state[4,Nb])
begin
    byte t[Nb]
    for r = 1 to 3
        for c = 0 to Nb - 1
            t[(c + h[r,Nb]) mod Nb] = state[r,c]
        end for
        for c = 0 to Nb - 1
            state[r,c] = t[c]
        end for
    end for
end
```

3.4.2.2 Phép biến đổi InvSubbytes

Phép biến đổi ngược của thao tác SubBytes, ký hiệu là InvSubBytes, sử dụng bảng thay thế nghịch đảo của S-box trên GF(2^8), ký hiệu là S-box^{-1}. Quá trình thay thế 1 byte *y* dựa vào S-box^{-1} bao gồm hai bước sau:

1. Áp dụng phép biến đổi affine (trên GF(2)) sau đối với *y* (có biểu diễn nhị phân là $\{y_7\, y_6\, y_5\, y_4\, y_3\, y_2\, y_1 y_0\}$):

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.21)$$

Hay : $x_i = y_{(i+2)\ mod8} \oplus y_{(i+5)\ mod8} \oplus y_{(i+7)mod8} \oplus d_{i,}$ (3.22)

với d_i là bit thứ i của giá trị {05}, $0 \le i \le 7$

Đây chính là phép biến đổi affine ngược của phép biến đổi affine ở bước 1 của S-box

2. Gọi *x* là phần tử thuộc GF(2^8) có biểu diễn nhị phân là $\{x_7\, x_6 x_5 x_4 x_3 x_2 x_1 x_0\}$.

Xác định phần tử nghịch đảo x^{-1} GF(2^8) với quy ước $\{00\}^{-1} = \{00\}$

Bảng 3.4: Bảng thay thế nghịch đảo giá trị {xy} ở dạng thập lục phân

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

Đoạn mã giả cho phép InvSubBytes

```
InvSubBytes(byte state[4,Nb])
begin
    for r = 0 to 3
        for c = 0 to Nb - 1
            state[r,c] = InvSbox[state[r,c]]
        end for
    end for
end
```

3.4.2.3 Phép biến đổi InvMixColumns

InvMixColumns là biến đổi ngược của phép biến đổi MixColumns. Mỗi cột của trạng thái hiện hành được xem như đa thức $s(x)$ bậc 4 có các hệ số thuộc GF(2^8) và được nhân với đa thức $a^{-1}(x)$ là nghịch đảo của đa thức $a(x)$ (modulo $M(x)$) được sử dụng trong phép biến đổi MixColumns.

$$a^{-1}(x) = \{0b\}\, x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \qquad (3.23)$$

Phép nhân $s'(x) = a^{-1}(x) \otimes s(x)$ có thể được biểu diễn dưới dạng ma trận:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \text{ với } 0 \le c < Nb \qquad (3.24)$$

Trong đoạn mã giả sau, hàm Ffmul (x, y) thực hiện phép nhân trường GF(2^8) hai phần tử x và y với nhau.

```
InvMixColumns(byte block[4,Nb])
begin
    byte t[4]
    for c = 0 to Nb - 1
        for r = 0 to 3
            t[r] = block[r,c]
        end for
        for r = 0 to 3
```

```
            block[r,c] =
                FFmul(0x0e, t[r])                xor
                FFmul(0x0b, t[(r + 1) mod 4])    xor
                FFmul(0x0d, t[(r + 2) mod 4])    xor
                FFmul(0x09, t[(r + 3) mod 4])
        end for
    end for
end
```

3.4.3 Các vấn đề cài đặt thuật toán

Gọi *a* là trạng thái khi bắt đầu chu kỳ mã hóa. Gọi *b*, *c*, *d*, *e* lần lượt là trạng thái. Kết quả đầu ra sau khi thực hiện các phép biến đổi SubBytes, ShiftRows, MixColumns và AddRoundKey trong chu kỳ đang xét. Quy ước: trong trạng thái *s* (*s* = *a*,*b*, *c*,*d*, *e*), cột thứ *j* được kí hiệu s_j, phần tử tại dòng *i* cột *j* kí hiệu là $s_{i,j}$.

Sau biến đổi SubBytes**:**

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j}] \\ S[a_{2,j}] \\ S[a_{3,j}] \end{bmatrix} \quad (3.25)$$

Sau biến đổi ShiftRows:

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,(j+shift(1,Nb))\bmod Nb} \\ b_{2,(j+shift(2,Nb))\bmod Nb} \\ b_{3,(j+shift(3,Nb))\bmod Nb} \end{bmatrix} \quad (3.26)$$

Sau biến đổi MixColumns**:**

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} \quad (3.27)$$

Sau biến đổi AddRoundKey:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad (3.28)$$

Kết hợp các kết quả trung gian của mỗi phép biến đổi trong cùng chu kỳ với nhau ta có:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S\left[a_{1,(j+shift(1,Nb))\bmod Nb}\right] \\ S\left[a_{2,(j+shift(2,Nb))\bmod Nb}\right] \\ S\left[a_{3,(j+shift(3,Nb))\bmod Nb}\right] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad (3.29)$$

Kí hiệu j[r] = (j + shift(r, Nb)) mod Nb ta có:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j[0]}] \\ S\left[a_{1,j[1]}\right] \\ S\left[a_{2,j[2]}\right] \\ S\left[a_{3,j[3]}\right] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad (3.30)$$

Khai triển phép nhân ma trận ta có:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S\left[a_{0,j[0]}\right] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S\left[a_{1,j[1]}\right] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S\left[a_{2,j[2]}\right] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S\left[a_{3,j[3]}\right] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \quad (3.31)$$

Định nghĩa các bảng tra cứu T_0, T_1, T_2, T_3 như sau:

$$T_0[a]=\begin{bmatrix} S[a]\bullet 02 \\ S[a] \\ S[a] \\ S[a]\bullet 03 \end{bmatrix}, T_1[a]=\begin{bmatrix} S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \\ S[a] \end{bmatrix},$$
$$T_2[a]=\begin{bmatrix} S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \end{bmatrix}, T_3[a]=\begin{bmatrix} S[a] \\ S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \end{bmatrix} \quad (3.32)$$

Khi đó ta viết lại biểu thức (3.32) như sau:

$$e_j = \left(\bigoplus_{i=0}^{3} T_i\left[a_{i,j[i]}\right] \right) \oplus w_{round*Nb+j} \quad (3.33)$$

Với round là số thứ tự chu kỳ đang xét.

Như vậy, mỗi cột *ej* của trạng thái kết quả sau khi thực hiện một chu kỳ mã hóa có thể được xác định bằng bốn phép toán XOR trên các số nguyên 32 bit sử dụng bốn bảng tra cứu *T*0, *T*1, *T*2 và *T*3.

Công thức (3.33) chỉ áp dụng được cho *Nr*-1 chu kì đầu. Do chu kỳ cuối cùng không thực hiện phép biến đổi MixColumns nên cần xây dựng 4 bảng tra cứu riêng cho chu kì này:

$$U_0[a]=\begin{bmatrix} S[a] \\ 0 \\ 0 \\ 0 \end{bmatrix}, U_1[a]=\begin{bmatrix} 0 \\ S[a] \\ 0 \\ 0 \end{bmatrix}, U_2[a]=\begin{bmatrix} 0 \\ 0 \\ S[a] \\ 0 \end{bmatrix}, U_3[a]=\begin{bmatrix} 0 \\ 0 \\ 0 \\ S[a] \end{bmatrix} \quad (3.34)$$

3.4.4 Kết quả thử nghiệm

Kích thước (bit)		Tốc độ xử lý (Mbit/giây)							
		Pentium 200 MHz		Pentium II 400 MHz		Pentium III 733 MHz		Pentium IV 2.4 GHz	
Khóa	Khối	C++	C	C++	C	C++	C	C++	C
128	128	69.4	70.5	138.0	141.5	252.9	259.2	863.0	884.7
192	128	58.0	59.8	116.2	119.7	212.9	219.3	726.5	748.3
256	128	50.1	51.3	101.2	101.5	185.5	186.1	633.5	634.9

Bảng 3.5: Tốc độ xử lý của phương pháp Rijndael

Kết quả thử nghiệm thuật toán Rijndael được ghi nhận trên máy Pentium 200 MHz (sử dụng hệ điều hành Microsoft Windows 98), máy Pentium II 400 MHz, Pentium III 733 MHz (sử dụng hệ điều hành Microsoft Windows 2000 Professional), Pentium IV 2,4GHz (sử dụng hệ điều hành Microsoft Windows XP Service Pack 2).

3.4.5 Kết luận

3.4.5.1 Khả năng an toàn

- Việc sử dụng các hằng số khác nhau ứng với mỗi chu kỳ giúp hạn chế khả năng tính đối xứng trong thuật toán.

- Sự khác nhau trong cấu trúc của việc mã hóa và giải mã đã hạn chế được các khóa "yếu" (weak key) như trong phương pháp DES.

- Trong các phiên bản mở rộng, các khóa được sử dụng thông qua thao tác XOR và tất cả những thao tác phi tuyến đều được cố định sẵn trong S-box mà không phụ thuộc vào giá trị cụ thể của mã khóa.

- Tính chất phi tuyến cùng khả năng khuếch tán thông tin (diffusion) trong việc tạo bảng mã khóa mở rộng làm cho việc phân tích mật mã dựa vào các khóa tương đương hay các khóa có liên quan trở nên không khả thi.

- Trong trường hợp thuật toán Rijndael với số lượng chu kỳ lớn hơn 6, không tồn tại phương pháp công phá mật mã nào hiệu quả hơn phương pháp thử và sai.

- Tính chất phức tạp của biểu thức S-box trên GF(28) cùng với hiệu ứng khuếch tán giúp cho thuật toán không thể bị phân tích bằng phương pháp nội suy.

3.4.5.2 Đánh giá

Phương pháp Rijndael thích hợp cho việc triển khai trên nhiều hệ thống khác nhau, không chỉ trên các máy tính cá nhân mà điển hình là sử dụng các chip Pentium, mà cả trên các hệ thống thẻ thông minh. Trên các máy tính cá nhân, thuật toán AES thực hiện việc xử lý rất nhanh so với các phương pháp mã hóa khác. Trên các hệ thống thẻ thông minh, phương pháp này càng phát huy ưu điểm không chỉ nhờ vào tốc độ xử lý cao mà còn nhờ vào mã chương trình ngắn gọn, thao tác xử lý sử dụng ít bộ nhớ. Ngoài ra, tất cả các bước xử lý của việc mã hóa và giải mã đều được thiết kế thích hợp với cơ chế xử lý song song nên phương pháp Rijndael càng chứng tỏ thế mạnh của mình trên các hệ thống thiết bị mới. Do đặc tính của việc xử lý thao tác trên từng byte dữ liệu nên không có sự khác biệt nào được đặt ra khi triển khai trên hệ thống big-endian hay little-endian.

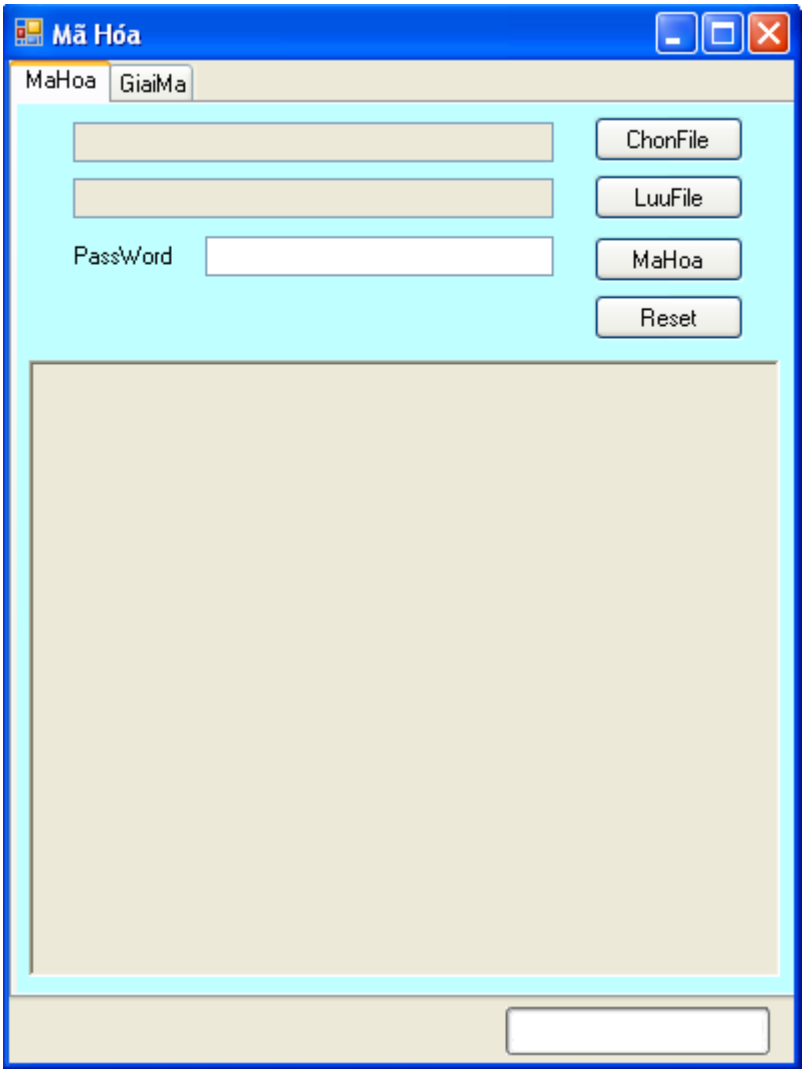
Xuyên suốt phương pháp AES, yêu cầu đơn giản trong việc thiết kế cùng tính linh hoạt trong xử lý luôn được đặt ra và đã được đáp ứng. Độ lớn của khối dữ liệu cũng như của mã khóa chính có thể tùy biến linh hoạt từ 128 đến 256-bit với điều kiện là chia hết cho 32. Số lượng chu kỳ có thể được thay đổi tùy thuộc vào yêu cầu riêng được đặt ra cho từng ứng dụng và hệ thống cụ thể.

Tuy nhiên, vẫn tồn tại một số hạn chế mà hầu hết liên quan đến quá trình giải mã. Mã chương trình cũng như thời gian xử lý của việc giải mã tương đối lớn hơn việc mã hóa, mặc dù thời gian này vẫn nhanh hơn đáng kể so với một số phương pháp khác. Khi cài đặt bằng chương trình, do quá trình mã hóa và giải mã không giống nhau nên không thể tận dụng lại toàn bộ đoạn chương trình mã hóa cũng như các bảng tra cứu cho việc giải mã. Khi cài đặt trên phần cứng, việc giải mã chỉ sử dụng lại một phần các mạch điện tử sử dụng trong việc mã hóa và với trình tự sử dụng khác nhau.

Phương pháp Rijndael với mức độ an toàn rất cao cùng các ưu điểm đáng chú ý khác chắc chắn sẽ nhanh chóng được áp dụng rộng rãi trong nhiều ứng dụng trên các hệ thống khác nhau.

3.5 Ứng dụng của thuật toán

3.5.1 Giao diện chương trình



Hình 3.7: Giao diện chương trình.

3.5.2 Chức năng chính của chương trình

3.5.2.1 Mã hóa

Trong quá trình mã hóa thực hiện các bước:

- Chọn file cần mã hóa bằng cách nhấn nút “ChonFile”.
- Nút “LuuFile” cho phép bạn lưu file cần mã hóa dưới dạng đuôi .rij .
- Nhập PassWord.

- Khi nhấp nút “MaHoa” file bạn muốn mã hóa sẽ được thực hiện. Kết quả sẽ được hiển thị ở bảng trắng phía dưới. Bảng này chỉ cho phép người dùng thấy được thông tin sau khi đã mã hóa, không cho phép người dùng nhập thêm vào.

Dưới cùng là thanh progress để biết tiến độ của quá trình mã hóa.

Các bước trên được thực hiện tuần tự.

3.5.2.2 Giải mã

Trong quá trình giải mã ta cần thực hiện:

- Chọn file cần giãi mã bằng cách nhấp “ChonFile”.

- Sau đó, chương trình tự động load file key dưới dạng XML trong cùng thư mục.

- Nếu muốn lưu file ở thư mục khác bạn sẽ nhấn nút “LuuFile” để thực hiện.

- Nếu file key là hợp lệ nút “GiaiMa” cho phép bạn thực hiện quá trình giải mã file với việc tự động nhập PassWord.

Dưới cùng là thanh progress để biết tiến độ quá trình giải mã.

3.5.3 Code thực hiện mã hóa và giải mã

// Mã Hóa

```
private void EncryptFile(string scrFileName, string destFileName,
        byte[] key, byte[] iv)
{
  Stream scrFile =
    new FileStream(scrFileName, FileMode.Open, FileAccess.Read);
  Stream rijFile =
        new FileStream(destFileName, FileMode.Create,
  FileAccess.Write);
  using (SymmetricAlgorithm alg =
                SymmetricAlgorithm.Create("Rijndael"))
 {
  alg.Key = key;
```

```
                alg.IV = iv;
               progressBar1.Minimum = 0;
                progressBar1.Maximum = Convert.ToInt16(scrFile.Length / 1024) +
1;
                progressBar1.Value = 1;
                progressBar1.Step = 1;
                CryptoStream cryptoStream = new CryptoStream(scrFile,
                              alg.CreateEncryptor(), CryptoStreamMode.Read);
                int bufferlength;
                byte[] buffer = new byte[1024];
                rijFile.Write(key, 0, 16);
                rijFile.Write(iv, 0, 16);
                  do
                   {
                      bufferlength = cryptoStream.Read(buffer, 0, 1024);
                      rijFile.Write(buffer, 0, bufferlength);
                      ChuoiMaHoa += "\n" + BitConverter.ToString(buffer, 0,
bufferlength);
                    progressBar1.PerformStep();
                   }
                  while (bufferlength > 0);
                   rijFile.Flush();
                   Array.Clear(key, 0, key.Length);
                   Array.Clear(iv, 0, iv.Length);
                   cryptoStream.Clear();
                   cryptoStream.Close();
                   scrFile.Close();
                   rijFile.Close();
```

```
                MessageBox.Show("Ma hoa file thanh cong");
            }
        }
    }
    // Giải mã
        private void DecryptFile(string scrFileName, string destFileName,
byte[] key, byte[] iv)
        {
            Stream scrFile =
                new FileStream(scrFileName, FileMode.Open,
FileAccess.Read);
            Stream destFile =
                new FileStream(destFileName, FileMode.Create,
                    FileAccess.Write);
            using (SymmetricAlgorithm alg =
                    SymmetricAlgorithm.Create("Rijndael"))
            {
                alg.Key = key;
                alg.IV = iv;
                CryptoStream cryptoStream = new CryptoStream(destFile,
                    alg.CreateDecryptor(),
CryptoStreamMode.Write)
                int bufferlength, i = 0;
                byte[] buffer = new byte[1024];
                progressBar1.Minimum = 0;
                progressBar1.Maximum = Convert.ToInt16(scrFile.Length/1024)
+1;
                progressBar1.Value = 1;
```

```
progressBar1.Step = 1;
do
{
   if (i == 0)
   {
      bufferlength = scrFile.Read(buffer, 0, 16);
      bufferlength = scrFile.Read(buffer, 0, 16);
      i++;
   }
   else
   {
      bufferlength = scrFile.Read(buffer, 0, 1024);
      cryptoStream.Write(buffer, 0, bufferlength);
      ChuoiGiaiMa += "\n" + BitConverter.ToString(buffer,
                                 0, bufferlength);
      progressBar1.PerformStep();
   }
}
while (bufferlength > 0);

cryptoStream.FlushFinalBlock();
Array.Clear(key, 0, key.Length);
Array.Clear(iv, 0, iv.Length);
cryptoStream.Clear();
cryptoStream.Close();
scrFile.Close();
destFile.Close();
```

```
                MessageBox.Show("Giai ma xong", "Thong Bao");
            }
        }
    }
}
```

KẾT LUẬN

Hiện nay, với sự phát triển của khoa học hiện đại và Công nghệ thông tin, ngành mật mã đã có những bước phát triển mạnh mẽ, đạt được nhiều kết quả lý thuyết sâu sắc và tạo cơ sở cho việc phát triển các giải pháp bảo mật, an toàn thông tin trong mọi lĩnh vực hoạt động của con người.

Tìm hiểu qua các tài liệu đề tài đã hệ thống lại các kiến thức cơ bản của lĩnh vực mã hóa, tập trung tìm hiểu phương pháp mã hóa khóa đối xứng và nghiên cứu từng bước thực hiện của thuật toán mã hóa Rijndael. Bước đầu xây dựng chương trình mã hóa tệp tin bằng thuật toán Rijndael sử dụng thư viện Cryptography.

Đồ án tuy còn nhiều điểm cần phải nghiên cứu và hoàn thiện nhưng do thời gian và trình độ còn hạn chế nên không thể tránh khỏi những thiếu xót,nhược điểm. Em rất mong được sự góp ý của các Thầy, Cô và các bạn.

Em xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

Tài liệu tiếng Việt

[1]. NHÓM TÁC GIẢ: TS. Dương Anh Đức - ThS. Trần Minh Triết cùng với sự đóng góp của các sinh viên Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia thành phố Hồ Chí Minh - Book_MaHoaVaUngDung.

[2]. Lê Thụy – ATBMTT1 - Trường Đại học Dân lập Hải Phòng.

[3]. Phan Đình Diệu – Lý thuyết mật mã & an toàn thông tin – Đại học quốc gia Hà Nội.

[4]. Phạm Trọng Huy – Giáo trình cấu trúc dữ liệu – Khoa Kỹ Thuật trường cao đẳng kinh tế, kĩ thuật Hải Dương.

[5].An toàn thông tin mạng máy tính, truyền tin số và truyền dữ liệu – PGS,TS.Thái Hồng Nhị & TS. Phạm Minh Việt.

Tài liệu tiếng anh

[6]. fip – Federal Information Processing Standards Publication 197 Specification for the Advanced Encrytion Standard .(November 26,2001)

[7]. A specification for Rijndael, the AES Algorithm - Dr.Brian Gladman, v3.1, 3 rd March 2001.