**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á**

**KHOA: CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN**

**Học phần: An toàn bảo mật**

**Nhóm: 18**

**Đề tài 7: Tìm hiểu về mã hoá đối xứng và  
Xây dựng chương trình mô tả thuật toán AES**

|  |  |  |
| --- | --- | --- |
| **Sinh viên thực hiện** | **Lớp** | **Khóa** |
| **Nguyễn Anh Vinh** | **DCCNTT11.10.1** | **11** |
| **Nguyễn Hồng Quân** | **DCCNTT11.10.1** | **11** |

**Bắc Ninh, ngày 28 tháng 03 năm 2022**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ ĐÔNG Á**

**KHOA: CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN**

**Học phần: An toàn bảo mật**

**Nhóm: 18**

**Đề tài 7: Tìm hiểu về mã hoá đối xứng và  
Xây dựng chương trình mô tả thuật toán AES**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Sinh viên thực hiện** | **Mã sinh viên** | **Điểm bằng số** | **Điểm bằng chữ** |
| **1** | **Nguyễn Anh Vinh** | **20200122** |  |  |
| **2** | **Nguyễn Hồng Quân** | **20200125** |  |  |

|  |  |
| --- | --- |
| **CÁN BỘ CHẤM 1**  *(Ký và ghi rõ họ tên)* | **CÁN BỘ CHẤM 2**  *(Ký và ghi rõ họ tên)* |
|  |  |

**Bắc Ninh, ngày 28 tháng 03 năm 2022**

**Mục lục**

[Lời nói đầu 1](#_Toc133401670)

[Chương I: Tổng quan đề tài 2](#_Toc133401671)

[1.1. Tổng quan hiện trạng, giới thiệu đề tài. 2](#_Toc133401672)

[1.2. Lí do lựa chọn đề tài. 2](#_Toc133401673)

[1.3. Mục tiêu, nội dung, phương pháp nghiên cứu của đề tài 2](#_Toc133401674)

[1.4. Đối tượng và phạm vi nghiên cứu 3](#_Toc133401675)

[Chương II: Lý thuyết hệ mã hoá đối xứng. 4](#_Toc133401676)

[2.1. Hệ thống thông tin và các khái niệm cơ bản 4](#_Toc133401677)

[**2.1.1. Dữ liệu và thông tin 4**](#_Toc133401678)

[**2.1.2. Hệ thống thông tin 4**](#_Toc133401679)

[**2.1.3. Mục tiêu của việc bảo mật hệ thống thông tin: 5**](#_Toc133401680)

[**2.1.4. Phương pháp và chiến lược để bảo vệ an toàn cho hệ thống 5**](#_Toc133401681)

[**2.1.5. Mã hoá thông tin 5**](#_Toc133401682)

[2.2. Hệ mã hoá đối xứng. 6](#_Toc133401683)

[**2.2.1. Tốc độ xử lý. 7**](#_Toc133401684)

[**2.2.2. Hạn chế. 7**](#_Toc133401685)

[Chương III: Cơ sở lý thuyết và hệ mã AES 8](#_Toc133401686)

[3.1. Giới Thiệu 8](#_Toc133401687)

[3.2. Các khái niệm và định nghĩa: 8](#_Toc133401688)

[**3.2.1. Các khái niệm và ký hiệu 8**](#_Toc133401689)

[**3.2.2. Các hàm, thủ tục, ký hiệu và các tham số của thuật toán 9**](#_Toc133401690)

[**3.2.3 Các khái niệm và quy ước 11**](#_Toc133401691)

[**3.2.4. Cơ sở toán học 13**](#_Toc133401692)

[Chương IV: Thuật toán AES 16](#_Toc133401693)

[4.1. Cấu trúc thuật toán AES 16](#_Toc133401694)

[4.2. Chi tiết một vòng lặp (Từ vòng 1 đến N-1) 16](#_Toc133401695)

[4.3. Thuật toán mã hoá 18](#_Toc133401696)

[**4.3.1. Sơ đồ khối các bước thực hiện 18**](#_Toc133401697)

[**4.3.2. Chi tiết các bước mã hoá 20**](#_Toc133401698)

[4.4. Thuật toán giải mã 25](#_Toc133401699)

[**4.4.1. Sơ đồ khối các bước thực hiện 25**](#_Toc133401700)

[**4.4.2. Chi tiết các bước thực hiện 26**](#_Toc133401701)

[Chương 5: MÔ PHỎNG THUẬT TOÁN 30](#_Toc133401702)

[KẾT LUẬN VÀ ĐÁNH GIÁ 30](#_Toc133401703)

[5.1. Ứng dụng của thuật toán AES 30](#_Toc133401704)

[5.2. Mô phỏng thuật toán AES 128-bit 31](#_Toc133401705)

[5.3. Kết quả khi chạy chương trình mô phỏng 40](#_Toc133401706)

[5.4. Tính bảo mật của AES 44](#_Toc133401707)

[5.5. KẾT LUẬN 45](#_Toc133401708)

[**5.5.1. Đánh giá thuật toán 45**](#_Toc133401709)

[**5.5.2. Đánh giá AES 45**](#_Toc133401710)

[TÀI LIỆU THAM KHẢO 47](#_Toc133401711)

**Danh mục các từ viết tắt**

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt (Tiếng anh)** | **Nghĩa tiếng anh** | **Nghĩa tiếng việt** |
| AES | Advanced Encryption Standard | Chuẩn mã hoá nâng cao |
| APT | Advanced Persistent Threat | Cuộc tấn công có chủ đích |
| BTS | Base Transceiver Station | Trạm thu phát sóng |
| CIPHERTEXT | Ciphertext | Bản mã |
| DECRYPTION | Decryption | Giải mã |
| ENCRYPTION | Encryption | Quá trình mã hóa |
| GF | Galois | Trường hữu hạn |
| KEY EXPANSION | Key Expansion | Mở rộng khoá |
| NIST | National Institute Standards and Technology | Viện tiêu chuẩn và Công nghệ quốc gia Hoa Kỳ |
| NSA | National Security Agency | Cơ quan an ninh Quốc gia |
| PLAINTEXT | Plaintext | Bản rõ |

# **Lời nói đầu**

Trong những năm gần đây, Công nghệ thông tin và Internet đã phát triển mạnh mẽ đi kèm với những dấu mốc quan trọng. Trải qua các cuộc cách mạng công nghiệp lớn, đặc biệt là cuộc cách mạng công nghệ 4.0, nhu cầu của con người trong việc sử dụng mạng Internet đã tăng mạnh. Các dịch vụ và ứng dụng trên Internet cũng ngày càng phát triển kéo theo đó yêu cầu bảo mật thông tin trên mạng thông tin cũng được ưu tiên phát triển.

An ninh, bảo mật hệ thông tin hiện nay là một vấn đề đang được rất nhiều nước trên thế giới quan tâm, đặc biệt là đối với các cơ quan hành chính nhà nước trong việc bảo vệ các dữ liệu, thông tin, các tài liệu điện tử.

Từ đây vấn đề đặt ra yêu cầu cần phải có một phương pháp mã hoá bảo mật đơn giản nhưng hiệu quả. Trong bài dưới đây, nhóm chúng tôi muốn giới thiệu một chuẩn mã hoá được coi là một trong những phương pháp mã hoá an toàn và hiệu quả: "**Chuẩn mã hoá nâng cao AES – Advanced Encryption Standard.**

Do thời gian có hạn và kiến thức chuyên môn còn hạn chế nên bài viết có thể chưa được hoàn chỉnh. Rất mong nhận được sự góp ý, bổ sung và sửa đổi của thầy, cô giáo và các bạn giúp cho bài báo cáo được hoàn thiện hơn. Mọi ý kiến bổ sung, góp ý xin gửi về địa chỉ email: **maiyonaisu1102@gmail.com** hoặc số điện thoại **0822206919.**

# **Chương I: Tổng quan đề tài**

* 1. **Tổng quan hiện trạng, giới thiệu đề tài.**

Với sự phát triển của kỷ nguyên công nghệ số và cuộc cách mạng công nghiệp 4.0, năm 2023 các cuộc tấn công lừa đảo trực tuyến và đánh cắp thông tin vẫn đang rất phức tạp, và ngày càng gia tăng về số lượng và phương thức.

Các đối tượng xấu sẽ cố gắng lợi dụng lượng lớn thông tin liên quan đến người dùng để thực hiện hành vi tấn công trực tuyến. Hơn thế nữa, tại Việt Nam việc các doanh nghiệp, tổ chức đang đẩy mạnh chuyển đổi số ở đa dạng các lĩnh vực, từ chính phủ, y tế, giáo dục cho tới du lịch, thương mại… cũng đang đặt ra hàng loạt thách thức mới cho vấn đề đảm bảo an toàn bảo mật.

* 1. **Lí do lựa chọn đề tài.**

Trong kỷ nguyên số, Mạng máy tính là một môi trường mở và những thông tin được đưa lên Internet hoặc nhận về Internet đều có thể bị lộ bởi các đối tượng xấu. Một trong những phương thức để bảo mật dữ liệu an toàn và được sử dụng phổ biến hiện nay chính là mã hóa thông tin. Tuy nhiên các phương pháp mã hoá thông tin trước đó được áp dụng đã bị tấn công bởi các tin tặc. Do đó chúng ta cần nghiên cứu một phương pháp mã hoá bảo mật có độ bảo mật cao hơn, phương pháp thực hiện dễ dàng nhưng hiệu quả đem lại cao.

Nhận thức được lợi ích và tầm quan trọng của bảo mật thông tin trong công nghệ thông tin và truyền thông trong việc duy trì và thúc đẩy sự phát triển bền vững, trong những năm vừa qua lĩnh vực công nghệ thông tin nói chung và hoạt động ứng dụng công nghệ thông tin nói riêng đã phát triển nhanh chóng, mạnh mẽ và ngày càng sâu rộng trong mọi mặt của đời sống kinh tế xã hội. Các hoạt động có thể kể đến như: Các hoạt động thương mại điện tử; các hoạt động hành chính công phổ biến như gửi nhận thư điện tử, các hệ thống hỗ trợ quản lý điều hành trên mạng. Chính vì những vấn đề thực tiễn trên, nhóm chúng tôi quyết định chọn đề tài về: “**Tìm hiểu về mã hoá đối xứng, xây dựng chương trình mô tả thuật toán AES”** nhằm nâng cao việc xác thực và an toàn thông tin trong các hoạt động của các cơ quan, tổ chức trong các ứng dụng trên môi trường mạng.

* 1. **Mục tiêu, nội dung, phương pháp nghiên cứu của đề tài**
* **Mục tiêu:**

Bài làm này nhằm mục đích đề xuất một thuật toán mã hoá khối, sử dụng khoá mã đối xứng để mã hoá và giải mã các tệp tin, nội dung văn bản yêu cầu tính bảo mật cao. Từ đó đề xuất áp dụng phổ biến rộng rãi việc sử dụng thuật toán AES để tạo lớp bảo vệ an toàn cho các nội dung tuyệt mật.

* **Nội dung:**
* Tìm hiểu về mã hoá đối xứng.
* Nghiên cứu về mã hoá và giải mã của mã hoá khối AES
* Đánh giá về mức độ an toàn của mã hoá AES trong việc bảo mật thông tin
* Mô phỏng thuật toán của hệ mã hoá dựa trên ngôn ngữ C++
* **Phương pháp nghiên cứu**
* Trong bài viết này, nhóm chúng em sẽ tìm hiểu khái niệm cơ bản, các hàm toán học được sử dụng và từng bước thuật toán được áp dụng để mã hoá. Sau đó nhóm sẽ minh hoạ dựa trên phần mềm được lập trình trên Visual Studio để mã hoá và giải mã bản rõ minh hoạ.
  1. **Đối tượng và phạm vi nghiên cứu**
* Đối tượng nghiên cứu: mã hoá đối xứng, mã hoá khối AES.
* Phạm vi nghiên cứu: Cơ sở lý thuyết mã hoá đối xứng, thuật toán mã hoá và giải mã của mã hoá khối AES.

# **Chương II: Lý thuyết hệ mã hoá đối xứng.**

* 1. **Hệ thống thông tin và các khái niệm cơ bản**
     1. **Dữ liệu và thông tin**
* **Dữ liệu (Data)** là các giá trị của thông tin định lượng hoặc định tính của các sự vật, hiện tượng trong cuộc sống. Trong tin học, dữ liệu được dùng như một cách biểu diễn hình thức hoá của thông tin về các sự kiện, hiện tượng thích ứng với các yêu cầu truyền nhận, thể hiện và xử lí bằng máy tính.
* **Thông tin (Information)** là dữ liệu đã được xử lý, phân tích, tổ chức nhằm mục đích hiểu rõ hơn sự vật, sự việc, hiện tượng theo một góc độ nhất định.
  + 1. **Hệ thống thông tin**
* **Hệ thống thông tin (Information Systems)** là một hệ thống gồm con người, dữ liệu và những hoạt động xử lý dữ liệu và thông tin trong một tổ chức.
* **Bảo mật hệ thống thông tin (Information Systems Security)** là bảo vệ hệ thống thông tin chống lại việc truy cập, sử dụng, chỉnh sửa, phá hủy, làm lộ và làm gián đoạn thông tin và hoạt động của hệ thống một cách trái phép.
* **Những yêu cầu bảo mật hệ thống thông tin**
  + Tính bí mật (Confidentiality): bảo vệ dữ liệu không bị lộ ra ngoài một cách trái phép. Đảm bảo thông tin không bị lọt ra ngoài hoặc nếu có lọt ra ngoài thì bên thứ ba cũng không biết được nội dung của thông tin.
  + Tính toàn vẹn (Integrity): Chỉ những người dùng được ủy quyền mới được phép chỉnh sửa dữ liệu. Đảm bảo thông tin chuyển đi đúng của người gửi và tới đúng người nhận thông tin đó với nội dung là của đúng người gửi gửi cho người nhận.
  + Tính sẵn sàng (Availability): Đảm bảo dữ liệu luôn sẵn sàng khi những người dùng hoặc ứng dụng được ủy quyền yêu cầu.
  + Tính chống thoái thác (Non-repudiation): Khả năng ngăn chặn việc từ chối một hành vi đã làm.

* + 1. **Mục tiêu của việc bảo mật hệ thống thông tin:**

Việc bảo mật thông tin bao gồm những mục tiêu sau:

* Ngăn chặn: Ngăn chặn kẻ tấn công vi phạm các chính sách bảo mật
* Phát hiện: Phát hiện các vi phạm chính sách bảo mật
* Phục hồi: Chặn các hành vi vi phạm đang diễn ra, đánh giá và sửa lỗi. Tiếp tục hoạt động bình thường ngay cả khi tấn công đã xảy ra
  + 1. **Phương pháp và chiến lược để bảo vệ an toàn cho hệ thống**
* **Giới hạn quyền tối thiểu:** đây là chiến lược cơ bản nhất. Theo nguyên tắc này bất kỳ một đối tượng nào cũng chỉ có những quyền hạn nhất định đối với tài nguyên mạng, khi thâm nhập vào mạng đôi tượng đó chỉ được sử dụng một số tài nguyên mạng nhất định.
* **Bảo vệ theo chiều sâu:** sử dụng nhiều biện pháp khác nhau bảo vệ cho hệ thống: Phần cứng, phần mềm, … để tương trợ hỗ trợ nhau.
* **Điểm nối yếu nhất:** chiến lược này dựa trên nguyên tắc:” Một dây xích chỉ chắc tại mắt duy nhất, một bức tường chỉ cứng tại điểm yếu nhất”. Kẻ tấn công thường tìm chỗ yếu nhất của hệ thống để tấn công, do đó ta cần phải gia cố các điểm yếu của hệ thống. Thông thường chúng ta chỉ quan tâm đến kẻ tấn công trên mạng hơn là kẻ tấn tiếp cận hệ thống, do đó an toàn vật lý được coi là điểm yếu nhất trong hệ thống của chúng ta.
* **Tính toàn cục:** Các hệ thống an toàn đòi hỏi phải có tính toàn cục của các hệ thống cục bộ. Nếu có một kẻ nào đó có thể bẻ gãy một cơ chế an toàn thì chúng cũng có thể thành công bằng cách tấn công hệ thống tương tự của ai đó và sau đó tấn công hệ thống từ nội bộ bên trong.
* **Tính đa dạng bảo vệ:** cần phải sử dụng nhiều biện pháp bảo vệ khác nhau cho hệ thống khác nhau, nếu không có kẻ tấn công vào được một hệ thống thì chúng cũng dễ dàng tấn công vào các hệ thống khác.
  + 1. **Mã hoá thông tin**

**Mã hóa** là cách xáo trộn dữ liệu chỉ để hai bên trao đổi thông tin có thể hiểu được. Về mặt kỹ thuật, đó là quá trình chuyển đổi văn bản gốc sang bản mã. Nói một cách đơn giản hơn, mã hóa lấy dữ liệu có thể đọc được và thay đổi nó để dữ liệu này không giống như ban đầu. Mã hóa yêu cầu sử dụng khóa mã hóa: một tập hợp các giá trị toán học mà cả người gửi và người nhận tin nhắn được mã hóa đều biết.

Ngày nay, cùng với sự trợ giúp của các máy tính điện tử. Các phương pháp mã hóa với khóa bí mật được sử dụng chung cho quá trình mã hóa và giải mã (hay còn gọi là mã hóa cổ điển) có thể dễ dàng bị giải mã.

Do vậy để đảm bảo sự an toàn cho các thông tin tuyệt mật, các phương pháp mã hoá an toàn hơn với sự kết hợp của các kết quả nghiên cứu của toán học là điều thực sự cần thiết. Các phương pháp mã hoá khác nhau có những ưu điểm và nhược điểm khác nhau tuỳ theo phương pháp mã hoá và biểu thức toán học được sử dụng. Người sử dụng sẽ cân nhắc lựa chọn phương pháp mã hoá phù hợp nhất đối với nhu cầu, mục đích bảo vệ của mình. Độ khó của thuật toán sẽ dựa trên thời gian, chi phí tạo ra thuật toán do vậy người dùng sẽ phải cân nhắc giữa bảo mật và chi phí.

Các thuật toán mã hoá đều có chung các thuật ngữ sau:

* **Bản rõ (Plaintext):** Hay còn gọi là “Văn bản thô”, đây có thể hiểu là một đoạn văn bản, tin nhắn đơn giản, không mã hoá do vậy bất cứ ai cũng có thể đọc được.
* **Bản mã (Ciphertext):** Đây là kết quả của quá trình mã hoá. Bản rõ sẽ được mã hoá và xuất hiện bản mã dưới dạng các chuỗi ký tự “ngẫu nghiên”. Bản mã còn có tên gọi khác là “Mật mã”, đây là một cách khác để chỉ thuật toán mã hoá biến đổi mã hoá biến đổi bản rõ.
* **Mã hoá (Encryption):** Đây là quá trình áp dụng một phương pháp mã hoá sử dụng một hàm toán học vào nội dung của một tệp tin khiến nội dung của nó không thể đọc được và không thể truy cập được trừ khi người dùng có “Khoá giải mã”
* **Giải mã (Decryption):** Sau khi mã hoá tệp, nếu đảo ngược lại quá trình đó sẽ là giải mã. Quá trình này sẽ chuyển bản mã trở lại thành bản rõ. Việc giải mã sẽ yêu cầu hai yếu tố đó là: Mật khẩu (Chìa khoá) chính xác và thuật toán giải mã tương ứng.
  1. **Hệ mã hoá đối xứng.**

Hay còn gọi là mã hóa không công khai, đây là những hệ mật được sử dụng chung 1 khóa trong quá trình mã hóa và mã hóa. Do đó khóa phải được giữ bí mật tuyện đối. Một số ví dụ các thuật toán đối xứng nổi tiếng và khá được tôn trọng bao gồm Twofish, Serpent, AES (còn được gọi là Rijndael), Blowfish, CAST5, RC4, Tam phần DES (Triple DES), và IDEA (International Data Encryption Algorithm – Thuật toán mật mã hóa dữ liệu quốc tế).

Thuật toán đối xứng có thể được chia ra làm hai thể loại, mật mã luồng (stream ciphers) và mật mã khối (block ciphers). Mật mã luồng mã hóa từng bit của thông điệp trong khi mật mã khối gộp một số bit lại và mật mã hóa chúng như một đơn vị. Cỡ khối được dùng thường là các khối 64 bit.

Các thuật toán đối xứng thường không được sử dụng độc lập. Trong thiết kế của các hệ thống mật mã hiện đại, cả hai thuật toán bất đối xứng (asymmetric) (dùng chìa khóa công khai) và thuật toán đối xứng được sử dụng phối hợp để tận dụng các ưu điểm của cả hai. Các thuật toán chìa khóa bất đối xứng được sử dụng để phân phối chìa khóa mật cho thuật toán đối xứng có tốc độ cao hơn.

* + 1. **Tốc độ xử lý.**

Các thuật toán đối xứng nói chung đòi hỏi công suất tính toán ít hơn các thuật toán khóa bất đối xứng (asymmetric key algorithms). Trên thực tế, một thuật toán khóa bất đối xứng có khối lượng tính toán nhiều hơn gấp hằng trăm, hằng ngàn lần một thuật toán khóa đối xứng (symmetric key algorithm) có chất lượng tương đương.

* + 1. **Hạn chế.**

Hạn chế của các thuật toán khóa đối xứng bắt nguồn từ yêu cầu về sự phân hưởng chìa khóa bí mật, mỗi bên phải có một bản sao của chìa. Do khả năng các chìa khóa có thể bị phát hiện bởi đối thủ mật mã, chúng thường phải được bảo an trong khi phân phối và trong khi dùng. Hậu quả của yêu cầu về việc lựa chọn, phân phối và lưu trữ các chìa khóa một cách không có lỗi, không bị mất mát là một việc làm khó khăn, khó có thể đạt được một cách đáng tin cậy.

Hiện nay người ta phổ biến dùng các thuật toán bất đối xứng có tốc độ chậm hơn để phân phối chìa khóa đối xứng khi một phiên giao dịch bắt đầu, sau đó các thuật toán khóa đối xứng tiếp quản phần còn lại. Vấn đề về bảo quản sự phân phối chìa khóa một cách đáng tin cậy cũng tồn tại ở tầng đối xứng, song ở một điểm nào đấy, người ta có thể kiểm soát chúng dễ dàng hơn. Tuy thế, các khóa đối xứng hầu như đều được sinh tạo tại chỗ.

Các thuật toán khóa đối xứng không thể dùng cho mục đích xác thực (authentication) hay mục đích chống thoái thác (non-repudiation) được.

**Chương III: Cơ sở lý thuyết và hệ mã AES**

* 1. **Giới Thiệu**

AES được NIST công bố năm 2001. AES là mật mã khối đối xứng nhằm thay thế DES như là tiêu chuẩn được chấp nhận cho nhiều ứng dụng thương mại. AES sử dụng kích thước khối 128-bit và kích thước khóa 128, 192 hoặc 256-bit.

Trong AES, tất cả các hoạt động được thực hiện trên các byte 8-bit. Các phép toán số học gồm phép cộng, nhân và chia được thực hiện trong trường hữu hạn GF (28).

* 1. **Các khái niệm và định nghĩa:**
     1. **Các khái niệm và ký hiệu**

|  |  |
| --- | --- |
| **Ký hiệu** | **Ý nghĩa** |
| AES | Chuẩn mã hoá nâng cao |
| Biến đổi Affine | Phép biến đổi bao gồm một phép nhân với một ma trận sau đó là một phép cộng của một vectơ |
| Bit | Một số nhị phân nhận giá trị 0 hoặc 1 |
| Block | Một dãy các bit nhị phân tạo thành input, output, trạng thái (state) và các khóa sử dụng tại các vòng lặp (Round Key) của hệ mã. Độ dài của dãy (khối) là số lượng các bit mà nó chứa. Các khối cũng có thể được xem là một dãy các byte |
| Byte | Một nhóm 8 bit |
| Cipher | Mã Hoá |
| Cipher Key | Khoá của hệ mã, có thể được biểu diễn dưới dạng một mảng 2 chiều gồm 4 hàng và Nk cột |
| Ciphertext | Bản mã |
| Inverse Cipher | Giải mã |
| Round Key | Là các giá trị sinh ra từ khóa chính bằng cách sử dụng thủ tục sinh khóa. Các khóa này được sử dụng tại các vòng lặp của thuật toán |
| State | Mảng trạng thái, là các giá trị mã hóa trung gian có thể biểu diễn dưới dạng một mảng 2 chiều gồm 4 hàng và Nb cột |
| S-box | Một bảng thế phi tuyến được sử dụng trong thủ tục sinh khóa và trong các biến đổi thay thế các byte để thực hiện các thay thế 1-1 đối với một giá trị 1 byte |
| Word | Một nhóm 32-bit có thể được xem như 1 đơn vị tính toán độc lập hoặc là một mảng 4 byte |

**Bảng 3.1. Các khái niệm và định nghĩa được sử dụng để trình bày về**

**chuẩn mã hoá nâng cao**

* + 1. **Các hàm, thủ tục, ký hiệu và các tham số của thuật toán**

|  |  |
| --- | --- |
| **Ký hiệu** | **Ý nghĩa** |
| AddRoundKey () | Thủ tục này được sử dụng trong thuật toán mã hóa và giải mã, trong đó thực hiện phép toán XOR bit giữa một trạng thái trung gian (State) và một khóa của vòng (Round Key). Kích thước của một Round Key bằng kích thước của trạng thái. |
| InvMixColumns () | Hàm biến đổi được sử dụng trong thuật toán giải mã, là hàm ngược của hàm MixColumns() |
| InvShiftRows () | Hàm biến đổi trong thuật toán giải mã, là hàm ngược của hàm ShiftRows () |
| InvSubBytes () | Hàm biến đổi trong thuật toán giải mã, là hàm ngược của hàm SubBytes () |
| K | Khóa mã hóa |
| MixColumns () | Hàm biến đổi trong thuật toán mã hóa nhận tất cả các cột của một trạng thái (State) và trộn với dữ liệu của nó (không phụ thuộc lẫn nhau) để nhận được một cột mới |
| Nb | Số lượng các cột (là các word 32 bit) tạo thành một trạng thái, Nb = 4) |
| Nk | Số lượng các word 32-bit tạo thành khóa mã hóa K (Nk = 4, 6, hoặc 8) |
| Nr | Số lượng các vòng lặp của thuật toán, là một hàm của Nk và Nb (là các giá trị cố định) (Nr = 10, 12 hoặc 14 tương ứng với các giá trị khác nhau của Nk) |
| Rcon [] | Mảng word hằng số sử dụng trong các vòng lặp |
| RotWord () | Hàm sử dụng trong thủ tục sinh khóa nhận một word 4-byte và thực hiện một hoán vị vòng |
| ShiftRows () | Hàm sử dụng trong quá trình mã hóa, xử lý các trạng thái bằng cách dịch vòng ba hàng cuối của trạng thái với số lần dịch khác nhau |
| SubBytes () | Hàm biến đổi sử dụng trong quá trình mã hóa, xử lý một trạng thái bằng cách sử dụng một bảng thế phi tuyến các byte (S-box) thao tác trên mỗi byte một cách độc lập |
| SubWord () | Hàm sử dụng trong thủ tục sinh khóa nhận một word input 4-byte và sử dụng một S-box trên mỗi giá trị 4-byte này để thu được 1 word output |
| XOR | Phép or bit tuyệt đối |
| ⊕ | Phép or bit tuyệt đối |
| ⮾ | Phép nhân trên trường hữu hạn |

**Bảng 3.2. Các tham số thuật toán, các ký hiệu và các hàm được sử dụng trong**

**mô tả thuật toán**

* + 1. **Các khái niệm và quy ước**
       1. **Input và Output**

Input và Output là các dữ liệu lối vào và lối ra của chuẩn mã hóa nâng cao, chúng đều là các dãy 128-bit, còn gọi là các khối (block). Khóa mã của chuẩn mã hóa nâng cao là một dãy có độ dài 128, 192 hoặc 256-bit. Chuẩn mã hóa nâng cao không làm việc với các giá trị input, output và khóa có độ dài khác. Chuỗi các bit của input, output và khóa của hệ mã được đánh chỉ số bắt đầu từ 0 và kết thúc bằng một giá trị nhỏ hơn chiều dài của chuỗi được sử dụng. Ví dụ i là một chỉ số bit thì 0 ≤ i <128, 0 ≤ i < 192 hoặc 0 ≤ i < 256 phụ thuộc vào chiều dài của khối hoặc chiều dài của Khóa mã.

* + - 1. **Đơn vị Byte**

Đơn vị cơ bản dùng để xử lý trong thuật toán AES là một byte. Byte là một dãy 8-bit được phân tách và sử dụng như một thực thể đơn. Các bit của lối vào, lối ra và Khóa trình bày trong phần 2.3.1 được xử lý như mảng của các bytes, bằng việc chia các bit thành nhóm 8-bit liên tục nhau để tạo thành mảng của các byte xem **phần Input và Output**. Các Byte có thể được biểu diễn dưới dạng các bit nhị phân theo thứ tự *{b7, b6, b5, b4, b3, b2, b1, b0}* hoặc biểu diễn trên trường hữu hạn bằng đa thức hoặc bằng 2 ký tự trong hệ Hexa.

**Ví dụ**

|  |  |  |  |
| --- | --- | --- | --- |
| Biễu diễn dạng đa thức | (x6+x4+x2+x+1) | (x7+x+1) | (x7+x6+x4+x2) |
| Biểu diễn dạng nhị phân | {01010111} | {10000011} | {11010100} |
| Biểu diễn dạng Hexa | {57} | {83} | {D4} |

**Bảng 3.3. Minh hoạ đơn vị Byte**

* + - 1. **Mảng Byte**

128-bit lối vào được đánh chỉ số là input0input1input2… input126input127 được chia thành một mảng các byte a0= {input0input1 … input6input7}; a1= {input8input9 … input15}; … a15= {input120input121 … input126input127}; hay an= {input8ninput8n+1 … input8n+7}; Các byte được biểu diễn thành một mảng dưới dạng sau: *a0 a1 a2 … an-1* (với n=16, 24 hay 32 tùy thuộc vào độ dài của dữ liệu đầu vào là 128-bit, 192-bit hay 256-bit).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | … |
| Byte | 0 | | | | | | | | 1 | | | | | | | | 2 | | | | | | | | … |
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | … |

**Bảng 3.4. Minh hoạ Mảng Byte**

* + - 1. **Mảng trạng thái (States)**

Mảng trạng thái là một mảng hai chiều gồm 4 hàng, Nb cột, ký hiệu là S được dùng để lưu trữ giá trị trung gian trong mỗi bước của quá trình xử lý.

Bắt đầu của phép mã hoá hay giải mã là việc sao chép mảng các byte in0, in1, …, in15 đầu vào vào mảng trạng thái S theo công thức sau:

*S [r, c] = in[r+4c], với 0 ≤ r, c ≤ 4*

Cuối quá trình mã hoá hay giải mã, mảng trạng thái sẽ được sao chép vào mảng byte đầu ra: *out0, out1, out2, …, out15*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Input byte* | | | | **=>** | *State array* | | | | **=>** | *output byte* | | | |
| *in0* | *in4* | *in8* | *in12* | *S0,0* | *S0,1* | *S0,2* | *S0,3* | *out0* | *out4* | *out8* | *out12* |
| *in1* | *in5* | *in9* | *in13* | *S1,0* | *S1,1* | *S1,2* | *S1,3* | *out1* | *out5* | *out9* | *out13* |
| *in2* | *in6* | *in10* | *in14* | *S2,0* | *S2,1* | *S2,2* | *S2,3* | *out2* | *out6* | *out10* | *out14* |
| *in3* | *in7* | *in11* | *in15* | *S3,0* | *S3,1* | *S3,2* | *S3,3* | *out3* | *out7* | *out11* | *out15* |

**Bảng 3.5. Minh hoạ Mảng trạng thái (States)**

* + - 1. **Mảng các từ**

Bốn byte trên mỗi cột của mảng trạng thái được tạo thành một từ kép 32 bit do đó mảng trạng thái có thể mô tả là mảng một chiều, với mỗi phần tử gồm 4-byte được đánh chỉ mục theo hàng r. Các phần tử của mạng ký hiệu là *w0, w1, w2, w3*.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Mảng trạng thái* | | | | **=>** | *Mảng trạng thái* | | | | Với | *Các phần tử của mảng* |
| *S0,0* | *S0,1* | *S0,2* | *S0,3* | *(w0,* | *w1,* | *w2,* | *w3)* | *w0= (S0,0, S1,0, S2,0, S3,0)* |
| *S1,0* | *S1,1* | *S1,2* | *S1,3* | *w1 = (S0,1, S1,1, S2,1, S3,1)* |
| *S2,0* | *S2,1* | *S2,2* | *S2,3* | *w2= (S0,2, S1,2, S2,2, S3,2)* |
| *S3,0* | *S3,1* | *S3,2* | *S3,3* | *w3= (S0,3, S1,3, S2,3, S3,3)* |

**Bảng 3.6. Minh hoạ Mảng các từ**

* + 1. **Cơ sở toán học**

Các byte trong thuật toán AES đều được biểu diễn như một phần tử trong trường hữu hạn sử dụng các ký hiệu như trình bày ở phần 2.3.2. Các phần tử trong trường hữu hạn đều có thể được cộng và nhân. Tuy nhiên các phép toán này khác hẳn với các phép cộng và nhân thông thường. Hai phép cộng và phép nhân trên trường Galoris GF (28) là cở sở toán học của thuật toán AES.

#### **3.2.4.1. Phép cộng**

Phép cộng hai phần tử trong trường hữu hạn được thực hiện bằng việc “Cộng” các hệ số có cùng trọng số trong các đa thức biểu diễn các phần tử. Việc cộng được thực hiện thông qua phép toán XOR ký hiệu là ⊕

|  |  |  |
| --- | --- | --- |
| *A* | *B* | *A⊕B* |
| *0* | *0* | *0* |
| *0* | *1* | *1* |
| *1* | *0* | *1* |
| *1* | *1* | *0* |

**Bảng 3.7. Phép cộng XOR**

Như vậy việc cộng các phần tử trong trường hữu hạn có thể mô tả bằng việc cộng modulo 2 của các bit tương ứng trong byte. Cho hai byte a7a6a5a4a3a2a1a0} và {b7b6b5b4b3b2b1b0}, thì tổng hai byte là {c7c6c5c4c3c2c1c0}, trong đó ci = ai ⊕ bi với i là vị trí các bit tương ứng.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Biễu diễn dạng đa thức | (x6+x4+x2+x+1) | ⊕ | (x7+x+1) | = | (x7+x6+x4+x2) |
| Biểu diễn dạng nhị phân | {01010111} | ⊕ | {10000011} | = | {11010100} |
| Biểu diễn dạng Hexa | {57} | ⊕ | {83} | = | {D4} |

**Bảng 3.8. Biểu diễn phép cộng XOR**

* + - 1. **Phép nhân**

Phép nhân trên trường Galois GF (28), ký hiệu là ⮾ tương ứng với phép nhân thông thường của hai đa thức rồi đem chia lấy dư (modulo) cho một đa thức tối giản bậc 8 (đa thức tối giản là đa thức chỉ chia hết cho 1 và chính nó). Trong thuật toán AES, đa thức tối giản được chọn là: m(x) = x8 + x4 + x3 + x + 1 hay biểu diễn dưới dạng hexa là {01} {1b}

Ví dụ: {57}⮾{83} = {c1}

Vì (x6 + x4 + x2 + x + 1) (x7 + x + 1) = x13 + x11 + x9+ x8 + x7 +

x7 + x5+x3+x2+x+

x6 + x4 + x2 + x + 1

= x13 + x11 + x9+ x6 + x5 + x4 + x3 + 1

x13 + x11 + x9+ x6 + x5 + x4 + x3 + 1 chia cho x8 + x4 + x3 + x + 1 dư **x7 + x6 + 1**

x7 + x6 + 1 biểu diễn nhị phân là: 11000001 hay c1 trong biễu diễn hexa.

Việc chia cho đa thức bậc 8 m(x) bảo đảm kết quả nhận được của phép rút gọn là một đa thức có bậc nhỏ hơn 8 nên có thể biểu diễn được dưới dạng 1 byte.

* + - 1. **Phép nhân với x**

Phép nhân với đa thức x (hay phần tử {00000010} € GF (28) có thể được thực hiện ở mức byte bằng một phép dịch trái và sau đó thực hiện tiếp phép XOR với giá trị {01}{1b} nếu bit b7=1. Thủ tục nhân với x ký hiệu là *xtime ()*. Phép nhân với các luỹ thừa của x có thể được thực hiện bằng cách áp dụng nhiều lần thủ tục *xtime ().* Kết quả phép nhân với một giá trị bất kỳ được xác định bằng phép cộng (⊕) các kết quả trung gian này lại với nhau.

Ví dụ:

Phép nhân {a2} ⮾ {03}

Ta có {03} = {02} ⊕ {01}

Vậy {a2} ⮾ {03} = {a2} ⮾ {02} ⊕ {a2} ⮾ {01}

{a2} ⮾ {01} = {a2} //1010 0010

{a2} ⮾ {02} = {01} {44} ⊕ {01} {1b} = {5F}

Kết quả {a2} ⮾ {03} = {a2} ⊕ {5F} = {FD}

# **Chương IV: Thuật toán AES**

* 1. Table

     Description automatically generated**Cấu trúc thuật toán AES**

**Bảng 4.1. Tổng quan thuật toán AES**

Đầu tiên, chúng ta có khoá đầu vào là **Khoá K** có M bit với ba giá trị là (128-bit, 192-bit và 256-bit). **Input** là dữ liệu cần được mã hoá, dữ liệu nhập vào sẽ ở 128-bit.

Để bắt đầu mã hoá, chúng ta sẽ phải thực hiện qua bước biến đổi ban đầu, sau đó từ Vòng 1 đến Vòng N-1 sẽ bao gồm đầy đủ 4 thủ tục: SubBytes, ShiftRows, MixColumns và AddRoundkey. Ngoại trừ vòng lặp cuối là Vòng N sẽ không có thủ tục MixColumns.

Thuật toán mã hoá này có thể thực hiện được cả bằng phần cứng và phần mềm.

Thuật toán AES được thực hiện tuần tự gồm nhiều bước biến đổi, kết quả đầu ra của phép biến đổi trước là đầu vào của phép biến đổi tiếp theo. Kết quả trung gian của các phép biến đổi chính là mảng trạng thái (state)

Độ dài của khối dữ liệu đầu vào của AES là cố định với Nb=4. Tùy vào độ dài khóa Nk (Nk=4, 6, 8) ta có số lần lặp Nr (Nr =10, 12, 14).

* 1. **Chi tiết một vòng lặp (Từ vòng 1 đến N-1)**

Một vòng lặp đầy đủ sẽ bao gồm 4 thủ tục:

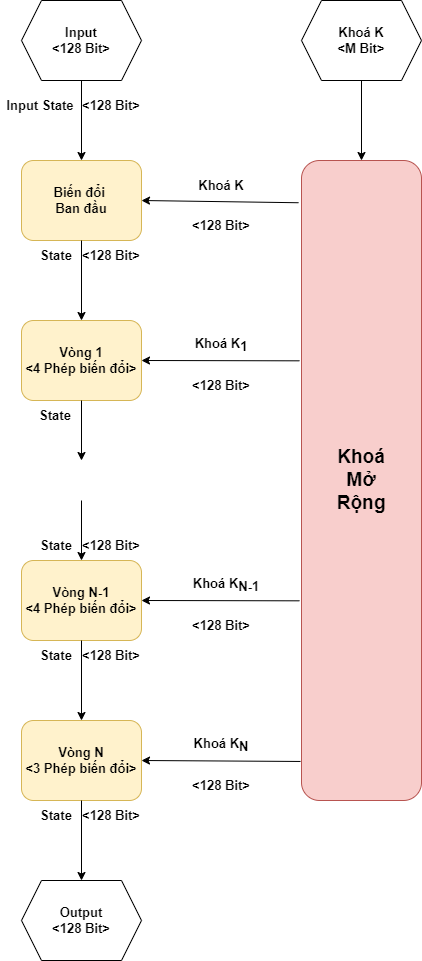
**> SubstituteBytes**

**> ShiftRows**

**> MixColumns**

**> AddRoundKey**

**Lưu ý:** Tại vòng lặp thứ N không có phép **MixColumns.**

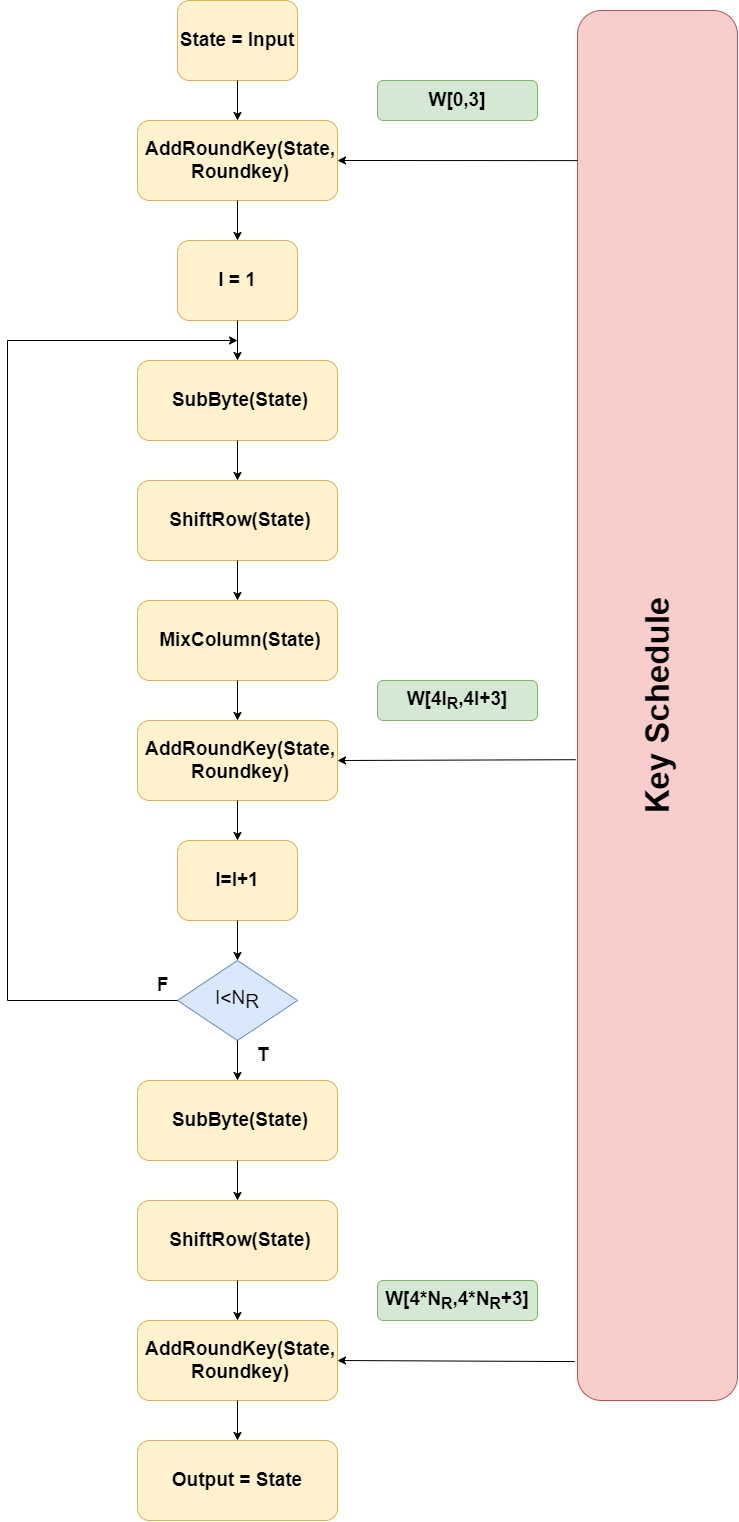


**Hình 4.1 Cấu trúc thuật toán AES**

* 1. **Thuật toán mã hoá**
     1. **Sơ đồ khối các bước thực hiện**

Bắt đầu thuật toán, dữ liệu cần mã hóa (Input) được sao chép vào mảng trạng thái (State) rồi XOR với khóa ban đầu (khóa khởi tạo). Thuật toán AES được thực hiện Nr vòng (Nr = 10, 12, 14 tùy theo độ dài khóa sử dụng là 128-bit, 192-bit, 256-bit). Nr - 1 vòng đầu tiên có 4 thủ tục được thực hiện lần lượt là Subbyte (), ShiftRow(), Mixcolumn() và AddRoundKey(), trong khi vòng cuối cùng chỉ có 3 thủ tục được thực hiện là Subbyte(), ShiftRow() và AddRoundKey(). Ở vòng cuối thủ tục Mixcolumn () không được thực hiện. Bước cuối cùng của thuật toán mã hóa là dữ liệu được đưa ra lối ra Output.

Như vậy thủ tục AddRoundKey () được thực hiện Nr + 1 lần, Mixcolumn () được thực hiện Nr - 1 lần, các thủ tục khác được thực hiện Nr lần.



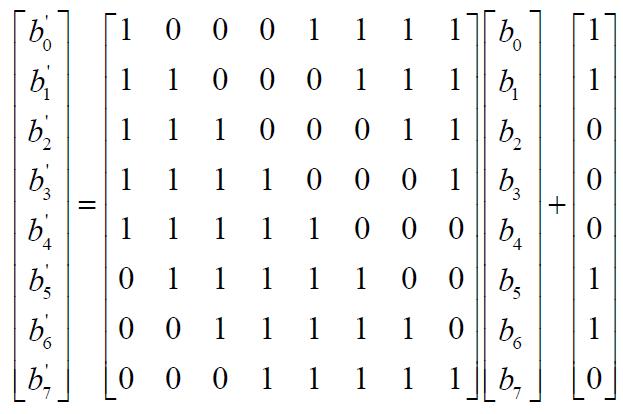
**Hình 4.2. Lưu đồ thuật toán AES**

* + 1. **Chi tiết các bước mã hoá**
       1. **Thủ tục SubBytes ()**

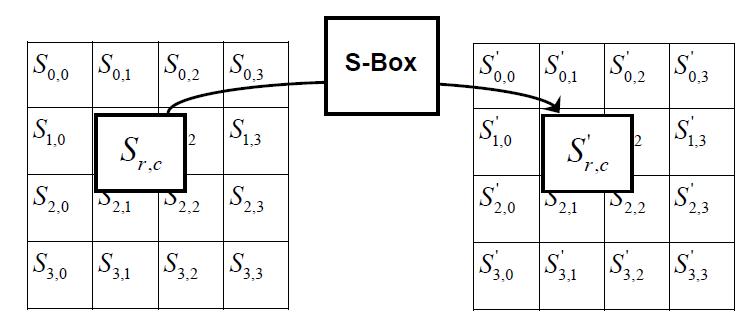
SubBytes () thực hiện phép thay thế các byte của mảng trạng thái bằng cách sử dụng một bảng thế S-box, bảng thế này là khả nghịch và được xây dựng bằng cách kết hợp hai biến đổi sau:

* Nhân nghịch đảo trên trường hữu hạn GF (28) (phần tử {00} được ánh xạ đến chính nó).
* Áp dụng biến đổi sau:

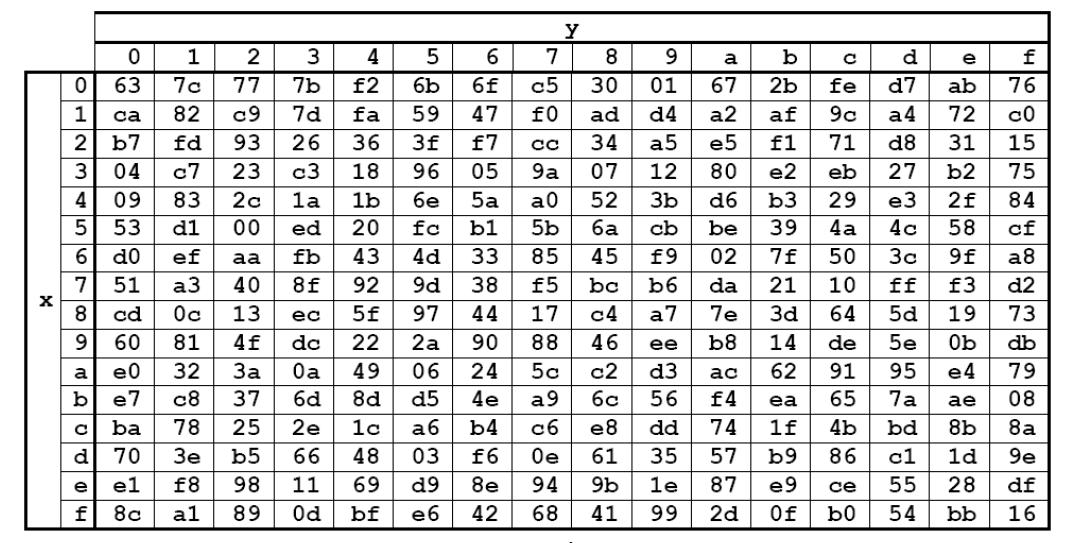
b’i = bi⊕ b(i+4) mod 8⊕ b(i+5) mod 8⊕ b(i+6) mod 8⊕ b(i+7) mod 8⊕ ci trong đó 0 <= I< 8 là bit thứ i của byte b tương ứng và ci là bit thứ i của byte c, byte c có giá trị {63} hay {01100011}.

Các phần tử biến đổi của S-box có thể được biểu diễn dưới dạng ma trận như sau:

**Hình 4.3. Ma trận Biến đổi của các phần tử S-Box**

Hình sau minh họa kết quả của việc áp dụng hàm biến đổi SubBytes () đối với mảng trạng thái:

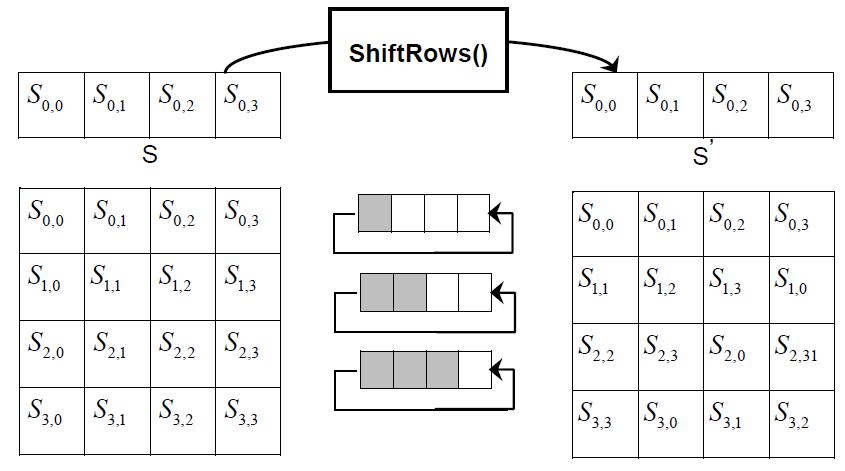
**Hình 4.4. Biển đổi SubBytes**

Bảng thế S-box được sử dụng trong hàm SubBytes () được cho như một hằng số và được biểu diễn dưới dạng hexa như sau:

**Bảng 4.2 Bảng thế S-Box của AES**

Ví dụ: Nếu S1,1 = {53} có nghĩa là giá trị thay thế sẽ được xác định bằng giao của hàng có chỉ số 5 với cột có chỉ số 3 trong bảng trên, kết quả cho S’1,1 = {ed}.

* + - 1. **Thủ tục ShiftRows ()**

Thủ tục này thực hiện việc dịch vòng của các hàng trong mảng trạng thái, cụ thể như sau: Hàng đầu tiên không bị dịch (giữ nguyên), hàng thứ 2 dịch vòng một byte, hàng thứ 3 dịch vòng 2 byte, hàng cuối cùng dịch vòng 3 byte. Hình minh họa dưới đây:

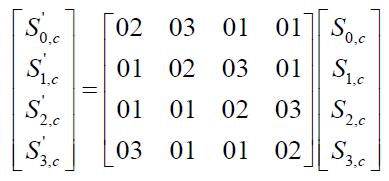
**Hình 4.5. Cách thức hoạt động Thủ tục ShiftRows**

* + - 1. **Thủ tục MixColumn ()**

Thủ tục này làm việc trên các cột của bảng trạng thái, nó coi mỗi cột của mảng trạng thái như là một đa thức gồm 4 hạng tử. Các cột sẽ được xem như là các đa thức trên GF (28) và được nhân theo modulo x4 + 1 với một đa thức cố định a(x):

a(x) = {03}x3 + {01}x2 + {01}x + {02}

Điều này có thể biểu diễn bằng một phép nhân ma trận:

S’(x) = a(x) ●S(x)

với 0 <= c < Nb = 4.

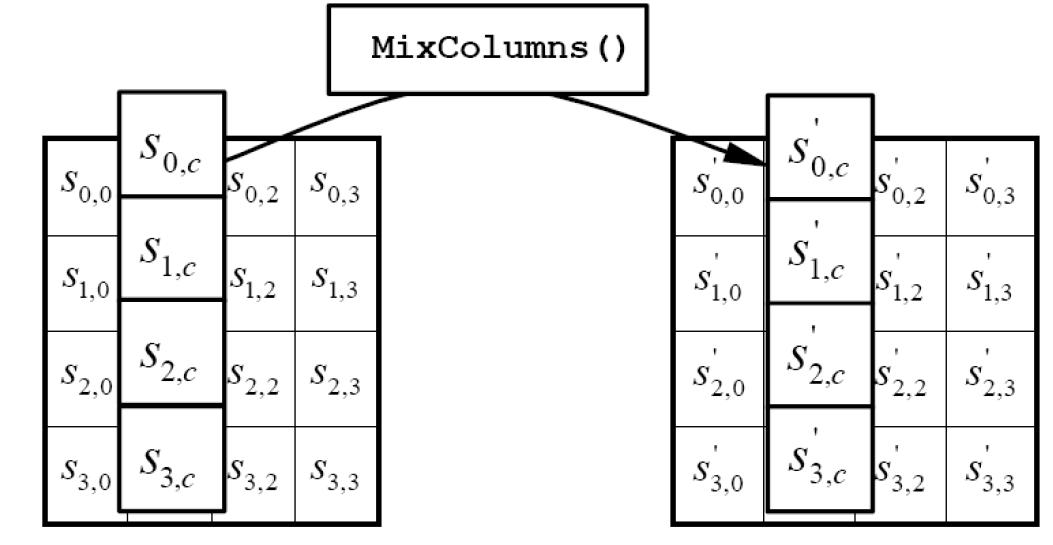
Kết quả là bốn byte trong mỗi cột sẽ được thay thế theo công thức sau:

S’0, c = ( {02}⮾ S0,c ) ⊕ ( {03} ⮾ S1,c) ⊕ S2,c⊕ S3,c

S’1, c = S0,c ⊕ ( {02}⮾ S1,c ) ⊕ ( {03} ⮾ S2,c ) ⊕ S3,c

S’2, c = S0,c ⊕ S1,c⊕ ( {02}⮾ S2,c ) ⊕ ( {03} ⮾ S3,c )

S’3, c = ({03}⮾ S0,c ) ⊕ S1,c⊕ S2,c⊕ ( {02} ⮾ S3,c )

Có thể minh họa việc thực hiện của hàm này bằng hình vẽ sau:

**Hình 4.6. Cách thức hoạt động của Thủ tục MixColumns**

* + - 1. **Thủ tục AddRoundKey ()**

Trong thủ tục này một khóa vòng (Round Key) sẽ được cộng vào mảng trạng thái bằng một thao tác XOR bit. Mỗi khóa vòng gồm Nb word được sinh ra bởi thủ tục sinh khóa. Các word này sẽ được cộng vào mỗi cột của mảng trạng thái như sau:

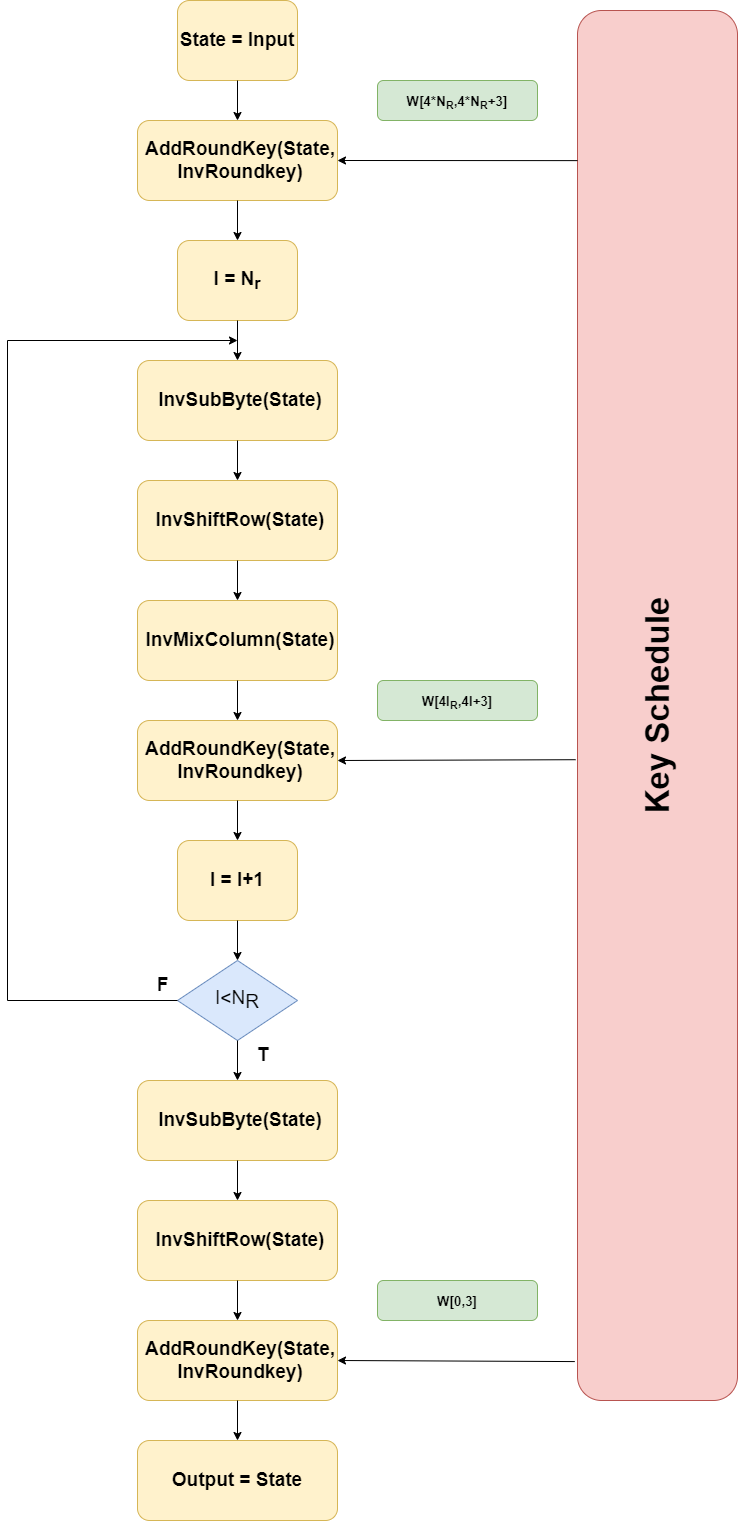
[S’0, c, S’1,c, S’2,c, S’3,c] = [S’0,c, S’1,c, S’2,c, S’3,c] ⊕ [Wround \* Nb + c ] ( với 0 ≤ c < Nb =4). Trong đó [wi] là các word của khóa và round là lần lặp tương ứng với 0 ≤ round ≤ Nr. Ngay trước khi vào vòng lặp khóa

Trong thuật toán mã hóa phép cộng khóa vòng khởi tạo xảy ra với round = 0 trước khi các vòng lặp của thuật toán được thực hiện. Hàm AddRoundKey () được thực hiện trong thuật toán mã hóa khi 1 ≤ round ≤ Nr.

Việc thực hiện của hàm này có thể minh họa qua hình vẽ, trong đó l = round \* Nb.

**Hình 4.7. Cách thức hoạt động của Thủ tục AddRoundKey**

* 1. **Thuật toán giải mã**
     1. **Sơ đồ khối các bước thực hiện**

Thuật toán giải mã của AES khá giống với thuật toán mã hóa về mặt cấu trúc, tuy nhiên các thủ tục sử dụng là ngược với các thủ tục trong thuật toán mã hóa. Sau đây là lưu đồ thuật toán và đoạn chương trình trình bày về cấu trúc của thuật toán giải mã:

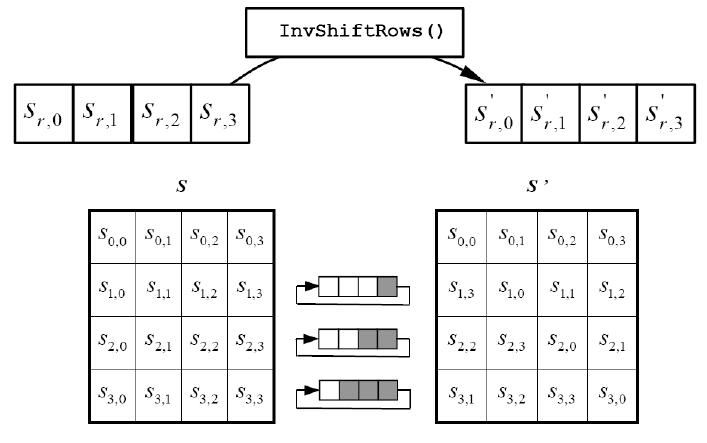
**Hình 4.8. Lưu đồ thuật toán giải mã AES**

* + 1. **Chi tiết các bước thực hiện**
       1. **Thủ tục InvShiftRows ()**

Trong thủ tục này các byte của ba hàng cuối của mảng trạng thái sẽ được dịch vòng với các vị trí dịch khác nhau. Cụ thể thủ tục này tiến hành như sau:

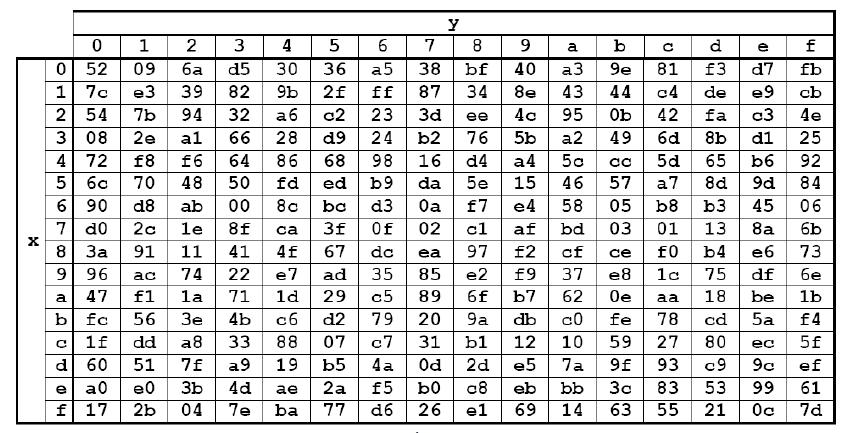
S’r, (c + shift (r, Nb)) mod Nb = Sr, c (với 0 <r <4, 0 ≤ c < Nb)

Hình minh hoạ:

* + - 1. **Thủ tục InvSubBytes ()**

**Hình 4.6. Cách thức hoạt động của Thủ tục InvShiftRows**

Tương tự như thủ tục SubByte (), thủ tục InvSubByte () sẽ tham chiếu đến một bảng gọi là InvS-Box để thay các giá trị byte tương ứng với chỉ số hang và cột của byte vào.

Bảng thế được sử dụng trong hàm là:

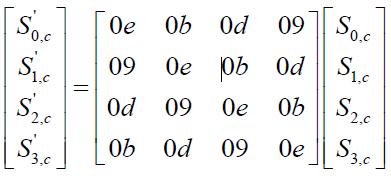
**Bảng 4.3. Bảng Inv S-Box**

* + - 1. **Thủ tục InvMixColumns ()**

Thủ tục này làm việc trên các cột của mảng trạng thái, coi mỗi cột như là một đa thức 4 hạng tử. Các cột được xem là các đa thức trên GF (28) và được nhân theo modulo x4 +1 với mộtđa thức cố đinh là a-1(x):

a-1(x) = {0b}x3 + {0d}x2 + {09}x + {0e}

Và có thể mô tả bằng phép nhân ma trận như sau:

S’(x) = a-1(x)⮾ S(x):

Trong đó 0≤ c < Nb.

Kết quả là bốn byte trong mỗi cột sẽ được thay thế theo công thức sau:

S’0, c = ({0e}⮾ S0, c) ⊕ ({0b}⮾ S1, c) ⊕({0d}⮾ S2, c) ⊕ ({09}⮾ S3, c)

S’1, c = ({09}⮾ S0,c ) ⊕({0e}⮾ S1,c) ⊕({0b}⮾ S2,c) ⊕({0d}⮾ S3,c)

S’2, c = ({0d}⮾ S0, c) ⊕({09}⮾ S1,c) ⊕({0e}⮾ S2,c) ⊕({0b}⮾ S3,c)

S’3, c = ({0b}⮾ S0,c ) ⊕({0d}⮾ S1,c) ⊕({09}⮾ S2,c) ⊕({0e}⮾ S3,c)

* + - 1. **Thủ tục AddRoundKey ()**

Do thủ tục này chỉ thực hiện hàm XOR nên bản thân nó cũng đã là đảo của chính nó.

* + - 1. **Thuật toán giải mã tương đương**

Trong thuật toán giải mã được trình bày ở trên chúng ta thấy thứ tự của các hàm biến đổi được áp dụng khác so với thuật toán mã hóa trong khi dạng của danh sách khóa cho cả 2 thuật toán vẫn giữ nguyên. Tuy vậy một số đặc điểm của AES cho phép chúng ta có một thuật toán giải mã tương đươg có thứ tự áp dụng các hàm biến đổi giống với thuật toán mã hóa (tất nhiên là thay các biến đổi bằng các hàm ngược của chúng). Điều này đạt được bằng cách thay đổi danh sách khóa.

Hai thuộc tính sau cho phép chúng ta có một thuật toán giải mã tương đương:

* Các thủ tục SubBytes () và ShiftRows() hoán đổi cho nhau; có nghĩa là một biến đổi SubBytes() theo sau bởi một biến đổi ShiftRows() tương đương với mộtbiến đổi ShiftRows() theo sau bởi một biến đổi SubBytes(). Điều này cũng đúng với các hàm ngược của chúng.
* Các hàm trộn cột – MixColumns () và InvMixColumns() là các hàm tuyến tính đối với các cột input, có nghĩa là:

InvMixColumns (state XOR Round Key) = InvMixColumns(state) XOR InvMixColumns (Round Key).

Các đặc điểm này cho phép thứ tự của các hàm InvSubBytes () và InvShiftRows() có thể đổi chỗ. Thư tự của các hàm AddRoundKey () và InvMixColumns() cũng có thể đổi chỗ miễn là các cột của danh sách khóa giải mã phải được thay đổi bằng cách sử dụng hàm InvMixColumns ().

Thuật toán giải mã tương đương được thực hiện bằ ng cách đảo ngược thứ tự của hàm InvSubBytes () và InvShiftRows (), và thay đổi thứ tự của AddRoundKey () và InvMixColumns () trong các lần lặp sau khi thay đổi khóa cho giá tri ̣round = 1 to Nr-1 bằng cách sử dụng biến đổi InvMixColumns (). Các word đầu tiên và cuối cùng của danh sách khóa không bị thay đổi khi ta áp dụng phương pháp này.

Thuật toán giải mã tương đương cho một cấu trúc hiệu quả hơn so với thuật toán giải mã trước đó.

Đoạn giả mã cho thuật toán giải mã tương đương:

EqInvCipher (byte in[4\*Nb], byte out[4\*Nb], word dw [Nb\*(Nr+1)])

* + - 1. **Thuật toán mở rộng khóa (Key Expansion)**

Việc mã hóa và giải mã của AES cần Nr + 1 lần thực hiện hàm AddRoundKey (), đối với lần đầu tiên khóa nhận được từ lối vào, do đó ta cần Nr lần sinh khóa mới. Khóa mới được sinh ra dựa trên các dữ liệu như: khóa trước đó, một hằng số chính là số thứ tự của vòng lặp và các giá trị truy vấn từ bảng S-BOX.

Thuật toán sinh khóa của AES nhận một khóa mã hóa K sau đó thực hiện một thủ tục sinh khóa để sinh một dãy các khóa cho việc mã hóa. Thủ tục này sẽ sinh tổng số Nb \* (Nr+1) word, thủ tục sử dụng một tập khởi tạo Nb word và mỗi một lần lặp trong số Nr lần sẽ cần tới Nb word của dữ liệu khóa. Dãy khóa kết quả là một mảng tuyến tính các word 4-byte được ký hiệu là [wi] trong đó 0 ≤ i < Nb (Nr + 1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w0 | w1 | w2 | w3 | w4 | w5 | w6 | w7 | …. | wNb\*Nr | wNb\*Nr+1 | wNb\*Nr+2 | wNb\*Nr+3 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

Sự mở rộng khóa được mô tả qua đoạn mã sau:

KeyExpansion (byte key[4\*Nk], word w[Nb\*(Nr+1)], Nk)

**Chương 5: MÔ PHỎNG THUẬT TOÁN**

**KẾT LUẬN VÀ ĐÁNH GIÁ**

**5.1. Ứng dụng của thuật toán AES**

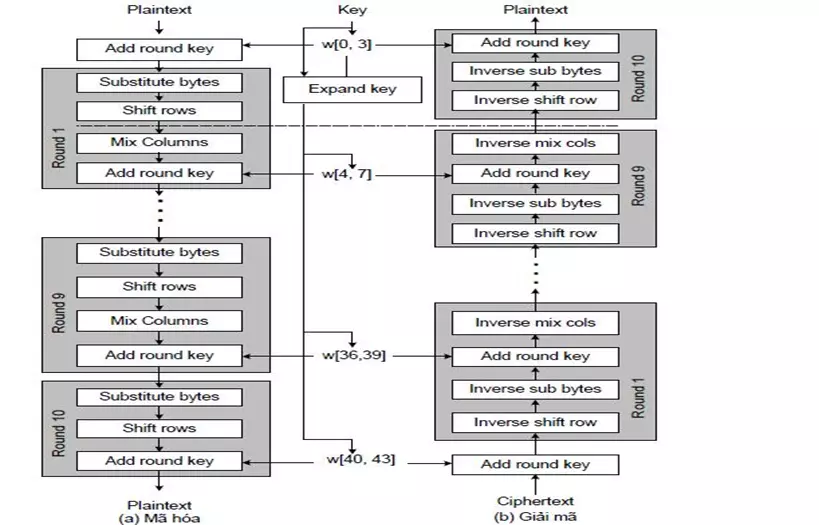
Như đã trình bày AES chỉ xử lý dữ liệu lối vào và lối ra là 128-bit tương ứng với 16-byte dữ liệu. Nếu ta sử dụng AES để mã hóa ký tự ASCII chuẩn thì mỗi lần ta có thể xử lý được 16 ký tự. Tuy nhiên đối với ký tự là tiếng Việt, thì mỗi ký tự có mã 2 byte do vậy ta chỉ có thể xử lý được 8 ký tự. Điều này có nghĩa là lối vào của mã hóa AES là 8 ký tự Unicode.

Chương trình mã hoá và giải mã AES được nhóm viết dưới ngôn ngữ C++, được biên dịch và chạy trên cửa sổ CMD.

Hàm SubWord () là một hàm nhận một input 4-byte và áp dụng bảng thế S-box lên input để nhận được một word output.

Hàm RotWord () nhận một word input [a0, a1, a2, a3] thực hiện một hoán vị vòng và trả về [a1, a2, a3, a0].

Các phần tử của mảng hằng số Rcon[i] chứa các giá trị nhận được bởi [2i-1, {00}, {00}, {00}] trong đó i bắt đầu từ 1.

Chú ý là thủ tục mở rộng khóa đối với các khóa có độ dài 256 hơi khác so với thủ tục cho các khóa có độ dài 128 hoặc 192. Nếu Nk = 8 và i – 4 là một bội số của Nk thì SubWord () sẽ được áp dụng cho w[i-1] trước khi thực hiện phép XOR bit.

**Hình 5.1. Sơ đồ hoạt động của thuật toán**

* 1. **Mô phỏng thuật toán AES 128-bit**

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

*unsigned* *int* *\**cipherStringToHexaArray(*const* std**::**string *&*, *int*);

*void* showWord(*unsigned* *int*);

*void* showMatrix(*unsigned* *int*);

string matrixToStr(*unsigned* *int* *\****word**);

string wordToStr(*unsigned* *int*);

*unsigned* *int* rotWord(*unsigned* *int*);

*unsigned* *int* subWord(*unsigned* *int*);

*unsigned* *int* xorRcon(*unsigned* *int*, *int*);

*unsigned* *int* G(*unsigned* *int*, *int*);

*unsigned* *int* *\**keyExpansion(*unsigned* *int* *\**);

*unsigned* *int* *\**SubBytes(*unsigned* *int* *\**);

*unsigned* *int* *\**ShiftRows(*unsigned* *int* *\**);

*unsigned* *int* multiply\_double(*unsigned* *int*);

*unsigned* *int* multiply\_triple(*unsigned* *int*);

*unsigned* *int* multiply\_column(*unsigned* *int*);

*unsigned* *int* *\**MixColumns(*unsigned* *int* *\**);

*unsigned* *int* *\**AddRoundKey(*unsigned* *int* *\**, *unsigned* *int* *\**);

*unsigned* *int* *\**EncryptionAES(*unsigned* *int* *\**, *unsigned* *int* *\**);

*unsigned* *int* *\**InvShiftRows(*unsigned* *int* *\**);

*unsigned* *int* InvSubWord(*unsigned* *int*);

*unsigned* *int* *\**InvSubBytes(*unsigned* *int* *\**);

*unsigned* *int* multiply\_9(*unsigned* *int*);

*unsigned* *int* multiply\_B(*unsigned* *int*);

*unsigned* *int* multiply\_D(*unsigned* *int*);

*unsigned* *int* multiply\_E(*unsigned* *int*);

*unsigned* *int* InvMultiply\_column(*unsigned* *int*);

*unsigned* *int* *\**InvMixColumns(*unsigned* *int* *\**);

*unsigned* *int* *\**DecryptionAES(*unsigned* *int* *\**, *unsigned* *int* *\**);

*unsigned* *int* *\**cipherStringToHexaArray(*const* std**::**string *&***str**, *int* **length**)

{

*unsigned* *int* **\***arrayString **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0, j **=** 0; i **<** **str**.length(); i **+=** **length**, j**++**)

    {

        arrayString[j] **=** std**::**stoul(**str**.substr(i, **length**), nullptr, 16);

        cout **<<** " ";

    }

**return** arrayString;

}

*int* main()

{

    string inputString **=** "", cipherKey **=** "";

**do**

    {

        cout **<<** "Enter a string to encode (The length of the input string in AES can be 128 bits, 192 bits, or 256 bits, which is equivalent to 16 bytes, 24 bytes, or 32 bytes; If the input string is not long enough, which is 16 bytes, the system will automatically add 0x00 to make up for it):" **<<** endl;

        getline(cin, inputString);

        inputString.erase(remove(inputString.begin(), inputString.end(), ' '), inputString.end());

    } **while** (inputString.length() **!=** 32);

**do**

    {

        cout **<<** "Enter the value of the security key (The length of the Cipher Key in AES can be 128 bits, 192 bits, or 256 bits, which is equivalent to 16 bytes, 24 bytes, or 32 bytes; If the security key is not long enough, which is 16 bytes, the system will automatically add 0x00 to make up for it):" **<<** endl;

        getline(cin, cipherKey);

        cipherKey.erase(remove(cipherKey.begin(), cipherKey.end(), ' '), cipherKey.end());

**if** (cipherKey.length() **<** 32)

        {

*int* diff **=** 32 **-** cipherKey.length();

            string zeros(diff, '0');

            cipherKey **+=** zeros;

        }

    } **while** (cipherKey.length() **!=** 32);

*unsigned* *int* **\***state **=** **new** *unsigned* *int*[4];

*unsigned* *int* **\***key **=** **new** *unsigned* *int*[4];

    state **=** cipherStringToHexaArray(inputString, 2 **\*** 4);

    key **=** cipherStringToHexaArray(cipherKey, 2 **\*** 4);

    cout **<<** endl;

*unsigned* *int* **\***Cipher **=** EncryptionAES(state, key);

*unsigned* *int* **\***Decipher **=** DecryptionAES(Cipher, key);

    cout **<<** "Encrypt String: " **<<** matrixToStr(Cipher) **<<** endl;

    cout **<<** "Decrypt String: " **<<** matrixToStr(Decipher) **<<** endl;

}

string wordToStr(*unsigned* *int* **word**)

{

    string hexanString **=** "";

**for** (*int* i **=** 1; i **<=** 8; i**++**)

    {

        std**::**ostringstream hexanChar;

*unsigned* *int* hexan **=** (**word** **>>** (32 **-** i **\*** 4)) **&** 0xF;

        hexanChar **<<** std**::**hex **<<** hexan;

        hexanString **+=** hexanChar.str();

    }

**return** hexanString;

}

*void* showWord(*unsigned* *int* **word**)

{

**for** (*int* i **=** 1; i **<=** 8; i**++**)

    {

*unsigned* *int* hexan **=** (**word** **>>** (32 **-** i **\*** 4)) **&** 0xF;

        printf("*%X*", hexan);

    }

}

*unsigned* *int* rotWord(*unsigned* *int* **word**)

{

*unsigned* *int* byte\_1 **=** (**word** **>>** 24) **&** 0xFF;

*unsigned* *int* byte\_234 **=** **word** **&** 0xFFFFFF;

*unsigned* *int* rotWord **=** byte\_1 **|** (byte\_234 **<<** 8);

**return** rotWord;

}

*unsigned* *int* subWord(*unsigned* *int* **rotWord**)

{

*int* subWordArray[] **=** {

        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,

        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,

        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,

        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,

        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,

        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,

        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,

        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,

        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,

        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,

        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,

        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,

        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,

        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,

        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,

        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16};

*unsigned* *int* result **=** 0;

**for** (*int* i **=** 1; i **<=** 4; i**++**)

    {

*unsigned* *int* byte\_i **=** (**rotWord** **>>** (32 **-** i **\*** 8)) **&** 0xFF;

*unsigned* *int* subB **=** subWordArray[byte\_i];

        result **=** (result **<<** 8) **|** subB;

    }

**return** result;

}

*unsigned* *int* xorRcon(*unsigned* *int* **subWord**, *int* **j**)

{

*int* Rc[] **=** {

        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,

        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39};

*unsigned* *int* byte\_1 **=** (**subWord** **>>** 24) **&** 0xFF;

*unsigned* *int* byte\_234 **=** **subWord** **&** 0xFFFFFF;

*unsigned* *int* resultXor **=** (byte\_1 **^** Rc[**j**]) **&** 0xFF;

*unsigned* *int* result **=** (resultXor **<<** 24) **|** byte\_234;

**return** result;

}

*unsigned* *int* G(*unsigned* *int* **word**, *int* **j**)

{

*unsigned* *int* rotW **=** rotWord(**word**);

*unsigned* *int* subW **=** subWord(rotW);

*unsigned* *int* result **=** xorRcon(subW, **j**);

**return** result;

}

*unsigned* *int* *\**keyExpansion(*unsigned* *int* **arrayWord**[4])

{

*unsigned* *int* **\***word **=** **new** *unsigned* *int*[44];

    word[0] **=** **arrayWord**[0];

    word[1] **=** **arrayWord**[1];

    word[2] **=** **arrayWord**[2];

    word[3] **=** **arrayWord**[3];

**for** (*int* i **=** 4; i **<=** 44; i**++**)

    {

**if** (i **%** 4 **==** 0)

            word[i] **=** G(word[i **-** 1], i **/** 4) **^** word[i **-** 4];

**else**

            word[i] **=** word[i **-** 1] **^** word[i **-** 4];

    }

**return** word;

}

*unsigned* *int* *\**AddRoundKey(*unsigned* *int* **state**[4], *unsigned* *int* *\****arrayKey**)

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

        result[i] **=** **state**[i] **^** **arrayKey**[i];

**return** result;

}

*unsigned* *int* *\**SubBytes(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

        result[i] **=** subWord(**state**[i]);

**return** result;

}

*unsigned* *int* *\**ShiftRows(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

    {

*unsigned* *int* byte\_1 **=** **state**[i] **&** 0xFF000000;

*unsigned* *int* byte\_2 **=** **state**[(i **+** 1) **%** 4] **&** 0xFF0000;

*unsigned* *int* byte\_3 **=** **state**[(i **+** 2) **%** 4] **&** 0xFF00;

*unsigned* *int* byte\_4 **=** **state**[(i **+** 3) **%** 4] **&** 0xFF;

        result[i] **=** byte\_1 **|** byte\_2 **|** byte\_3 **|** byte\_4;

    }

**return** result;

}

*unsigned* *int* multiply\_double(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** **word** **<<** 1;

    result **>** 256 **?** result **^=** 0x11B **:** result;

    result **=** result **&** 0xFF;

**return** result;

}

*unsigned* *int* multiply\_triple(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** **word** **^** multiply\_double(**word**);

    result **=** result **&** 0xFF;

**return** result;

}

*unsigned* *int* multiply\_column(*unsigned* *int* **word**)

{

*unsigned* *int* result;

*unsigned* *int* byte\_1 **=** (**word** **>>** 24) **&** 0xFF;

*unsigned* *int* byte\_2 **=** (**word** **>>** 16) **&** 0xFF;

*unsigned* *int* byte\_3 **=** (**word** **>>** 8) **&** 0xFF;

*unsigned* *int* byte\_4 **=** **word** **&** 0xFF;

*unsigned* *int* result\_1 **=** multiply\_double(byte\_1) **^** multiply\_triple(byte\_2) **^** byte\_3 **^** byte\_4;

*unsigned* *int* result\_2 **=** byte\_1 **^** multiply\_double(byte\_2) **^** multiply\_triple(byte\_3) **^** byte\_4;

*unsigned* *int* result\_3 **=** byte\_1 **^** byte\_2 **^** multiply\_double(byte\_3) **^** multiply\_triple(byte\_4);

*unsigned* *int* result\_4 **=** multiply\_triple(byte\_1) **^** byte\_2 **^** byte\_3 **^** multiply\_double(byte\_4);

    result **=** (result\_1 **<<** 24) **|** (result\_2 **<<** 16) **|** (result\_3 **<<** 8) **|** result\_4;

**return** result;

}

*unsigned* *int* *\**MixColumns(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

    {

        result[i] **=** multiply\_column(**state**[i]);

    }

**return** result;

}

*void* showMatrix(*unsigned* *int* **word**[4])

{

**for** (*int* i **=** 0; i **<** 4; i**++**)

    {

        cout **<<** "\n\t";

        showWord(**word**[i]);

    }

}

string matrixToStr(*unsigned* *int* *\****word**)

{

    string str **=** "";

**for** (*int* i **=** 0; i **<** 4; i**++**)

    {

        str **+=** wordToStr(**word**[i]);

    }

**return** str;

}

*unsigned* *int* *\**EncryptionAES(*unsigned* *int* **state**[4], *unsigned* *int* **arrayKey**[4])

{

*unsigned* *int* **\***key **=** keyExpansion(**arrayKey**);

**state** **=** AddRoundKey(**state**, **&**key[0]);

**for** (*int* j **=** 1; j **<=** 9; j**++**)

    {

**state** **=** SubBytes(**state**);

**state** **=** ShiftRows(**state**);

**state** **=** MixColumns(**state**);

**state** **=** AddRoundKey(**state**, **&**key[4 **\*** j]);

    }

**state** **=** SubBytes(**state**);

**state** **=** ShiftRows(**state**);

**state** **=** AddRoundKey(**state**, **&**key[40]);

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

    result **=** **state**;

**return** result;

}

*unsigned* *int* *\**InvShiftRows(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

    {

*unsigned* *int* byte\_1 **=** **state**[i] **&** 0xFF000000;

*unsigned* *int* byte\_2 **=** **state**[(i **+** 3) **%** 4] **&** 0xFF0000;

*unsigned* *int* byte\_3 **=** **state**[(i **+** 2) **%** 4] **&** 0xFF00;

*unsigned* *int* byte\_4 **=** **state**[(i **+** 1) **%** 4] **&** 0xFF;

        result[i] **=** byte\_1 **|** byte\_2 **|** byte\_3 **|** byte\_4;

    }

**return** result;

}

*unsigned* *int* InvSubWord(*unsigned* *int* **word**)

{

*int* InvS[] **=** {

        0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,

        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,

        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,

        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,

        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,

        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,

        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,

        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,

        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,

        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,

        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,

        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,

        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,

        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,

        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,

        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d};

*unsigned* *int* result **=** 0;

**for** (*int* i **=** 1; i **<=** 4; i**++**)

    {

*unsigned* *int* byte\_i **=** (**word** **>>** (32 **-** i **\*** 8)) **&** 0xFF;

*unsigned* *int* subB **=** InvS[byte\_i];

        result **=** (result **<<** 8) **|** subB;

    }

**return** result;

}

*unsigned* *int* *\**InvSubBytes(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

        result[i] **=** InvSubWord(**state**[i]);

**return** result;

}

*unsigned* *int* multiply\_9(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** (**word** **<<** 3) **^** **word**;

**if** (result **>** (256 **<<** 2))

        result **^=** (0x11b **<<** 2);

**if** (result **>** (256 **<<** 1))

        result **^=** (0x11b **<<** 1);

**if** (result **>** 256)

        result **^=** 0x11b;

    result **&=** 0xFF;

**return** result;

}

*unsigned* *int* multiply\_B(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** (**word** **<<** 3) **^** (**word** **<<** 1) **^** **word**;

**if** (result **>** (256 **<<** 2))

        result **^=** (0x11b **<<** 2);

**if** (result **>** (256 **<<** 1))

        result **^=** (0x11b **<<** 1);

**if** (result **>** 256)

        result **^=** 0x11b;

    result **&=** 0xFF;

**return** result;

}

*unsigned* *int* multiply\_D(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** (**word** **<<** 3) **^** (**word** **<<** 2) **^** **word**;

**if** (result **>=** (256 **<<** 2))

        result **^=** (0x11b **<<** 2);

**if** (result **>=** (256 **<<** 1))

        result **^=** (0x11b **<<** 1);

**if** (result **>=** 256)

        result **^=** 0x11b;

    result **&=** 0xFF;

**return** result;

}

*unsigned* *int* multiply\_E(*unsigned* *int* **word**)

{

*unsigned* *int* result **=** (**word** **<<** 3) **^** (**word** **<<** 2) **^** (**word** **<<** 1);

**if** (result **>=** (256 **<<** 2))

        result **^=** (0x11b **<<** 2);

**if** (result **>=** (256 **<<** 1))

        result **^=** (0x11b **<<** 1);

**if** (result **>=** 256)

        result **^=** 0x11b;

    result **&=** 0xFF;

**return** result;

}

*unsigned* *int* InvMultiply\_column(*unsigned* *int* **word**)

{

*unsigned* *int* result;

*unsigned* *int* byte\_1 **=** (**word** **>>** 24) **&** 0xFF;

*unsigned* *int* byte\_2 **=** (**word** **>>** 16) **&** 0xFF;

*unsigned* *int* byte\_3 **=** (**word** **>>** 8) **&** 0xFF;

*unsigned* *int* byte\_4 **=** **word** **&** 0xFF;

*unsigned* *int* result\_1 **=** multiply\_E(byte\_1) **^** multiply\_B(byte\_2) **^** multiply\_D(byte\_3) **^** multiply\_9(byte\_4);

*unsigned* *int* result\_2 **=** multiply\_9(byte\_1) **^** multiply\_E(byte\_2) **^** multiply\_B(byte\_3) **^** multiply\_D(byte\_4);

*unsigned* *int* result\_3 **=** multiply\_D(byte\_1) **^** multiply\_9(byte\_2) **^** multiply\_E(byte\_3) **^** multiply\_B(byte\_4);

*unsigned* *int* result\_4 **=** multiply\_B(byte\_1) **^** multiply\_D(byte\_2) **^** multiply\_9(byte\_3) **^** multiply\_E(byte\_4);

    result **=** (result\_1 **<<** 24) **|** (result\_2 **<<** 16) **|** (result\_3 **<<** 8) **|** result\_4;

**return** result;

}

*unsigned* *int* *\**InvMixColumns(*unsigned* *int* **state**[4])

{

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

**for** (*int* i **=** 0; i **<** 4; i**++**)

        result[i] **=** InvMultiply\_column(**state**[i]);

**return** result;

}

*unsigned* *int* *\**DecryptionAES(*unsigned* *int* **Cipher**[4], *unsigned* *int* **arrayKey**[4])

{

*unsigned* *int* **\***key **=** keyExpansion(**arrayKey**);

*unsigned* *int* **\***state **=** AddRoundKey(**Cipher**, **&**key[40]);

**for** (*int* j **=** 1; j **<=** 9; j**++**)

    {

        state **=** InvShiftRows(state);

        state **=** InvSubBytes(state);

        state **=** AddRoundKey(state, **&**key[40 **-** 4 **\*** j]);

        state **=** InvMixColumns(state);

    }

    state **=** InvShiftRows(state);

    state **=** InvSubBytes(state);

    state **=** AddRoundKey(state, **&**key[0]);

*unsigned* *int* **\***result **=** **new** *unsigned* *int*[4];

    result **=** state;

**return** result;

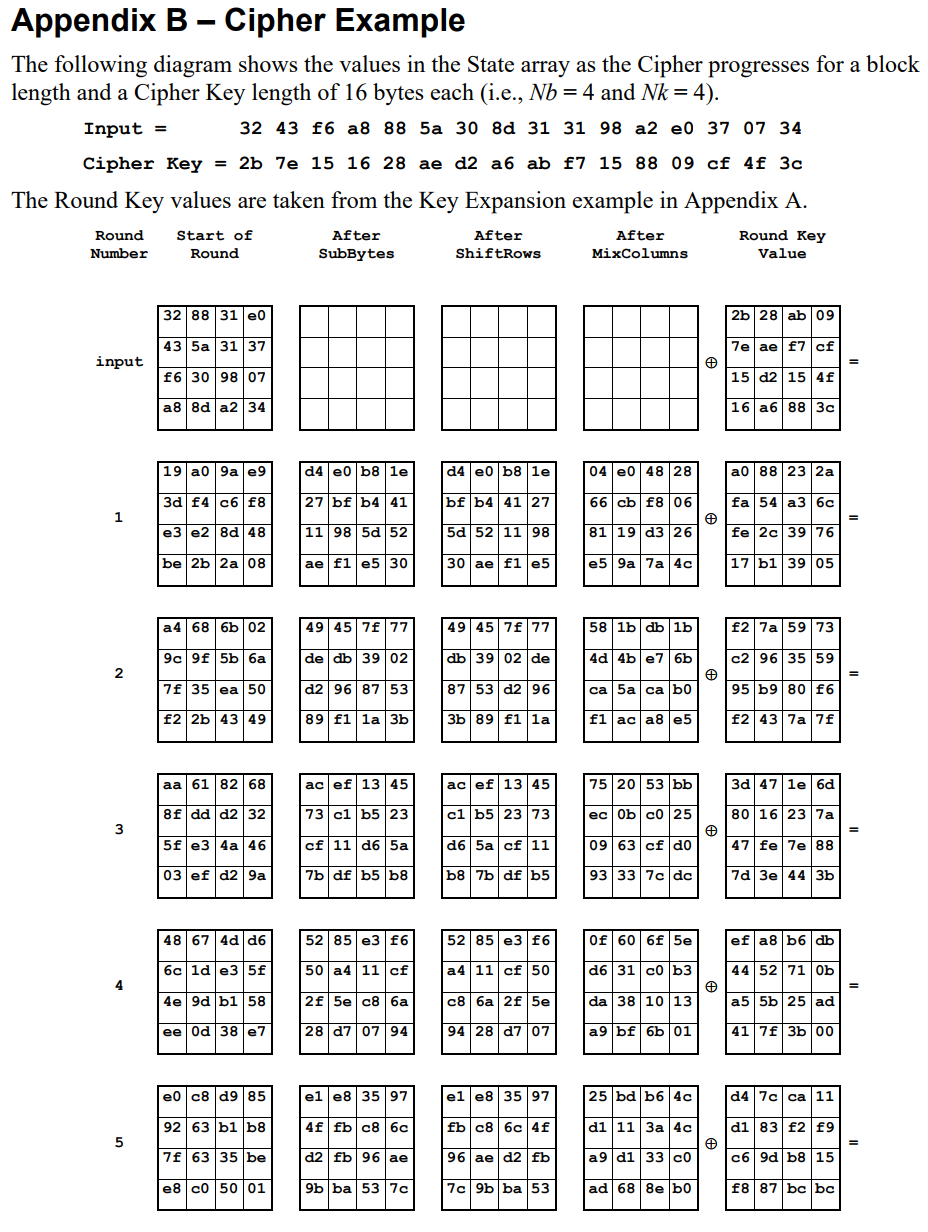
}

* 1. **Kết quả khi chạy chương trình mô phỏng**

Text

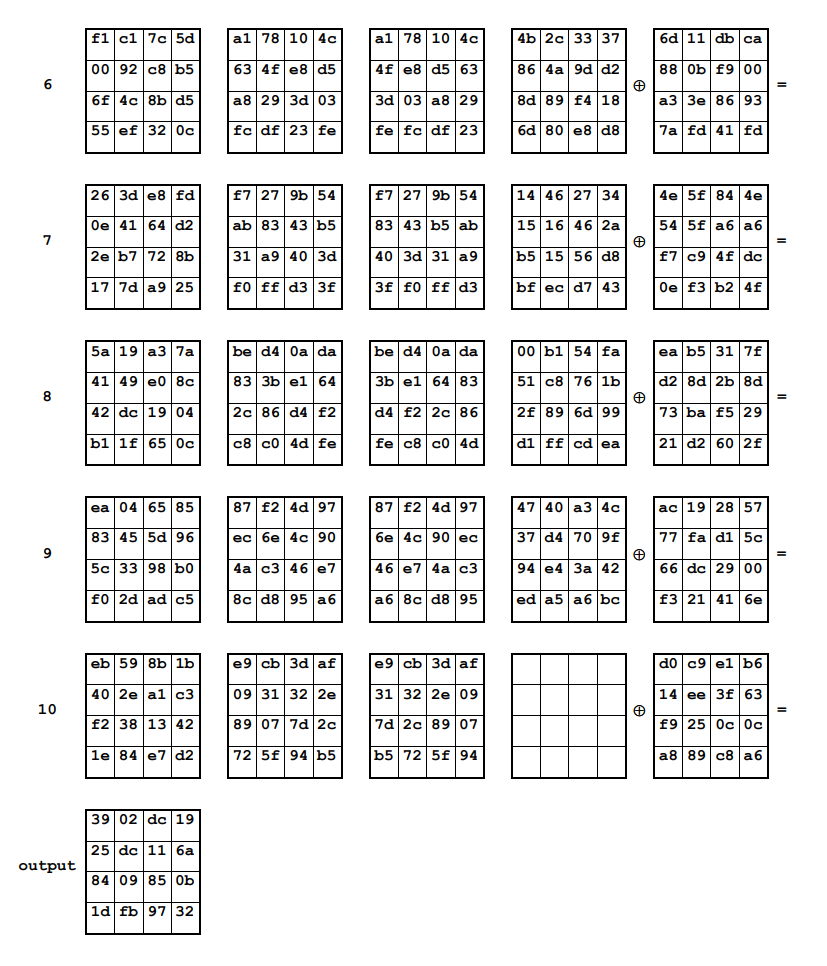
Description automatically generated

**Hình 5.3.1 Kết quả mô phỏng thuật toán AES**



***Hình 5.3.2 Kết quả từng vòng của thuật toán AES***

**Hình 5.3. Kết quả mô phỏng Giải mã AES**



***Hình 5.3.3 Kết quả từng vòng của thuật toán AES (Tiếp)***

## Tính bảo mật của AES

Thiết kế và độ dài khoá của thuật toán AES (128-bit, 192-bit và 256-bit) là đủ an toàn để bảo vệ các thông tin được xếp vào loại TỐI MẬT (SECRET). Các thông tin TUYỆT MẬT (TOP SECRET) sẽ phải dùng khóa 192-bit hoặc 256-bit. Các phiên bản thực hiện AES nhằm mục đích bảo vệ hệ thống an ninh hay thông tin quốc gia phải được NSA kiểm tra và chứng nhận trước khi sử dụng.

Điều này đánh dấu lần đầu tiên công chúng có quyền tiếp xúc với thuật toán mật mã mà NSA phê chuẩn cho thông tin TUYỆT MẬT. Nhiều phần mềm thương mại hiện nay sử dụng mặc định khóa có độ dài 128-bit.

Phương pháp thường dùng nhất để tấn công các dạng mã hoá khối là thử các kiểu tấn công lên phiên bản có số chu trình thu gọn. Đối với khóa 128-bit, 192-bit và 256-bit, AES có tương ứng 10, 12 và 14 chu trình. Tại thời điểm năm 2006, những tấn công thành công được biết đến là 7 chu trình đối với khóa 128-bit, 8 chu trình với khóa 192-bit và 9 chu trình với khóa 256-bit.

Một số nhà khoa học trong lĩnh vực mật mã lo ngại về an ninh của AES. Họ cho rằng ranh giới giữa số chu trình của thuật toán và số chu trình bị phá vỡ quá nhỏ. Nếu các kỹ thuật tấn công được cải thiện thì AES có thể bị phá vỡ. Ở đây, *phá vỡ* có nghĩa chỉ bất cứ phương pháp tấn công nào nhanh hơn tấn công kiểu duyệt toàn bộ. Vì thế một tấn công cần thực hiện 2120 cũng được coi là thành công mặc dù tấn công này chưa thể thực hiện trong thực tế. Tại thời điểm hiện nay, nguy cơ này không thực sự nguy hiểm và có thể bỏ qua. Tấn công kiểu duyệt toàn bộ quy mô nhất đã từng thực hiện là do distributed.net thực hiện lên hệ thống 64 bit RC5 vào năm 2002 (Theo định luật Moore thì nó tương đương với việc tấn công vào hệ thống 66 bit hiện nay).

Một vấn đề khác nữa là cấu trúc toán học của AES. Không giống với các thuật toán mã hóa khác, AES có mô tả toán học khá đơn giản. Tuy điều này chưa dẫn đến mối nguy hiểm nào nhưng một số nhà nghiên cứu sợ rằng sẽ có người lợi dụng được cấu trúc này trong tương lai.

## KẾT LUẬN

* + 1. **Đánh giá thuật toán**
* Phương pháp mã hoá AES với tên tiếng anh là Advanced Encryption Standard – Tiêu chuẩn mã hoá nâng cao. Thuật toán mã hoá
* Thực hiện hiệu quả trên các vi xử lý 8-bit, 32-bit.
* AES sẽ thay thế cho DES trong tương lai. Do có độ bảo mật cao, dễ thiết kế và đạt hiệu quả cao.
  + 1. **Đánh giá AES**

Tổng kết lại quá trình tìm hiểu, mã AES có những đặc điểm sau

* Độ bảo mật lúc nào cũng ở mức cao
* Với 128-Bit => 10 Vòng mã hoá.
* Với 192-Bit => 12 Vòng mã hoá.
* Với 256 Bit => 14 Vòng mã hoá.

Việc phát triển Internet ngày càng tăng. Nhiều dữ liệu dạng kỹ thuật số được chia sẻ mỗi ngày. Nhiều thông tin nhạy cảm cũng như tài liệu riêng tư cần có lớp bảo mật cao để chống lại những kẻ xâm nhập.

* + 1. **Tổng kết**

AES là một trong những thuật toán mạnh mẽ nhất và rộng rãi, được hỗ trợ trên phần cứng và phần mềm. Với kích thước khối 128-bit, 192-bit, 256-bit, AES cung cấp khả năng bảo vệ nhiều hơn các thuật toán khác.

* 1. **Hướng phát triển của đề tài**
  + Theo tính dễ mở rộng về kích thước khóa và kích thước khối: thuật toán mã hoá khối được xây dựng ứng với một số kích thước khoá và một số kích thước khối nhất định. Tuỳ theo nhu cầu sử dụng và mức độ an toàn cần được áp dụng để mã hoá, người dùng sẽ cân nhắc việc sử dụng khoá 128-bit lên 192-bit hay 256-bit.
  + Đề xuất thêm thuật toán mã hoá khối mới để nâng mức bảo mật an toàn thông tin lên. Trong tương lai, nếu có chuẩn mã mới được tìm hiểu và nghiên cứu có cấu trúc tốt, khả năng tổng quát hoá để phù hợp với kích thước khoá và kích thước khối lớn thì nên cân nhắc mở rộng thuật toán để thay thế cho phiên bản hiện tại.
  + Hầu hết các thuật toán mã hoá khối hiện này được xây dựng trên kiến trúc xử lý byte (8-bit). Do đó, các thuật toán này chưa thực sự phù hợp trong các hệ thống mà đơn vị dữ liệu xử lý không phải là byte. Ví dụ như trong một số hệ thống mạng cảm ứng không dậy hoặc Ubiquitous, dữ liệu được xử lý theo từng nhóm 4-bit hoặc 6-bit.

# **TÀI LIỆU THAM KHẢO**

1. **Đỗ Quang Hưng (Chủ biên), Phạm Trường Giang, Bùi Hải Đăng (2017) -** “Giáo trình Kiến Trúc Máy Tính” **–** Bộ Giao Thông Vận Tải, Trường Đại học Công Nghệ Giao Thông Vận Tải. ISBN: 978-604-913-606-1.
2. **Dương Anh Đức, Trần Minh Triết, Lương Hán Cơ (2001),** “Ứng dụng chuẩn mã hóa AES và các phiên bản mở rộng vào Hệ thống Thư điện tử an toàn tại Việt Nam “, Hội nghị khoa học kỷ niệm 25 năm Viện Công Nghệ Thông Tin, Hà Nội, Việt Nam.
3. **Vietnamnet (28 Oct 2021) –** An ninh mạng Việt Nam trong năm 2021: Tấn công ngày càng tinh vi
4. **TechTarget (September 2021)** – What is AES – Advanced Encryption Standard
5. **Geekforgeeks.org (11 Feb 2022)** – Step by Step of Encryption and Decryption AES.
6. **SergeyBel – Github.com (30 Apr 2022) -** C++ AES (Advanced Encryption Standard) implementation
7. **E. Barkan, E. Biham (2002),** In how many ways can you write Rijndael, ASIACRYPT EP02, LNCS, vol.2501, Springer,
8. **Federal Information (2001) -** Specification for the Advanced Encryption Standard (AES)
9. **Cloudflare (2018) -** What is encryption?
10. **Wikipedia -** The Advanced Encryption Standard