|  |  |
| --- | --- |
| HỌC VIỆN KỸ THUẬT MẬT MÃ  **KHOA CÔNG NGHỆ THÔNG TIN**  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯  Logo HvKTMM | HỌC VIỆN KỸ THUẬT MẬT MÃ  **KHOA CÔNG NGHỆ THÔNG TIN**  ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯  Logo HvKTMM |
| BÀI TẬP MÔN HỌC CƠ SỞ AN TOÀN VÀ BẢO MẬT THÔNG TIN  **VIẾT CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ MỘT FILE DỮ LIỆU BẰNG MẬT MÃ DES VÀ NGÔN NGỮ C#** |
| Ngành: Công nghệ thông tin  Chuyên ngành: Kỹ thuật phần mềm nhúng và di động  Nhóm: 13  Sinh viên thực hiện:  **Hà Tiến Thành – CT060138**  **Phạm Năng Chiến – CT060106**  Lớp: L03  Người hướng dẫn:  **TS. Nguyễn Đào Trường**  Khoa Công nghệ thông tin – Học viện Kỹ thuật mật mã  Hà Nội, 2025 |  |

**NHẬN XÉT VÀ CHO ĐIỂM CỦA GIÁO VIÊN**

MỤC LỤC

[**THUẬT NGỮ VÀ TỪ VIẾT TẮT** iv](#_Toc194442006)

[**DANH MỤC HÌNH ẢNH v**](#_Toc194442007)

[**LỜI NÓI ĐẦU 1**](#_Toc194442008)

[**CHƯƠNG 1. TỔNG QUAN VỀ AN TOÀN VÀ BẢO MẬT THÔNG TIN 2**](#_Toc194442009)

[1.1. Tổng quan về mật mã học và an toàn bảo mật thông tin 2](#_Toc194442010)

[1.1.1. Tổng quan về mật mã học 2](#_Toc194442011)

[1.1.2. Vai trò của mật mã trong đảm bảo an toàn thông tin 3](#_Toc194442012)

[1.2. Tổng quan về thuật toán DES 4](#_Toc194442013)

[1.2.1. Giới thiệu tổng quan về DES 4](#_Toc194442014)

[1.2.2. Cách thức hoạt động của DES 4](#_Toc194442015)

[1.2.3. Quá trình mã hóa với DES 5](#_Toc194442016)

[1.2.4. Quá trình giải mã với DES 6](#_Toc194442017)

[1.3. Tổng quan về C# 7](#_Toc194442018)

[**CHƯƠNG 2. XÂY DỰNG CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ BẰNG MẬT MÃ DES 9**](#_Toc194442019)

[2.1. Trình soạn thảo Visual Studio 9](#_Toc194442020)

[2.2. Xây dựng chương trình 10](#_Toc194442021)

[2.2.1. Thuật toán DES 10](#_Toc194442022)

[2.2.2. Hàm tạo khóa con 19](#_Toc194442023)

[2.2.3. Hàm mã hóa f(R,K) 20](#_Toc194442024)

[2.2.4. Hàm mã hóa và giải mã 20](#_Toc194442025)

[2.2.5. Các hàm hỗ trợ 22](#_Toc194442026)

[**CHƯƠNG 3. TỔNG KẾT 24**](#_Toc194442027)

[3.1. Kết quả đạt được 24](#_Toc194442028)

[3.1.1. Về mặt lý thuyết 24](#_Toc194442029)

[3.1.2. Chương trình mã hóa và giải mã 24](#_Toc194442030)

[3.2. Đánh giá 26](#_Toc194442031)

[3.2.1. Ưu điểm 26](#_Toc194442032)

[3.2.2. Nhược điểm 26](#_Toc194442033)

[3.3. Phương hướng phát triển 27](#_Toc194442034)

[**PHỤ LỤC 28**](#_Toc194442035)

[**TÀI LIỆU THAM KHẢO 33**](#_Toc194442036)

# THUẬT NGỮ VÀ TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Viết tắt** | **Tiếng Anh** | **Tiếng Việt** |
| DES | Data Encryption Standard | Tiêu chuẩn mã hóa dữ  liệu |
| AES | Advanced Encryption Standard | Tiêu chuẩn mã hóa tiên  tiến |
| RSA | Rivest-Shamir-Adleman | Một thuật toán mã hóa  khóa công khai |
| EC13C | Elliptic Curve Cryptography | Mật mã đường cong  elip |
| IBM | International Business Machines  Corporation | Tập đoàn Máy tính  Quốc tế |
| IDE | Integrated Development  Environment | Môi trường phát triển  tích hợp |

# DANH MỤC HÌNH ẢNH

[Hình 1.1. Sơ đồ mã hóa và giải mã 2](#_Toc194440531)

[Hình 1.2.Quá trình mã hóa của DES 6](#_Toc194440532)

[Hình 2.1. Visual Studio 9](#_Toc194440533)

[Hình 2.2. Sơ đồ tạo khóa con 11](#_Toc194440534)

[Hình 2.3. Bảng hoán vị PC – 1 11](#_Toc194440535)

[Hình 2.4. Bảng dịch bit theo vòng lặp 12](#_Toc194440536)

[Hình 2.5. Bảng ma trận hoán vị PC – 2 12](#_Toc194440537)

[Hình 2.6. Bảng hoán vị IP 13](#_Toc194440538)

[Hình 2.7. Sơ đồ một vòng mã hóa DES 13](#_Toc194440539)

[Hình 2.8. Sơ đồ hàm F 14](#_Toc194440540)

[Hình 2.9. Bảng hộp S – Box 17](#_Toc194440541)

[Hình 2.10. Bảng hoán vị P (P – Pox) 17](#_Toc194440542)

[Hình 2.11. Bảng mô tả hàm mở rộng E 18](#_Toc194440543)

[Hình 2.12. Bảng hoán vị IP – 1 18](#_Toc194440544)

[Hình 2.13. Quá trình mã hóa 19](#_Toc194440545)

[Hình 3.1. Giao diện ứng dụng mã hóa DES và gửi file bằng C# 24](#_Toc194440546)

[Hình 3.2. Giao diện nhận file và giải file mã hóa 25](#_Toc194440547)

[Hình 3.3. Hình thông báo gửi file thành công 25](#_Toc194440548)

[Hình 3.4. Hình ảnh đã nhận được file mã hóa ở chương trình nhận 26](#_Toc194440549)

# LỜI NÓI ĐẦU

Trong thế giới hiện đại, vai trò của máy tính và hệ thống thông tin điện tử ngày càng quan trọng, càng ngày càng có nhiều nhu cầu truyền dẫn, lưu trữ và thậm chí là thực hiện các giao dịch nghiệp vụ trên các hệ thống thông tin điện tử. Trong xã hội bùng nổ thông tin, khi mà thông tin có vai trò và giá trị vượt trội quyết định đến sự thành bại của công tác nghiệp vụ, từ các doanh nghiệp vừa và nhỏ đến các tập đoàn lớn xuyên quốc gia, các cơ quan an ninh, các tổ chức chính trị, xã hội cho đến các trường học, viện nghiên cứu thì vấn đề đảm bảo được an ninh thông tin là một vấn đề được đặt lên hàng đầu. Do vậy, một ứng dụng công nghệ thông tin ngoài việc đáp ứng đầy đủ các yêu cầu nghiệp vụ còn đòi hỏi phải đảm bảo được tính an toàn cho thông tin và dữ liệu trong quá trình xử lý và lưu trữ, tức là phải đảm bảo được các đặc tính:

* Tính bí mật (Confidential)
* Tính xác thực (Authentication)
* Tính toàn vẹn (Intergrity) của thông tin.

Để đảm bảo được các đặc tính này của thông tin, hệ thống thông tin và người quản trị hệ thống cần thực hiện rất nhiều quy tắc và phương pháp khác nhau, từ đảm bảo an toàn vật lý cho đến đảm bảo an toàn người dùng. Và đặc biết quan trọng nhất là đảm bảo an toàn dữ liệu khi lưu trữ va truyền dẫn. Chính vì lý do đó, nhóm chúng em đã thực hiện đề tài: “Viết chương trình mã hóa và giải mã một file dữ liệu bằng mật mã DES với ngôn ngữ C#”.

Nội dung báo cáo bao gồm:

Chương 1. Tổng quan về an toàn và bảo mật thông tin

Chương 2. Xây dựng chương trình mã hóa và giải mã bằng mật mã DES

Chương 3. Tổng kết

Trong quá trình làm báo cáo, do trình độ chuyên môn và năng lực còn nhiều hạn chế cũng như thời gian thực hiện có hạn nên không thể tránh khỏi những sai sót. Vì vậy rất mong được sự đóng góp của các thầy, cô và các bạn để bài báo cáo hoàn thiện hơn.

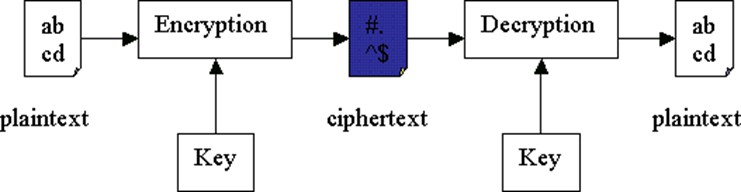
Nhóm xin chân thành cảm ơn!

# TỔNG QUAN VỀ AN TOÀN VÀ BẢO MẬT THÔNG TIN

## Tổng quan về mật mã học và an toàn bảo mật thông tin

### Tổng quan về mật mã học

Mật mã học là một lĩnh vực của khoa học máy tính và toán học, nghiên cứu về cách để bảo vệ thông tin và truyền thông trước các kẻ tấn công. Mật mã học cung cấp các công cụ để mã hóa thông tin, giúp người gửi thông tin và người nhận thông tin có thể trao đổi dữ liệu một cách an toàn mà không bị lộ thông tin. Đây là một lĩnh vực 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…



Hình 1.1. Sơ đồ mã hóa và giải mã

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…

Có hai loại cơ bản của mật mã học: mật mã hóa đối xứng và mật mã hóa không đối xứng. Trong mật mã hóa đối xứng, cùng một khóa được sử dụng để mã hóa và giải mã thông tin. Các ví dụ điển hình của mật mã hóa đối xứng bao gồm DES, AES, và Blowfish.

Trong khi đó, mật mã hóa không đối xứng sử dụng hai khóa khác nhau để mã hóa và giải mã thông tin. Một khóa được gọi là khóa công khai và được công bố rộng rãi, trong khi khóa còn lại được gọi là khóa bí mật và chỉ được giữ bí mật. Các ví dụ của mật mã hóa không đối xứng bao gồm RSA và ECC.

Ngoài ra, mật mã học còn bao gồm các kỹ thuật khác như mã hóa đối xứng dòng, mã hóa một lần, hàm băm, chữ ký điện tử và chứng chỉ số. Mật mã học được sử dụng rộng rãi trong các ứng dụng bảo mật như giao dịch ngân hàng trực tuyến, email, ứng dụng trò chuyện và truyền thông an toàn trong các mạng lưới lớn.

### Vai trò của mật mã trong đảm bảo an toàn thông tin

Mật mã đóng vai trò rất quan trọng trong việc đảm bảo an toàn thông tin. Nó được sử dụng để bảo vệ các thông tin nhạy cảm và đảm bảo rằng chỉ những người được ủy quyền mới có thể truy cập thông tin đó.

Mật mã là một hệ thống mã hóa thông tin, trong đó thông tin được chuyển đổi từ dạng gốc sang dạng mã hóa, để ngăn chặn các kẻ tấn công từ việc đọc hoặc hiểu được thông tin đó.

Mật mã được sử dụng trong nhiều lĩnh vực, bao gồm truyền thông trên mạng, lưu trữ dữ liệu trên các thiết bị, thanh toán điện tử, truy cập vào các hệ thống máy tính và nhiều hơn nữa.

Một số kỹ thuật mã hóa thông tin phổ biến bao gồm mã hóa đối xứng, mã hóa không đối xứng và hàm băm. Mã hóa đối xứng sử dụng cùng một khóa để mã hóa và giải mã thông tin, trong khi mã hóa không đối xứng sử dụng một khóa công khai để mã hóa thông tin và một khóa bí mật để giải mã thông tin. Hàm băm được sử dụng để tạo ra một giá trị băm độc nhất của thông tin, để đảm bảo tính toàn vẹn của thông tin khi truyền qua mạng. thực hiện mã hóa/giải mã trong thuât toán. 8 bít còn lại được dùng để kiểm tra/phát hiện lỗi.

Các bước hoạt động của DES như sau:

Mật mã đóng vai trò quan trọng trong đảm bảo an toàn thông tin bởi vì nó giúp ngăn chặn các kẻ tấn công có ý định truy cập vào thông tin nhạy cảm, như mật khẩu, tài khoản ngân hàng hoặc thông tin cá nhân của người dùng. Nếu mật mã không được sử dụng, thông tin có thể dễ dàng bị đánh cắp hoặc xâm nhập, gây ra nguy hiểm cho người dùng và tổ chức.

Mật mã học giúp đảm bảo những tính chất sau cho thông tin:

* Tính bí mật (confidentiality): thông tin chỉ được tiết lộ cho những ai được phép.
* Tính toàn vẹn (integrity): thông tin không thể bị thay đổi mà không bị phát hiện.
* Tính xác thực (authentication): người gửi (hoặc người nhận) có thể chứng minh đúng họ.
* Tính chống chối bỏ (non – repudiation): người gửi hoặc nhận này không thể chối bỏ việc đã gửi hoặc nhận thông tin.

## Tổng quan về thuật toán DES

### Giới thiệu tổng quan về DES

Thuật toán mã khối DES (Data Encryption Standard) là một thuật toán mã khối với kích thước khối 64 bít và kích thước khóa 56 bít, được công bố chính thức bởi Tổ chức Tiêu chuẩn xử lý thông tin liên bang Hoa Kỳ (FIPS) vào tháng 11/1976 và được xuất bản trong tài liệu FIPS PUB 46 (01/1977). Thuật toán DES đã trải quả nhiều lần cập nhật: năm 1988 (FIPS-46-1), 1993 (FIPS-46-2), 1998 (FIPS-46-3).

Tiền thân của thuật toán DES là thuật toán Lucifer, một thuật toán do IBM phát triển. Cuối năm 1976, DES được chọn làm chuẩn mã hóa dữ liệu của Hoa Kỳ, sau đó được sử dụng rộng rãi trên toàn thế giới trong lĩnh vực an toàn, bảo mật thông tin trên môi trường số.

### Cách thức hoạt động của DES

Thuật toán DES được thiết kế để mã hóa và giải mã các khối dữ liệu 64 bít với một khóa có độ dài 64 bít. Việc giải mã phải được thực hiện bởi một khóa tương ứng như trong quá trình mã hóa, mỗi khóa gồm 64 bít, trong đó 56 bít được sử dụng để thực hiện mã hóa/giải mã trong thuât toán. 8 bít còn lại được dùng để kiểm tra/phát hiện lỗi.

Các bước hoạt động của DES như sau:

Chuẩn bị dữ liệu: Dữ liệu được mã hóa bằng DES được chia thành các khối có độ dài cố định là 64 bit.

Khóa: DES sử dụng một khóa bí mật có độ dài 56 bit. Trước khi sử dụng, khóa này sẽ được xử lý bằng thuật toán "key schedule" để tạo ra 16 khóa con có độ dài 48 bit mỗi khóa.

Pha mã hóa: Dữ liệu được chia thành hai nửa có độ dài 32 bit mỗi nửa. Tiếp theo, mỗi nửa dữ liệu được đưa qua một hàm mã hóa khác nhau và mỗi hàm sử dụng một khóa con. Quá trình mã hóa này được lặp lại 16 lần, với dữ liệu và khóa con được cập nhật sau mỗi lần mã hóa.

Ghép dữ liệu: Sau khi 16 lần mã hóa, hai nửa dữ liệu được ghép lại và trả về dưới dạng một khối 64 bit đã được mã hóa.

Quá trình giải mã tương tự như pha mã hóa, nhưng khóa con được sử dụng theo thứ tự ngược lại.

### Quá trình mã hóa với DES

Quá trình mã hóa với DES bao gồm các bước sau:

Chuẩn bị dữ liệu: Dữ liệu cần mã hóa được chia thành các khối có độ dài cố định là 64 bit. Nếu dữ liệu ban đầu không đủ 64 bit, chúng ta sẽ thêm các bit không cần thiết cho đủ độ dài.

Khởi tạo khóa: Khóa bí mật được sử dụng để mã hóa và giải mã dữ liệu có độ dài 56 bit. Tuy nhiên, để tăng cường bảo mật, khóa bí mật này được mở rộng lên thành 64 bit bằng cách thêm một bit kiểm tra cho mỗi byte.

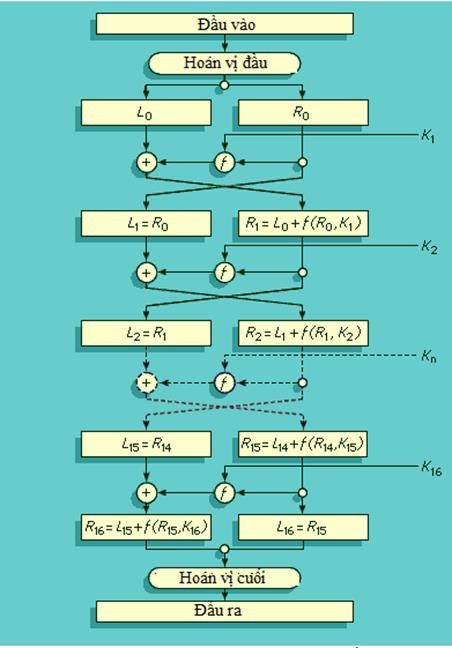
Tạo các khóa con: Sau khi khởi tạo khóa, chúng ta sử dụng một thuật toán gọi là "key schedule" để tạo ra 16 khóa con có độ dài 48 bit mỗi khóa. Các khóa con này sẽ được sử dụng để mã hóa dữ liệu trong quá trình mã hóa.

Pha mã hóa: Dữ liệu được chia thành hai nửa có độ dài 32 bit mỗi nửa. Tiếp theo, mỗi nửa dữ liệu được đưa qua một hàm mã hóa khác nhau và mỗi hàm sử dụng một khóa con. Quá trình mã hóa này được lặp lại 16 lần, với dữ liệu và khóa con được cập nhật sau mỗi lần mã hóa.

Ghép dữ liệu: Sau khi 16 lần mã hóa, hai nửa dữ liệu được ghép lại và trả về dưới dạng một khối 64 bit đã được mã hóa.

Quá trình mã hóa với DES sử dụng phương pháp mã hóa đối xứng, có nghĩa là cùng một khóa bí mật được sử dụng cho cả quá trình mã hóa và giải mã. Quá trình giải mã với DES tương tự với quá trình mã hóa, nhưng khóa con được sử dụng theo thứ tự ngược lại.

Trong quá trình mã hóa, dữ liệu đầu vào phải thực hiện quá trình hoán vị đầu (initial permutation) và hoán vị cuối (final permutation) sau vòng thứ 16. Việc thực hiện hoán vị phục vụ cho quá trình đưa thông tin vào và lấy thông tin ra từ các khối phần cứng, tạo điều kiện cho việc cài đặt phần cứng. Hàm cơ sở f cho phép đảm bảo tính bảo mật trong thuật toán DES này.



Hình 1.2.Quá trình mã hóa của DES

### Quá trình giải mã với DES

Quá trình giải mã chính là quá trình mã hóa. Nhưng dùng khóa con theo chiều ngược lại. Tức là khối đầu vào là bản mã hóa, dùng khóa con theo thứ tự ngược lại: K16, K15, ..., K1

Quá trình giải mã với DES được thực hiện theo các bước sau:

Chuẩn bị dữ liệu: Khối dữ liệu đã được mã hóa cần được chia thành hai nửa có độ dài 32 bit mỗi nửa.

Khởi tạo khóa: Khóa bí mật được sử dụng để giải mã cũng phải được khởi tạo và mở rộng lên thành khóa 64 bit giống như trong quá trình mã hóa.

Tạo các khóa con: Tương tự như trong quá trình mã hóa, chúng ta sử dụng thuật toán key schedule để tạo ra 16 khóa con có độ dài 48 bit mỗi khóa.

Pha giải mã hóa: Tiếp theo, mỗi nửa dữ liệu được đưa qua hàm giải mã hóa tương ứng với khóa con theo thứ tự ngược lại so với quá trình mã hóa. Quá trình giải mã hóa này được lặp lại 16 lần, với dữ liệu và khóa con được cập nhật sau mỗi lần giải mã hóa.

Ghép dữ liệu: Cuối cùng, hai nửa dữ liệu đã giải mã được ghép lại để tạo ra khối dữ liệu ban đầu.

Tương tự như quá trình mã hóa, quá trình giải mã với DES sử dụng phương pháp mã hóa đối xứng, có nghĩa là cùng một khóa bí mật được sử dụng cho cả quá trình mã hóa và giải mã. Quá trình giải mã hóa với DES tương tự như quá trình mã hóa, chỉ khác là khóa con được sử dụng theo thứ tự ngược lại.

## Tổng quan về C#

C# là là một ngôn ngữ lập trình hiện đại, mạnh mẽ và đa mục đích được phát triển bởi Microsoft. Nó là một phần của .NET Framework, một nền tảng phát triển phần mềm chủ yếu dành cho hệ điều hành Windows, nhưng cũng có thể được sử dụng trên các nền tảng khác như Linux và macOS thông qua .NET Core và .NET5+

Các đặc điểm của C#:

Đơn giản và dễ đọc: C# được thiết kế với cú pháp đơn giản và dễ hiểu, giúp cho người mới bắt đầu có thể nhanh chóng tiếp cận và học cách sử dụng ngôn ngữ này.

An toàn và kiểm soát kiểu: C# hỗ trợ kiểm soát kiểu mạnh mẽ, giúp phát hiện và ngăn chặn lỗi trong quá trình biên dịch và thời gian chạy.

Hướng đối tượng (OOP): C# hỗ trợ các tính năng của lập trình hướng đối tượng như kế thừa, đóng gói, và đa hình, giúp việc phát triển phần mềm dễ dàng hơn và dễ quản lý hơn.

Thư viện chuẩn và mạnh mẽ: C# được kết hợp với một bộ thư viện chuẩn rất mạnh mẽ, bao gồm thư viện xử lý chuỗi, thư viện IO, thư viện mạng, và nhiều hơn nữa, giúp giảm thiểu thời gian và công sức khi phát triển ứng dụng.

Đa nền tảng: Với sự ra đời của .NET Core và .NET 5+, C# trở thành một ngôn ngữ lập trình đa nền tảng, có thể chạy trên nhiều hệ điều hành khác nhau.

Hỗ trợ cộng đồng và công cụ phát triển: C# có cộng đồng lớn và nhiều công cụ phát triển mạnh mẽ như Visual Studio, Visual Studio Code, ReSharper, và nhiều công cụ khác.

# XÂY DỰNG CHƯƠNG TRÌNH MÃ HÓA VÀ GIẢI MÃ BẰNG MẬT MÃ DES

## Trình soạn thảo Visual Studio

Visual studio là một phần mềm hỗ trợ đắc lực hỗ trợ công việc lập trình website. Công cụ này được tạo lên và thuộc quyền sở hữu của ông lớn công nghệ Microsoft. Năm 1997, phần mềm lập trình nay có tên mã Project Boston. Nhưng sau đó, Microsoft đã kết hợp các công cụ phát triển, đóng gói thành sản phẩm duy nhất



Hình 2.1. Visual Studio

Visual Studio là hệ thống tập hợp tất cả những gì liên quan tới phát triển ứng dụng, bao gồm trình chỉnh sửa mã, trình thiết kế, gỡ lỗi. Tức là, bạn có thể viết code, sửa lỗi, chỉnh sửa thiết kế ứng dụng dễ dàng chỉ với 1 phần mềm Visual Studio mà thôi. Không dừng lại ở đó, người dùng còn có thể thiết kế giao diện, trải nghiệm trong Visual Studio như khi phát triển ứng dụng Xamarin, UWP bằng XAML hay Blend vậy.

Phần mềm Visual studio được chia thành 2 phiên bản Visual Studio Enterprise và Visual Studio Professional, các phiên bản cao cấp có tính phí này được sử dụng nhiều bởi các công ty chuyên về lập trình. Bên cạnh đó, Microsoft cũng cho ra mắt phiên bản Community (phiên bản miễn phí) của gói phần mềm, cung cấp cho người dùng những tính năng cơ bản nhất, phù hợp với các đối tượng lập trình không chuyên, mới tiếp cận tìm hiểu về công nghệ (đối tượng nghiên cứu, nhà phát triển cá nhân, hỗ trợ dự án mỡ, các tổ chức phi doanh nghiệp dưới 5 người dùng).

Các tính năng phần mềm Visual Studio:

* Đa nền tảng
* Đa ngôn ngữ lập trình
* Hỗ trợ website
* Kho tiện ích mở rộng phong phú
* Lưu trữ phân cấp
* Kho lưu trữ an toàn
* Màn hình đa nhiệm
* Hỗ trợ viết code
* Hỗ trợ thiết bị đầu cuối
* Hỗ trợ Git
* Tính năng comment

## Xây dựng chương trình

### Thuật toán DES

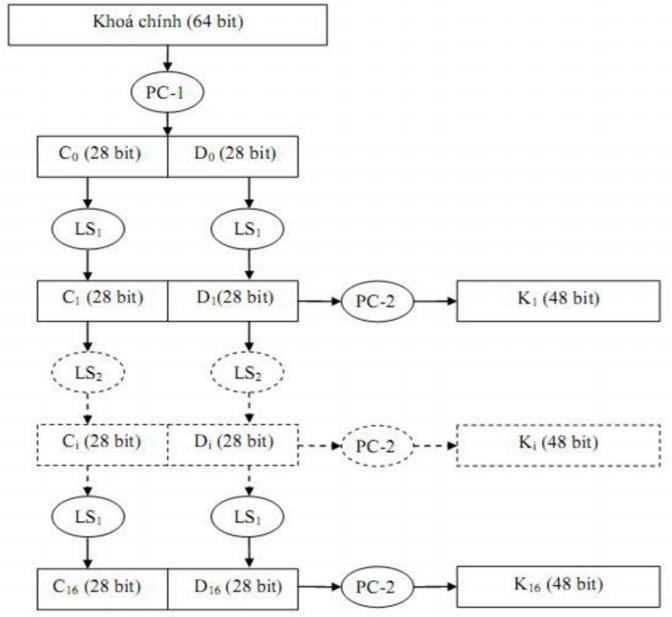
DES là thuật toán mã hóa khối, nó xử lý từng khối thông tin của bản rõ có độ dài xác định là 64 bit và được thực hiện mã hóa 16 lần. Vậy trước khi đi vào chu trình mã hóa thì khối dữ liệu cần mà hóa cần được tách ra từng khối 64 bít, và từng khối này sẽ được lần lượt đưa vào 16 vòng mã hóa DES để thực hiện. DES cũng sử dụng khóa để cá biệt hóa quá trình chuyển đổi. Nhờ vậy, chỉ khi biết khóa mới có thể giải mã được văn bản mã. Khóa dùng trong DES có độ dài toàn bộ là 64 bit. Tuy nhiên chỉ có 56 bit thực sự được sử dụng; 8 bit còn lại chỉ dùng cho việc kiểm tra. Vì thế, độ dài thực tế của khóa chỉ là 56 bit.

Quá trình thực hiện:

* Tạo khóa con
* Mã hóa
* Giải mã

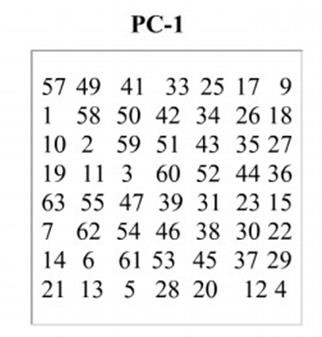
#### Quá trình tạo khóa con

Mười sáu vòng lặp DES chạy cùng thuật toán như nhau nhưng mỗi vòng lại sử dụng 16 khoa con khác nhau. Các khóa con đều được sinh ra từ khóa chính của DES bằng một thuật toán (giống nhau).



Hình 2.2. Sơ đồ tạo khóa con

K là xâu có độ dài 64 bit (8 byte), với mỗi byte bit 8 của byte được lấy ra dùng để kiểm tra phát hiện lỗi (các bít này ở vị trí 8,16,24…64) tạo ra chuỗi 56 bit. Sau khi bỏ các bit kiểm tra ta sẽ hoán vị chuỗi 56 bit, 2 bước trên thực hiện thông qua hoán vị ma trận PC -1.



Hình 2.3. Bảng hoán vị PC – 1

Chuỗi bit sau khi hoán vị qua PC – 1 được chia thành 2 phần:

C0 = 28 bit đầu

D0 = 28 bít cuối

Mỗi phần được thực hiện xử lý một cách độc lập: Ci :=LSi Ci-1

Di := LSi Di-1

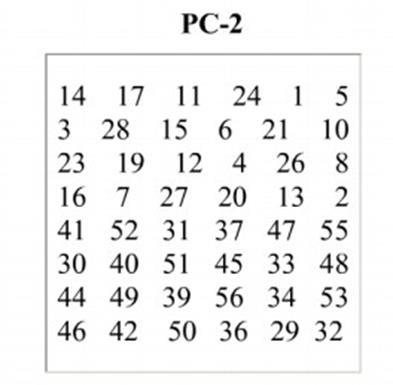
Với i chạy từ 0  i  16.

Trong đó: 𝐿𝑆𝑖 biểu diễn phép dịch vòng sang trái 1 hoặc 2 bit tùy thuộc vào i

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Vòng | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Số bít | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Hình 2.4. Bảng dịch bit theo vòng lặp

Sau khi được CiDi khóa Ki được xác định bằng các hoán vị theo ma trận PC-2

Ki = PC – 2(CiDi)

Hình 2.5. Bảng ma trận hoán vị PC – 2

#### Quá trình mã hóa

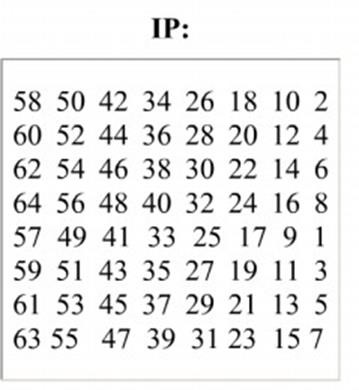
Ở quá trình này chia thành 3 giai đoạn

***Giai đoạn 1:***

Với bản rõ cho trước X, 1 xâu X’ được tạo ra bằng cách hoán vị các bit của X theo bảng hoán vị IP: X’ = IP(X) = L0R0

L0 : 32 bit đầu

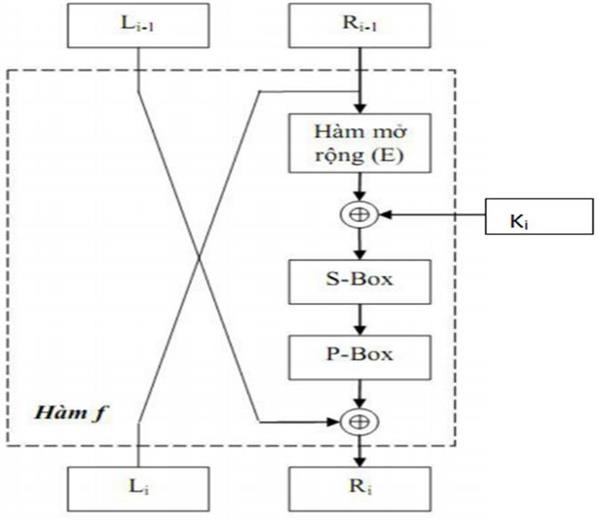
R0 : 32 bit cuối



Hình 2.6. Bảng hoán vị IP

***Giai đoạn 2:***

Tính 16 lần lặp theo 1 hàm xác định. Ta sẽ tính LiRi (1≤ i ≤16) theo quy tắc: Li = Ri-1 Ri = Li-1 xor F(Ri-1 ,Ki)



Hình 2.7. Sơ đồ một vòng mã hóa DES

* Hàm F có 2 biến vào là R và K
* Ri-1 được mở rộng từ 32 bit lên 48 bit thông hàm mở rộng E
  + - * 1. Hàm F

A diagram of a computer algorithm

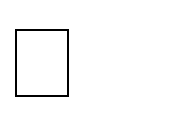
AI-generated content may be incorrect.

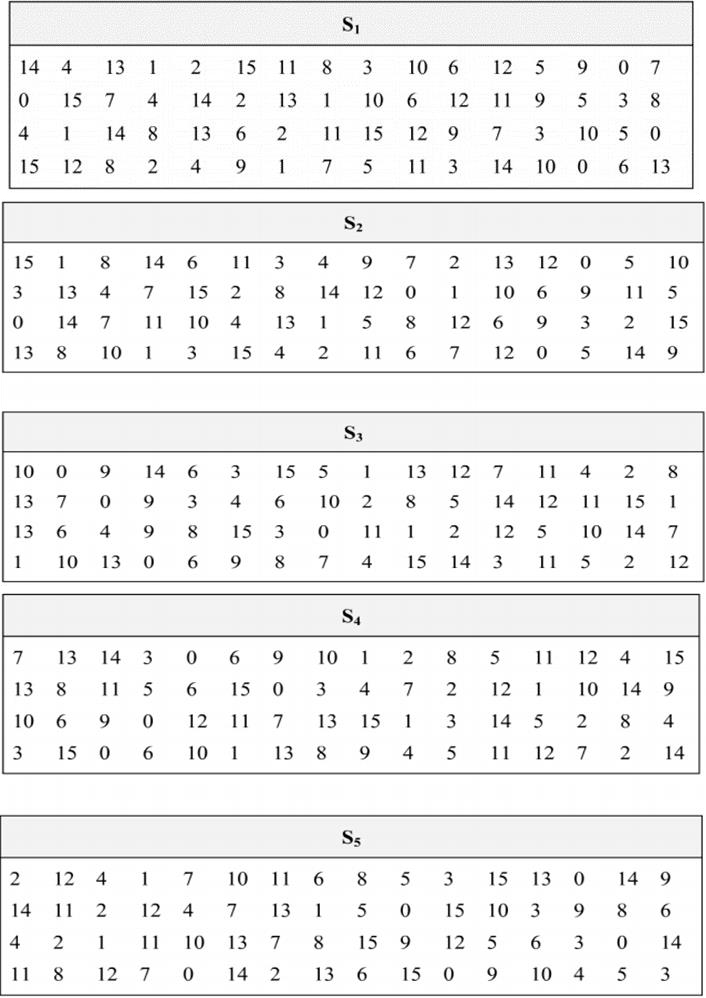
Hình 2.8. Sơ đồ hàm F

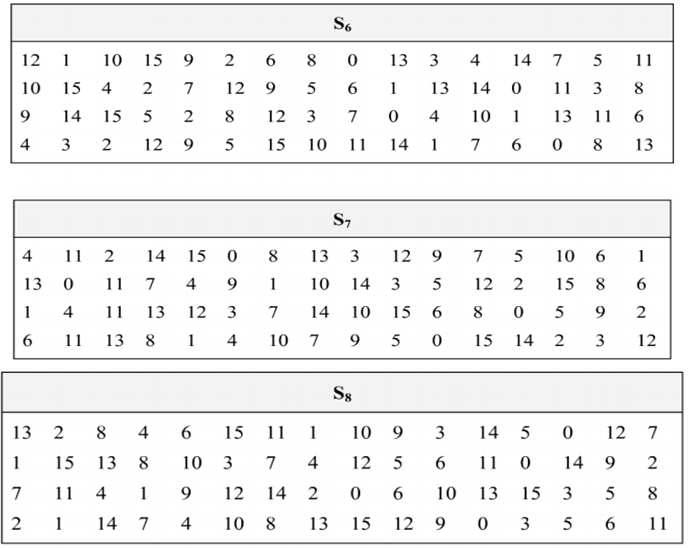
* Đầu vào hàm F có 2 biến:

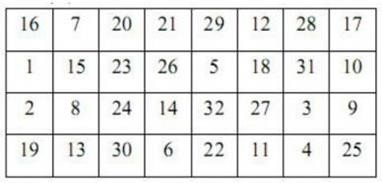
Biến 1: R là xâu bit có độ dài 32 bit

Biến 2: Khóa K là xâu bit có độ dài 48 bit.

* Đầu ra của F là xâu có độ dài 32 bit.
* Biến thứ nhất Ri-1 được mở rộng thành một xâu bit có độ dài 48 bit theo một hàm mở rộng cố định E. Thực chất hàm mở rộng E(Ri-1) là một hoán vị có lặp lại 16 bit của Ri-1.
* Tính E(Ri-1) XOR Ki và viết kết quả thành 8 xâu B1 B2 B3 B4 B5B6 B7 B8. Mỗi xâu 6 bit.
* Đưa khối 6 bit (Bi= b1b2b3b4b5b6) vào 8 bảng S1S2…S8 được gọi là các hộp S-Box. Mỗi hộp S-Box là một bảng ma trận 4\*16 cố định có các cột từ 0 đến 15 và các hạng từ 0 đến 3. Mới mỗi xâu 6 bit Bi = b1 b2 b3 b4 b5 b6ta tính được Si(Bi) bằng cách. Hai bit b1b6 xác định hang r của hộp S. (VD: 01B 1D thì vị chí là hàng 2). Bốn bit b2 b3 b4 b5xác định định cột c của hộp S (VD: 0101B 5D thì vị chí là cột 5). Khi đó Si(Bi) sẽ xác định được phần tử đứng ở vị trí (r,c), phần tử này đổi sang hệ nhị phân 4 bít. Như vậy 8 khối 6 bit B1 B2 B3 B4 B5B6 B7 B8 sẽ cho ra 8 khối 4 bit. Vậy khi ra khỏi S-Box ta sẽ được một xâu Ci 32 bit.
* Xâu Ci khi ra khỏi S-Box được hoán vị theo phép toàn hoán vị P (còn được gọi là P-Box). Kết quả P(C) là kết quả của hàm F(Ri-1,Ki).
* Bảng S (S – Box) và bảng hoán vị P (P – Box):





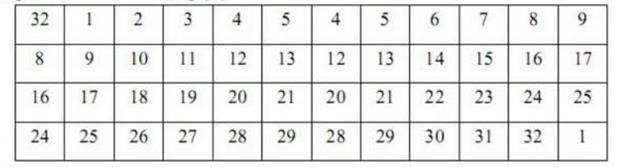
Hình 2.9. Bảng hộp S – Box

Hình 2.10. Bảng hoán vị P (P – Pox)

* + - * 1. Hàm mở rộng E

Hàm mở rộng E sẽ tăng độ dài dãy Ri từ 32bit lên 48bit bằng các thay đổi các thứ tự của các bit cũng như lặp lại các bit. Việc thực hiện này nhằm hai mục định:

* Làm độ dài của Ri bằng cỡ khóa K (48 bit) để thực hiện phép cộng modulo XOR.
* Cho kết quả dài hơn để có thế nén được trong suốt quá trình thay thế.

***Giai đoạn 3:***

Hình 2.11. Bảng mô tả hàm mở rộng E

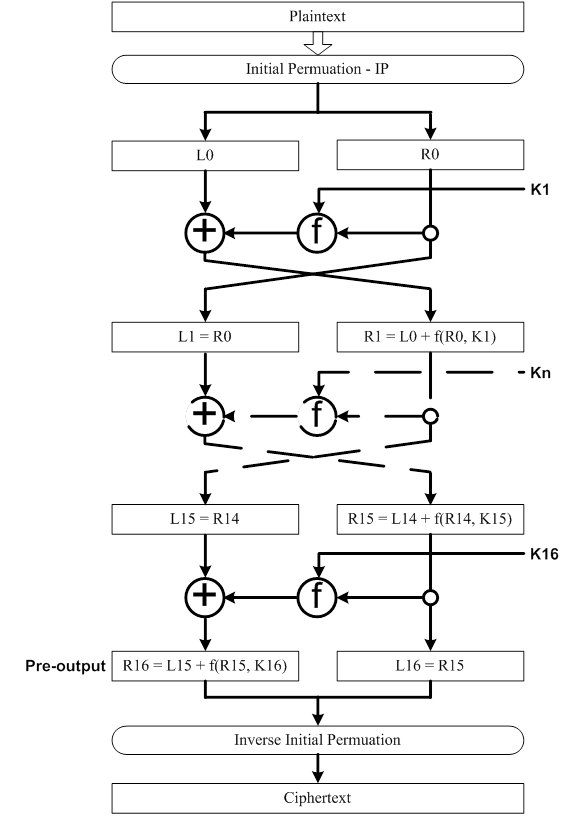
Áp dụng hoán vị ngược IP-1 cho xâu bit R16L16 ta thu được bản mã Y:

Y = IP – 1(R16L16)



Hình 2.12. Bảng hoán vị IP – 1

Toàn bộ quá trình mã hóa ta có thể mô tả:



Hình 2.13. Quá trình mã hóa

#### Quá trình giải mã

Do là một thuật toán đối xứng nên quá trình giải mã và mã hóa là giống nhau chỉ khác ở khóa Ki chạy từ 16 tới 1:

Li:= Ri-1

Ri := Li-1 XOR F(Ri-1 ,K16-i)

### Hàm tạo khóa con

|  |
| --- |
| private byte[][] GenerateSubKeys()  {  byte[][] subKeys = new byte[16][];  for (int i = 0; i < 16; i++)  {  subKeys[i] = new byte[6];  }  byte[] permutedKey = Permute(key, PC1);  byte[] C = new byte[4];  byte[] D = new byte[4];  for (int i = 0; i < 28; i++)  {  SetBit(C, i, GetBit(permutedKey, i));  SetBit(D, i, GetBit(permutedKey, i + 28));  }  for (int i = 0; i < 16; i++)  {  LeftShift(C, SHIFTS[i], 28);  LeftShift(D, SHIFTS[i], 28);  byte[] combined = new byte[7];  for (int j = 0; j < 28; j++)  {  SetBit(combined, j, GetBit(C, j));  SetBit(combined, j + 28, GetBit(D, j));  }  subKeys[i] = Permute(combined, PC2);  }  return subKeys;  } |

### Hàm mã hóa f(R,K)

|  |
| --- |
| private byte[] Feistel(byte[] right, byte[] subKey)  {  byte[] expanded = Permute(right, E);  byte[] xored = Xor(expanded, subKey);  byte[] substituted = new byte[4];  for (int i = 0; i < 8; i++)  {  int start = i \* 6;  int row = (GetBit(xored, start) << 1) | GetBit(xored, start + 5);  int col = (GetBit(xored, start + 1) << 3) | (GetBit(xored, start + 2) << 2) | (GetBit(xored, start + 3) << 1) | GetBit(xored, start + 4);  int val = SBOX[i][row, col];  for (int j = 0; j < 4; j++)  {  SetBit(substituted, i \* 4 + j, (val >> (3 - j)) & 1);  }  }  return Permute(substituted, P);  } |

### Hàm mã hóa và giải mã

#### Hàm mã hóa

|  |
| --- |
| public byte[] Encrypt(byte[] data)  {  int paddingLength = 8 - (data.Length % 8);  if (paddingLength == 0) paddingLength = 8;  byte[] paddedData = new byte[data.Length + paddingLength];  Array.Copy(data, paddedData, data.Length);  for (int i = data.Length; i < paddedData.Length; i++)  {  paddedData[i] = (byte)paddingLength;  }  byte[] result = new byte[paddedData.Length];  byte[] previousBlock = new byte[iv.Length];  Array.Copy(iv, previousBlock, iv.Length);  for (int i = 0; i < paddedData.Length; i += 8)  {  byte[] block = new byte[8];  Array.Copy(paddedData, i, block, 0, 8);  for (int j = 0; j < 8; j++)  block[j] ^= previousBlock[j];  byte[] encryptedBlock = ProcessBlock(block, true);  Array.Copy(encryptedBlock, 0, result, i, 8);  previousBlock = encryptedBlock;  }  return result;  } |

#### Hàm giải mã

|  |
| --- |
| public byte[] Decrypt(byte[] data)  {  if (data.Length % 8 != 0)  {  throw new ArgumentException("Encrypted data length must be a multiple of 8 bytes.");  }  byte[] result = new byte[data.Length];  byte[] previousBlock = new byte[iv.Length];  Array.Copy(iv, previousBlock, iv.Length);  for (int i = 0; i < data.Length; i += 8)  {  byte[] block = new byte[8];  Array.Copy(data, i, block, 0, 8);  byte[] decryptedBlock = ProcessBlock(block, false);  for (int j = 0; j < 8; j++)  decryptedBlock[j] ^= previousBlock[j];  Array.Copy(decryptedBlock, 0, result, i, 8);  Array.Copy(block, previousBlock, 8);  }  int paddingLength = result[result.Length - 1];  if (paddingLength > 0 && paddingLength <= 8)  {  Array.Resize(ref result, result.Length - paddingLength);  }  return result;  } |

#### Hàm trung gian

|  |
| --- |
| private byte[] ProcessBlock(byte[] block, bool encrypt)  {  byte[][] keys = GenerateSubKeys();  byte[] permuted = Permute(block, IP);  byte[] left = new byte[4];  byte[] right = new byte[4];  Array.Copy(permuted, 0, left, 0, 4);  Array.Copy(permuted, 4, right, 0, 4);  for (int i = 0; i < 16; i++)  {  byte[] temp = new byte[right.Length];  Array.Copy(right, temp, right.Length);  right = Xor(left, Feistel(right, keys[encrypt ? i : 15 - i]));  left = temp;  }  byte[] combined = new byte[8];  Array.Copy(right, 0, combined, 0, 4);  Array.Copy(left, 0, combined, 4, 4);  return Permute(combined, FP);  } |

### Các hàm hỗ trợ

#### Hàm hoán vị Permute()

|  |
| --- |
| private byte[] Permute(byte[] input, int[] table)  {  byte[] result = new byte[table.Length / 8];  for (int i = 0; i < table.Length; i++)  {  int pos = table[i] - 1;  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  int resultBytePos = i / 8;  int resultBitPos = 7 - (i % 8);  if ((input[bytePos] & (1 << bitPos)) != 0)  result[resultBytePos] |= (byte)(1 << resultBitPos);  }  return result;  } |

#### Hàm dịch bit

|  |
| --- |
| private static void LeftShift(byte[] data, int bits, int len)  {  byte[] temp = new byte[4];  for (int i = 0; i < len; i++)  {  SetBit(temp, i, GetBit(data, (i + bits) % len));  }  Array.Copy(temp, data, 4);  } |

#### Hàm Xor

|  |
| --- |
| private byte[] Xor(byte[] a, byte[] b)  {  byte[] result = new byte[Math.Min(a.Length, b.Length)];  for (int i = 0; i < result.Length; i++)  {  result[i] = (byte)(a[i] ^ b[i]);  }  return result;  } |

#### Hàm set bit

|  |
| --- |
| private static void SetBit(byte[] data, int pos, int value)  {  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  if (value != 0)  data[bytePos] |= (byte)(1 << bitPos);  else  data[bytePos] &= (byte)~(1 << bitPos);  } |

#### Hàm get bit

|  |
| --- |
| private static int GetBit(byte[] data, int pos)  {  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  return (data[bytePos] >> bitPos) & 1;  } |

# TỔNG KẾT

## Kết quả đạt được

### Về mặt lý thuyết

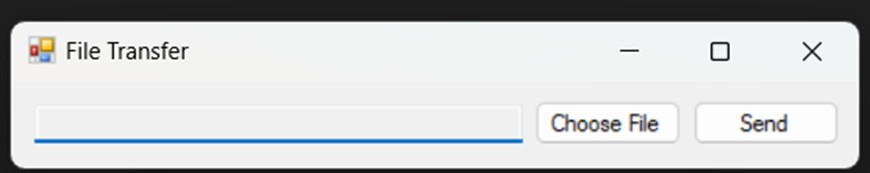
Sau khi hoàn thành đề tài, nhóm đã thu được cho mình những kiến thức mới, đồng thời hiểu được thêm một số thuật toán mã hóa và giải mã mà sau này có thể áp dụng vào công việc.

* + - * Có thêm kiến thức về mật mã học với các khái niệm cơ bản của mật mã học, bao gồm thuật toán mã hóa, giải mã, khóa, v.v. Từ đó hiểu được tại sao mật mã học là cần thiết để bảo vệ thông tin cá nhân và tài sản của các tổ chức.
      * Hiểu về thuật toán mã hóa DES.
      * Được tiếp xúc với ngôn ngữ lập trình mới là C#. Khi hoàn thành đề tài này, sẽ có khả năng viết các chương trình mã hóa và giải mã các file dữ liệu bằng DES sử dụng ngôn ngữ C#.
      * Khả năng viết chương trình mã hóa và giải mã sẽ được tăng lên.
      * Kiến thức và kỹ năng bạn học được từ đề tài này có thể được áp dụng vào các dự án thực tế, trong đó việc bảo vệ dữ liệu là một yếu tố quan trọng.

### Chương trình mã hóa và giải mã

* + - * Giao diện của chương trình mã hóa và giải file bằng DES

Sau khi nghiên cứu và triển khai thành công đề tài, nhóm đã tiếp tục phát triển và bổ sung thêm giao diện để người dùng có thể thuận lợi quan sát và sử dụng.

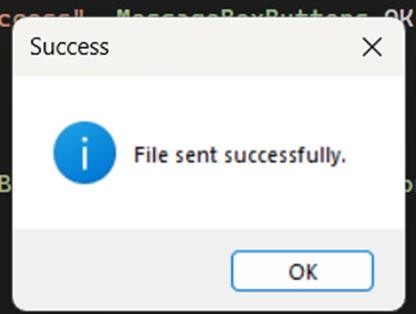


Hình 3.1. Giao diện ứng dụng mã hóa DES và gửi file bằng C#



Hình 3.2. Giao diện nhận file và giải file mã hóa

Quá trình sử dụng ứng dụng Mã hóa và Giải mã DES bằng C# đã được nhóm phát triển một cách tối giản nhất về cách dùng:

* + - * Khi bắt đầu cần chạy hai chương trình rồi bấm “StartListening” ở chương trình nhận file để lắng nghe thông tin từ chương trình gửi.
      * Sau đó lựa chọn file ký tự cần mã hóa ở chương trình gửi file rồi nhấn nút “Send”, nếu gửi thành công sẽ có thông báo như sau:

Hình 3.3. Hình thông báo gửi file thành công

* + - * Nếu gặp lỗi thì sẽ hiện thông báo lỗi
      * Sau khi chương trình gửi thành công ở bên máy nhận chương trình sẽ hiển thị như sau:



Hình 3.4. Hình ảnh đã nhận được file mã hóa ở chương trình nhận

Khi chương trình nhận đã hiện thị dòng chữ “File received. Click Download to save.” (đã nhận được file mã hóa), chúng ta bấm “Download” để tải file mã hóa đã nhận được vào một thư mục nào đó.

Sau đó ta bấm “Upload and Encrypt” để đẩy file đã mã hóa lên để chương trình giải mã, sau khi giải mã chương trình sẽ tự động lưu xuống file đã được mã hóa ở cùng thư mục với file mã hóa.

## Đánh giá

### Ưu điểm

* + - * Chương trình hoạt động chính xác 100%
      * Tốc độ mã hóa và giải mã là rất nhanh, đơn vị là miliseconds và sẽ tùy thuộc một phần vào độ lớn của file ký tự
      * Chương trình mã hóa và giải mã dễ sử dụng

### Nhược điểm

* + - * Trong quá trình nghiên cứu đề tài thì còn khó khăn trong việc hiểu rõ và sâu vào thuật toán mã hóa và giải mã, làm cho tốc độ hoàn thiện đề tài là còn chưa nhanh.
      * Theo nghiên cứu thì DES là một thuật toán mã hóa cổ điển và đã bị đánh bại bởi các kỹ thuật tấn công như brute force, phân tích miền tần số, phân tích hiệu suất và xem trước.
      * Chương trình này chỉ mã hóa và giải mã các tệp tin trực tiếp từ hệ thống tệp tin của nó. Nó không có chức năng để đọc các tệp tin mã hóa từ các nguồn bên ngoài như email hoặc đám mây, điều này có thể giới hạn khả năng sử dụng chương trình.

## Phương hướng phát triển

* Nghiên cứu và tối ưu hóa mã nguồn để tăng tốc độ xử lý mã hóa và giải mã.
* Thêm tính năng mã hóa và giải mã các loại file khác: Ngoài các file dữ liệu thông thường, chương trình có thể được phát triển để mã hóa và giải mã các loại file khác như hình ảnh, âm thanh, video, vv.
* Phát triển chương trình để mã hóa và giải mã theo các thuật toán khác như AES, RSA, Blowfish, vv.
* Chương trình có thể được phát triển để chống sao chép bằng cách thêm tính năng mã hóa các thông tin đặc biệt trong file, giúp ngăn chặn các file được sao chép hoặc sử dụng trái phép.
* Phát triển thêm để chương trình yêu cầu người dùng xác thực trước khi mã hóa hoặc giải mã các file, giúp đảm bảo an toàn thông tin.

# PHỤ LỤC

Code chương trình:

|  |
| --- |
| namespace DESLibrary  {  public class DES  {  private static readonly int[] IP = new int[] {  58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,  62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,  57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,  61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7  };  private static readonly int[] FP = new int[] {  40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,  38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,  36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,  34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25  };  private static readonly int[] PC1 = new int[] {  57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,  10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36,  63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,  14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4  };  private static readonly int[] PC2 = new int[] {  14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10,  23, 19, 12, 4, 26, 8, 16, 7, 27, 20, 13, 2,  41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,  44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32  };  private static readonly int[] E = new int[] {  32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9,  8, 9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,  16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,  24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1  };  private static readonly int[] P = new int[] {  16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23, 26,  5, 18, 31, 10, 2, 8, 24, 14, 32, 27, 3, 9,  19, 13, 30, 6, 22, 11, 4, 25  };  private static readonly int[][,] SBOX = new int[8][,] {  new int[,] { {14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7},  {0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8},  {4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0},  {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13} },  new int[,] { {15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10},  {3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5},  {0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15},  {13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9} },  new int[,] { {10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8},  {13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1},  {13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7},  {1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12} },  new int[,] { {7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15},  {13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9},  {10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4},  {3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14} },  new int[,] { {2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9},  {14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6},  {4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14},  {11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3} },  new int[,] { {12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11},  {10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8},  {9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6},  {4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13} },  new int[,] { {4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1},  {13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6},  {1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2},  {6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12} },  new int[,] { {13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7},  {1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2},  {7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8},  {2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11} }  };  private static readonly int[] SHIFTS = new int[16] { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };  private readonly byte[] key;  private readonly byte[] iv;  public DES(string keyString, byte[] iv = null)  {  this.key = Encoding.UTF8.GetBytes(keyString.PadRight(8, '\0').Substring(0, 8));  this.iv = iv ?? new byte[8]; // Nếu iv là null, khởi tạo mảng 8 byte toàn 0  }  public byte[] Encrypt(byte[] data)  {  int paddingLength = 8 - (data.Length % 8);  if (paddingLength == 0) paddingLength = 8;  byte[] paddedData = new byte[data.Length + paddingLength];  Array.Copy(data, paddedData, data.Length);  for (int i = data.Length; i < paddedData.Length; i++)  {  paddedData[i] = (byte)paddingLength;  }  byte[] result = new byte[paddedData.Length];  byte[] previousBlock = new byte[iv.Length];  Array.Copy(iv, previousBlock, iv.Length);  for (int i = 0; i < paddedData.Length; i += 8)  {  byte[] block = new byte[8];  Array.Copy(paddedData, i, block, 0, 8);  for (int j = 0; j < 8; j++)  block[j] ^= previousBlock[j]; // XOR với previousBlock  byte[] encryptedBlock = ProcessBlock(block, true);  Array.Copy(encryptedBlock, 0, result, i, 8);  previousBlock = encryptedBlock; // Cập nhật previousBlock  }  return result;  }  public byte[] Decrypt(byte[] data)  {  if (data.Length % 8 != 0)  {  throw new ArgumentException("Encrypted data length must be a multiple of 8 bytes.");  }  byte[] result = new byte[data.Length];  byte[] previousBlock = new byte[iv.Length];  Array.Copy(iv, previousBlock, iv.Length);  for (int i = 0; i < data.Length; i += 8)  {  byte[] block = new byte[8];  Array.Copy(data, i, block, 0, 8);  byte[] decryptedBlock = ProcessBlock(block, false);  for (int j = 0; j < 8; j++)  decryptedBlock[j] ^= previousBlock[j];  Array.Copy(decryptedBlock, 0, result, i, 8);  Array.Copy(block, previousBlock, 8);  }  int paddingLength = result[result.Length - 1];  if (paddingLength > 0 && paddingLength <= 8)  {  Array.Resize(ref result, result.Length - paddingLength);  }  return result;  }  private byte[] ProcessBlock(byte[] block, bool encrypt)  {  byte[][] keys = GenerateSubKeys(); // Lấy 16 khóa con  byte[] permuted = Permute(block, IP); // Hoán vị ban đầu (IP)  byte[] left = new byte[4];  byte[] right = new byte[4];  Array.Copy(permuted, 0, left, 0, 4); // Lấy nửa bên trái (L0)  Array.Copy(permuted, 4, right, 0, 4); // Lấy nửa bên phải (R0)  for (int i = 0; i < 16; i++)  {  byte[] temp = new byte[right.Length];  Array.Copy(right, temp, right.Length);  right = Xor(left, Feistel(right, keys[encrypt ? i : 15 - i])); // R(i+1) = L(i) XOR f(R(i), K(i))  left = temp; // L(i+1) = R(i)  }  byte[] combined = new byte[8];  Array.Copy(right, 0, combined, 0, 4); // Ghép R16  Array.Copy(left, 0, combined, 4, 4); // Ghép L16  return Permute(combined, FP); // Hoán vị cuối (FP)  }  private byte[][] GenerateSubKeys()  {  byte[][] subKeys = new byte[16][];  for (int i = 0; i < 16; i++)  {  subKeys[i] = new byte[6];  }  byte[] permutedKey = Permute(key, PC1); // Hoán vị khóa ban đầu bằng PC1  byte[] C = new byte[4];  byte[] D = new byte[4];  for (int i = 0; i < 28; i++)  {  SetBit(C, i, GetBit(permutedKey, i)); // Chia thành C0 (28 bit)  SetBit(D, i, GetBit(permutedKey, i + 28)); // Chia thành D0 (28 bit)  }  for (int i = 0; i < 16; i++)  {  LeftShift(C, SHIFTS[i], 28); // Dịch trái C  LeftShift(D, SHIFTS[i], 28); // Dịch trái D  byte[] combined = new byte[7];  for (int j = 0; j < 28; j++)  {  SetBit(combined, j, GetBit(C, j));  SetBit(combined, j + 28, GetBit(D, j));  }  subKeys[i] = Permute(combined, PC2); // Tạo khóa con bằng PC2  }  return subKeys;  }  private byte[] Permute(byte[] input, int[] table)  {  byte[] result = new byte[table.Length / 8];  for (int i = 0; i < table.Length; i++)  {  int pos = table[i] - 1;  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  int resultBytePos = i / 8;  int resultBitPos = 7 - (i % 8);  if ((input[bytePos] & (1 << bitPos)) != 0)  result[resultBytePos] |= (byte)(1 << resultBitPos);  }  return result;  }  private static void SetBit(byte[] data, int pos, int value)  {  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  if (value != 0)  data[bytePos] |= (byte)(1 << bitPos);  else  data[bytePos] &= (byte)~(1 << bitPos);  }  private static int GetBit(byte[] data, int pos)  {  int bytePos = pos / 8;  int bitPos = 7 - (pos % 8);  return (data[bytePos] >> bitPos) & 1;  }  private static void LeftShift(byte[] data, int bits, int len)  {  byte[] temp = new byte[4];  for (int i = 0; i < len; i++)  {  SetBit(temp, i, GetBit(data, (i + bits) % len));  }  Array.Copy(temp, data, 4);  }  private byte[] Feistel(byte[] right, byte[] subKey)  {  byte[] expanded = Permute(right, E); // Mở rộng R từ 32-bit lên 48-bit  byte[] xored = Xor(expanded, subKey); // XOR với khóa con  byte[] substituted = new byte[4];  for (int i = 0; i < 8; i++)  {  int start = i \* 6;  int row = (GetBit(xored, start) << 1) | GetBit(xored, start + 5); // Lấy 2 bit đầu cuối làm hàng  int col = (GetBit(xored, start + 1) << 3) | (GetBit(xored, start + 2) << 2) |  (GetBit(xored, start + 3) << 1) | GetBit(xored, start + 4); // Lấy 4 bit giữa làm cột  int val = SBOX[i][row, col]; // Tra bảng S-box  for (int j = 0; j < 4; j++)  {  SetBit(substituted, i \* 4 + j, (val >> (3 - j)) & 1); // Ghi giá trị 4-bit từ S-box  }  }  return Permute(substituted, P); // Hoán vị P  }  private byte[] Xor(byte[] a, byte[] b)  {  byte[] result = new byte[Math.Min(a.Length, b.Length)];  for (int i = 0; i < result.Length; i++)  {  result[i] = (byte)(a[i] ^ b[i]);  }  return result;  }  }  } |

Đường dẫn mã nguồn: <https://github.com/chiennc09/DES-CSATTT>

# TÀI LIỆU THAM KHẢO

[1]. *Bài giảng An toàn và Bảo mật thông tin* (2008), Trường Đại học Nha Trang, Nha Trang.

[2]. Hồ Quang Bửu (2013). *Luận án về một phương pháp xây dựng hàm băm cho việc xác thực trên cơ sở ứng dụng thuật toán mã hóa đối xứng*. Luận án tiến sỹ ký thuật, Học viện Công nghệ bưu chính viễn thông.

[3]. Nguyễn Quân (2019). [DES] *Thuật toán mã hóa và giải mã DES*, 1/4/2025 [https://nguyenquanicd.blogspot.com/2017/08/background-thuat-toan-ma-hoa-va-](https://nguyenquanicd.blogspot.com/2017/08/background-thuat-toan-ma-hoa-va-giai-ma.html) [giai-ma.html](https://nguyenquanicd.blogspot.com/2017/08/background-thuat-toan-ma-hoa-va-giai-ma.html)>.