ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Mật mã an ninh mạng

BÁO CÁO BÀI TẬP LỚN 1

Sinh viên thực hiện: **Nguyễn Trần Minh Đăng**

MSSV: **1711014**

Giáo viên hướng dẫn: Nguyễn Hữu Hiếu

Tháng 5 năm 2020

Mục lục

1 Tóm tắt

Mã hóa với mục đích tạo ra một lớp bảo vệ cho dữ liệu để người khác không đọc được, ngoại trừ những ai được phép đọc. Mã hóa sử dụng thuật toán và khóa để biến đổi dữ liệu từ hình thức đơn giản rõ ràng (plain hay cleartext), làm biến dữ liệu sang hình thức mật mã (code hay ciphertext). Chỉ có những ai có thông tin giải mã thì mới giải mã được và đọc được dữ liệu. Chúng ta biết rằng các dịch vụ lưu trữ đám mây phổ biến hiện nay (như DropBox hay SugarSync) đã mã hóa dữ liệu của chúng ta, bảo vệ dữ liệu khi truyền và trong khi được lưu trên các máy chủ của họ. Tuy nhiên, cũng chính những dịch vụ đó nắm giữ chìa khóa giải mã, có nghĩa là họ có thể giải mã các tập tin của bạn mà bạn không hề hay biết. Nếu bạn có bất kỳ tập tin thực sự nhạy cảm nào trong dịch vụ lưu trữ đám mây của mình, hãy sử dụng lớp mã hóa thứ hai để giữ cho chúng an toàn khỏi những con mắt tò mò.

Trong assigment này, tôi sẽ sử dụng lớp mã hóa thứ hai để giữ cho tập tin trong quá trình truyền hay gửi dữ liệu được an toàn. Cụ thể là xây dựng chương trình mã hóa và giải mã các tập tin sử dụng các giải thuật mã hóa DES, RSA, AES,...

2 Giới thiệu

- Chương trình tích hợp 2 giải thuật mã hóa bao gồm giải thuật mã hóa DES và giải thuật mã hóa AES.
- Chương trình hỗ trợ mã hóa một tập tin với định dạng bất kì như: hình ảnh, word, txt,..
- Quá trình mã hóa: nhận input là tập tin bất kì và tập tin text chứa chìa khóa mã hóa (encryption key) và một số option khác (nếu cần), output là tập tin hay thư mục chứa dữ liệu đã được mã hóa.
- Quá trình giải mã: nhận input là tập tin hay thư mục chứa dữ liệu đã được mã hóa và tập tin text chứa chìa khóa giải mã (decryption key) và một số option khác (nếu cần), output chương trình là tập tin được giãi mã thành công.
- Sự dụng hàm hash MD5 để chứng minh tính toàn vẹn giữa tập tin gốc ban đầu được chọn và tập tin output của quá trình giải mã.
- Chương trình sử dụng ngôn ngữ: **C#**
- Thư viện mã hóa:**System.Security.Cryptography**
- Chương trình chưa thực hiện được các tính năng nâng cao như file bài tập lớn yêu cầu.

3 Thân bài

3.1 Cơ sở lý thuyết

3.1.1 Giải thuật mã hóa DES

3.1.1.1 Giới thiệu về DES

DES (viết tắt của Data Encryption Standard, hay Tiêu chuẩn Mã hóa Dữ liệu) là một phương pháp mật mã hóa được FIPS (Tiêu chuẩn Xử lý Thông tin Liên bang Hoa Kỳ) chọn làm chuẩn chính thức vào năm 1976. Sau đó chuẩn này được sử dụng rộng rãi trên phạm vi thế giới. Ngay từ đầu, thuật toán của nó đã gây ra rất nhiều tranh cãi, do nó bao gồm các thành phần thiết kế mật, độ dài khóa tương đối ngắn, và các nghi ngờ về cửa sau để Cơ quan An ninh quốc gia Hoa Kỳ (NSA) có thể bẻ khóa. Do đó, DES đã được giới nghiên cứu xem xét rất kỹ lưỡng, việc này đã thúc đẩy hiểu biết hiện đại về mật mã khối (block cipher) và các phương pháp thám mã tương ứng.

3.1.1.2 Mô tả thuật toán

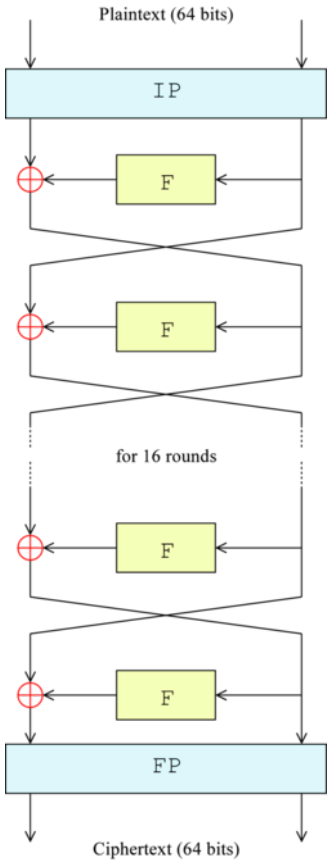

Figure 1: *Cấu trúc giải thuật Feistel dùng trong DES*

DES là thuật toán mã hóa khối: nó xử lý từng khối thông tin của bản rõ có độ dài xác định và biến

đổi theo những quá trình phức tạp để trở thành khối thông tin của bản mã có độ dài không thay đổi. Trong trường hợp của DES, độ dài mỗi khối là 64 bit. 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 bi.

Giống như các thuật toán mã hóa khối khác, khi áp dụng cho các văn bản dài hơn 64 bit, DES phải được dùng theo một phương pháp nào đó. Trong tài liệu FIPS-81 đã chỉ ra một số phương pháp, trong đó có một phương pháp dùng cho quá trình nhận thực. Một số thông tin thêm về những cách sử dụng DES được miêu tả trong tài liệu FIPS-74.

Tổng thể:

- Cấu trúc tổng thể của thuật toán được thể hiện ở Figure 1: có 16 chu trình giống nhau trong quá trình xử lý. Ngoài ra còn có hai lần hoán vị đầu và cuối (Initial and final permutation - IP EP). Hai quá trình này có tính chất đối nhau (Trong quá trình mã hóa thì IP trước EP, khi giải mã thì ngược lại). IP và EP không có vai trò xét về mật mã học và việc sử dụng chúng chỉ có ý nghĩa đáp ứng cho quá trình đưa thông tin vào và lấy thông tin ra từ các khối phần cứng có từ thập niên 1970. Trước khi đi vào 16 chu trình chính, khối thông tin 64 bit được tách làm hai phần 32 bit và mỗi phần sẽ được xử lý tuần tự (quá trình này còn được gọi là mạng Feistel).
- Cấu trúc của thuật toán (mạng Feistel) đảm bảo rằng quá trình mã hóa và giải mã diễn ra tương tự. Điểm khác nhau chỉ ở chỗ các khóa con được sử dụng theo trình tự ngược nhau. Điều này giúp cho việc thực hiện thuật toán trở nên đơn giản, đặc biệt là khi thực hiện bằng phần cứng.
- Kí hiệu sau: $\oplus$ thể hiện phép toán XOR. Hàm F làm biến đổi một nửa của khối đang xử lý với một khóa con. Đầu ra sau hàm F được kết hợp với nửa còn lại của khối và hai phần được tráo đổi để xử lý trong chu trình kế tiếp. Sau chu trình cuối cùng thì 2 nửa không bị tráo đổi; đây là đặc điểm của cấu trúc Feistel khiến cho quá trình mã hóa và giải mã trở nên giống nhau.

Hàm Feistel: Hàm F, hoạt động trên khối 32 bit và bao gồm bốn giai đoạn:

1. *Mở rộng*: 32 bit đầu vào được mở rộng thành 48 bit sử dụng thuật toán hoán vị mở rộng (expansion permutation) với việc nhân đôi một số bit. Giai đoạn này được ký hiệu là E trong sơ đồ.
2. *Trộn khóa*: 48 bit thu được sau quá trình mở rộng được XOR với khóa con. Mười sáu khóa con 48 bit được tạo ra từ khóa chính 56 bit theo một chu trình tạo khóa con (key schedule) miêu tả ở phần sau.
3. *Thay thế*: 48 bit sau khi trộn được chia làm 8 khối con 6 bit và được xử lý qua hộp thay thế S-box. Đầu ra của mỗi khối 6 bit là một khối 4 bit theo một chuyển đổi phi tuyến được thực hiện bằng một bảng tra. Khối S-box đảm bảo phần quan trọng cho độ an toàn của DES. Nếu không có S-box thì quá

trình sẽ là tuyến tính và việc thám mã sẽ rất đơn giản. *Hoán vị*: Cuối cùng, 32 bit thu được sau S-box sẽ được sắp xếp lại theo một thứ tự cho trước (còn gọi là P-box).

Quá trình luân phiên sử dụng S-box và sự hoán vị các bít cũng như quá trình mở rộng đã thực hiện được tính chất gọi là sự xáo trộn và khuyếch tán (confusion and diffusion). Đây là yêu cầu cần có của một thuật toán mã hoá được Claude Shannon phát hiện trong những năm 1940.

Quá trình tạo khóa con

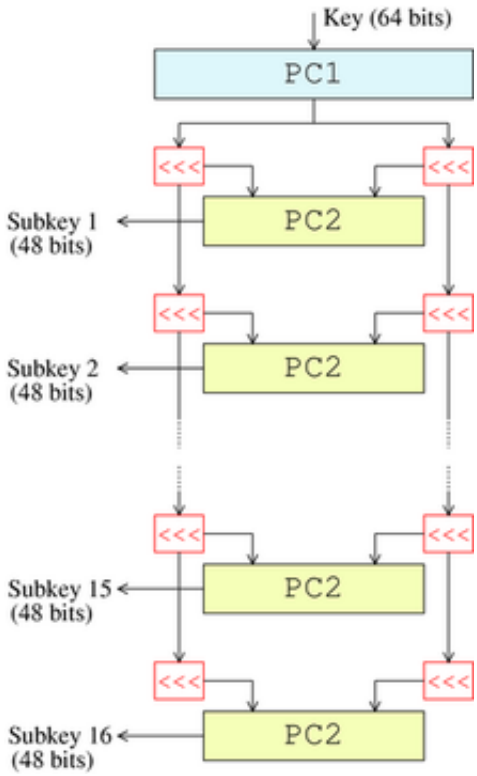

Figure 2: *Quá trình tạo khóa con dùng DES*

Figure 2 mô tả thuật toán tạo khóa con cho các chu trình. Đầu tiên, từ 64 bit ban đầu của khóa, 56 bit được chọn (Permuted Choice 1, hay PC-1); 8 bit còn lại bị loại bỏ. 56 bit thu được được chia làm hai phần bằng nhau, mỗi phần được xử lý độc lập. Sau mỗi chu trình, mỗi phần được dịch đi 1 hoặc 2 bit (tùy thuộc từng chu trình). Các khóa con 48 bit được tạo thành bởi thuật toán lựa chọn 2 (Permuted Choice 2, hay PC-2) gồm 24 bit từ mỗi phần. Quá trình dịch bit (được ký hiệu là "«<" trong sơ đồ) khiến cho các khóa con sử dụng các bit khác nhau của khóa chính; mỗi bit được sử dụng trung bình ở 14 trong tổng số 16 khóa con.

3.1.2 Giải thuật mã hóa AES

3.1.2.1 Giới thiệu về AES

Tiêu chuẩn Advanced Encryption Standard(AES)-tiêu chuẩn mã hóa tiên tiến- là một thuật toán tiêu chuẩn của chính phủ Hoa kỳ nhằm mã hóa và giải mã dữ liệu do Viện Tiêu chuẩn và Công nghệ quốc gia Hoa Kỳ phát hành ngày 26/11/2001.

AES là một thuật toán " mã hóa khối "(block cipher) ban đầu được tạo ra bởi hai nhà mật mã học người Bỉ là Joan Daemen và Vincent Rijmen. Kể từ khi được công bố là một tiêu chuẩn, AES trở thành một trong những thuật toán phổ biến nhất sử dụng khóa mã đối xứng để mã hóa và giải mã.

3.1.2.2 Mô tả thuật toán

AES là một thuật toán mã hóa khối đối xứng với độ dài là 128 bít, 192 bít, 256 bít tương ứng gọi là AES-128, AES-192, AES-256. AES-128 sử dụng 10 vòng, AES-192 sử dụng 12 vòng và AES-256 sử dụng 14 vòng.

Quá trình tạo khóa con

Phép mã hóa

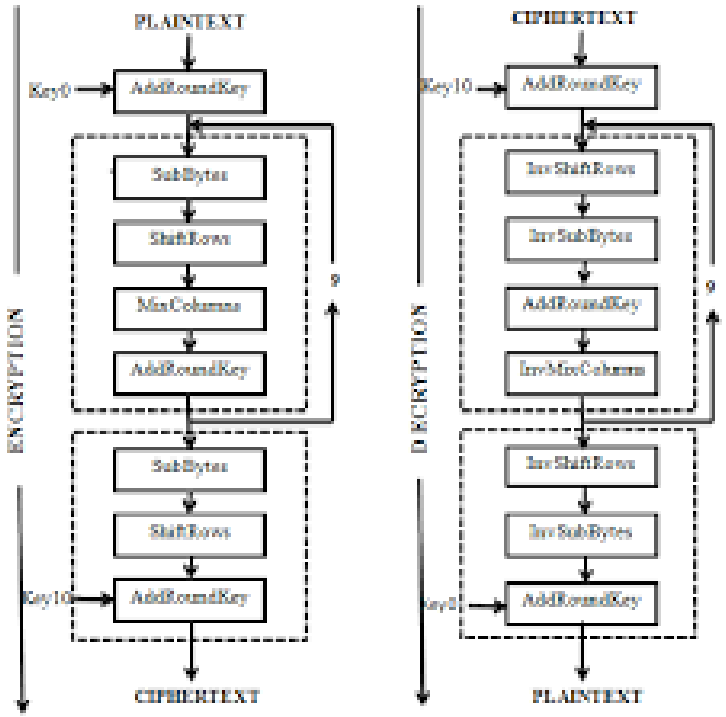

Figure 3: *Quá trình mã hóa/giải mã dùng AES*

Tại thời điểm bắt đầu phép mã hóa, đầu vào được sao chép vào mảng trạng thái sử dụng các quy ước. Sau phép cộng khóa vòng khởi đầu, mảng trạng thái được biến đổi bằng cách thực hiện một hàm vòng liên

tiếp với số vòng lặp là 10,12 hoặc 14 (tương ứng với độ dài khóa) vòng cuối cùng khác biệt không đáng kể với các vòng đầu tiên. Trạng thái cuối cùng được chuyển thành đầu ra.Hàm vòng được tham số hóa bằng cách sử dụng một lược đồ khóa- mảng một chiều chứa các từ 4 byte nhận từ phép mở rộng khóa. Phép biến đổi cụ thể gồm SubBytes(), ShiftRows(), MixColums() và AddRoundkey() dùng để xử lý trạng thái.

- SubBytes(): Phép biến đổi dùng trong phép mã hóa áp dụng lên trạng thái (kết quả mã hóa trung gian, được mô tả dưới dạng một mảng chữ nhật của các byte) sử dụng một bảng thay thế byte phi tuyến (Hộp S- bảng thay thế phi tuyến được sử dụng trong một phép thay thế byte và trong quy trình mở rộng khóa, nhằm thực hiện một phép thay thê 1-1 đối với giá trị mỗi byte) trên mỗi byte trạng thái một cách độc lập.
- ShiftRows(): Phép biến đổi dùng trong phép mã hóa áp dụng thực hiện bằng cách chuyển dịch các vòng ba hàng cuối của trạng thái theo số lượng byte các ofset khác nhau.
- MixColumns(): Phép biến đổi trong phép mã hóa thực hiện bằng cách lấy tất cả các cột trạng thái trộn với dữ liệu của chúng (một cách độc lập nhau) để tạo ra các cột mới.
- AddRoundKey(): Phép biến đổi dùng trong phép mã hóa và giải mã. Trong đó, một khóa vòng (các giá trị sinh ra từ khóa mã bằng quy trình mở rộng khóa) được thêm vào trạng thái bằng phép toán XOR. Độ dài của khóa vòng bằng độ dài của trạng thái.

Mở rộng khóa: Thuật toán AES nhận vào một mã K và thực hiện phép mở rộng khóa để tạo ra một lược đồ khóa. Phép mở rộng khóa tạo ra tổng số Nb(Nr+1) từ(với Nb độ dài khối và Nr số vòng). Thuật toán yêu cầu một tập khởi tạo gồm Nb từ và mỗi trong số nr vòng đòi hỏi Nb từ làm dữ liệu khóa đầu vào. Lược đồ khóa kết quả là một mảng tuyến tính các từ 4 byte.

Phép giải mã: Các phép biến đổi trong phép mã hóa có thể được đảo ngược và sau đó thực hiện theo chiều ngược lại nhằm tạo ra phép giải mã trực tiếp của thuật toán AES. Các phép biến đổi sử dụng trong phép giải mã gồm: InvSubButytes(), InvShiftRows(), InvMixColumns() và AddRoundKey().

- InvSubBytes(): Là nghịch đảo của phép SubBytes, trong đó sử dụng một hộp-S nghịch đảo áp dụng cho mỗi byte của trạng thái
- InvShiftRows(): Là phép biến đổi ngược của ShiftRows(). Các byte trong ba từ cuối của trạng thái được dịch vòng theo số byte khác nhau. Ở hàng đầu tiên (r=0) không thực hiện phép dịch chuyển, ba hàng dưới cùng được dịch vòng Nb-shift(r,Nb) byte.
- InvMixColumns(): Là phép ngược của MixColumns(). Nó thap tác theo từng cột của trạng thái, xem mỗi cột như một đa thức bốn hạng tử.

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

3.1.3 Hàm hash MD5

3.1.3.1 Giới thiệu về MD5

MD5 (Message-Digest algorithm 5) là một hàm băm để mã hóa với giá trị băm là 128bit. Từng được xem là một chuẩn trên Internet, MD5 đã được sữ dụng rông rải trong các chương trình an ninh mạng, và cũng thường được dùng để kiểm tra tính nguyên vẹn của tập tin.

MD5 được thiết kế bởi Ronald Rivest vào năm 1991 để thay thế cho hàm băm trước đó, MD4 (cũng do ông thiết kế, trước đó nữa là MD2)

3.1.3.2 Nguyên tắc hoạt động

Thuật giải

MD5 biến đổi một thông điệp có chiều dài bất kì thành một khối có kích thước cố định 128 bits. Thông điệp đưa vào sẻ được cắt thành các khối 512 bits. Thông điệp được đưa vào bộ đệm để chiều dài của nó sẻ chia hết cho 512.

Bộ đệm hoạt động như sau:

- Trước tiên nó sẻ chèn bit 1 vào cuối thông điệp.
- Tiếp đó là hàng loạt bit Zero cho tới khi chiều dài của nó nhỏ hơn bội số của 512 một khoảng 64 bit.
- Phần còn lại sẻ được lấp đầy bởi một số nguyên 64 bit biểu diễn chiều dài ban đầu của thông điệp.

Thuật toán chính của MD5 hoạt động trên một bộ 128 bit. Chia nhỏ nó ra thành 4 từ 32 bit, kí hiệu là A,B,C và D. Các giá trị này là các hằng số cố định. Sau đó thuật toán chính sẻ luân phiên hoạt động trên các khối 512 bit. Mỗi khối sẻ phối hợp với một bộ. Quá trình xữ lý một khối thông điệp bao gồm 4 bước tương tự nhau, gọi là vòng (“round”). Mỗi vòng lại gồm 16 quá trình tương tự nhau dựa trên hàm một chiều F, phép cộng module và phép xoay trái…

Hình bên dưới mô tả một quá trình trong một vòng. Có 4 hàm một chiều F có thể sử dụng. Mỗi vòng sử dụng một hàm khác nhau.

Hàm băm MD5 (còn được gọi là hàm tóm tắt thông điệp - message degests) sẻ trả về một chuỗi số thập lục phân gồm 32 số liên tiếp.Thậm chỉ chỉ cần một thay đổi nhỏ cũng làm thay đổi hoàn toàn kết quả trả về.

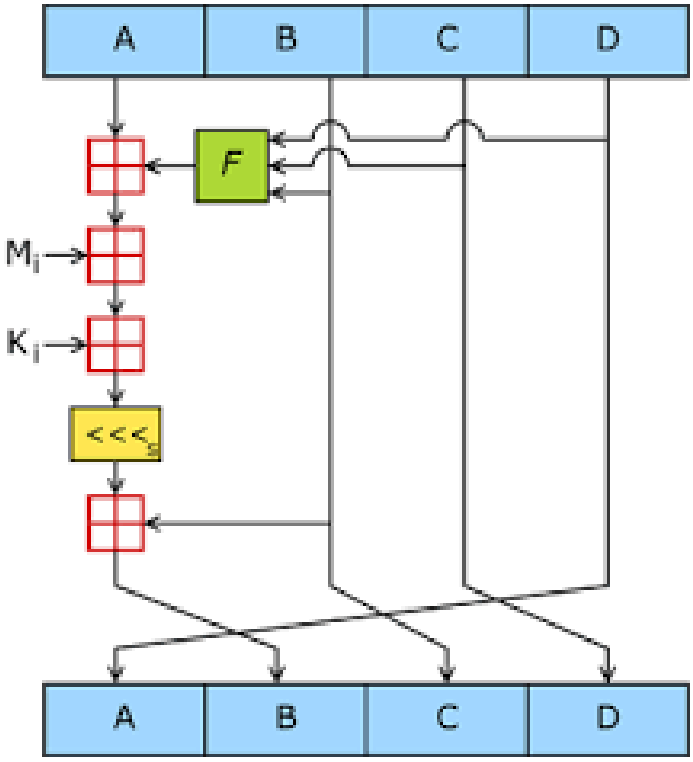

Figure 4: *Một vòng của MD5*

3.2 Hiện thực chương trình

Thư viện dùng trong chương trình là: System.Security.Cryptography;

Chương trình hiện thực gồm 2 giải thuật: AES và DES.

AES:

(Code được hiện thực trong file FileFactory.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;

namespace Ass1EDCS
{
    public class FileFactory
    {
        public static byte[] EncryptAES(byte[] input, string key)
        {
            RijndaelManaged AES = new RijndaelManaged();
            MD5CryptoServiceProvider Hash_AES = new MD5CryptoServiceProvider();
```

```
16              byte[] encrypted;
17              try
18              {
19                  byte[] hash = new byte[32];
20                  byte[] temp = Hash_AES.ComputeHash(System.Text.ASCIIEncoding.
                        ASCII.GetBytes(key));
21                  Array.Copy(temp, 0, hash, 0, 16);
22                  Array.Copy(temp, 0, hash, 15, 16);
23                  AES.Key = hash;
24                  AES.Mode = CipherMode.ECB;
25                  ICryptoTransform DESEncrypter = AES.CreateEncryptor();
26                  encrypted = DESEncrypter.TransformFinalBlock(input, 0, input.
                        Length);
27                  return encrypted;
28              }
29              catch (Exception ex)
30              {
31                  throw ex;
32              }
33
34          }
35
36          public static byte[] DecryptAES(byte[] input, string key)
37          {
38              System.Security.Cryptography.RijndaelManaged AES = new System.
                    Security.Cryptography.RijndaelManaged();
39              System.Security.Cryptography.MD5CryptoServiceProvider Hash_AES =
                    new System.Security.Cryptography.MD5CryptoServiceProvider();
40              byte[] decrypted;
41              try
42              {
43                  byte[] hash = new byte[32];
44                  byte[] temp = Hash_AES.ComputeHash(System.Text.ASCIIEncoding.
                        ASCII.GetBytes(key));
45                  Array.Copy(temp, 0, hash, 0, 16);
46                  Array.Copy(temp, 0, hash, 15, 16);
47                  AES.Key = hash;
48                  AES.Mode = System.Security.Cryptography.CipherMode.ECB;
49                  System.Security.Cryptography.ICryptoTransform DESDecrypter =
                        AES.CreateDecryptor();
50                  decrypted = DESDecrypter.TransformFinalBlock(input, 0, input.
                        Length);
51                  return decrypted;
52              }
53              catch (Exception ex)
54              {
55                  throw ex;
```

```
            }
        }
    }
}
```

DES:

(Code được hiện thực trong file FileFactoryDES.cs)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Security.Cryptography;

namespace Ass1EDCS
{
    public class FileFactoryDES
    {

        public static byte[] EncryptDES(byte[] source, string key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
                TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new
                MD5CryptoServiceProvider();

            byte[] byteHash;

            byteHash = hashMD5Provider.ComputeHash(Encoding.UTF8.GetBytes(key))
                ;
            desCryptoProvider.Key = byteHash;
            desCryptoProvider.Mode = CipherMode.ECB; //CBC, CFB

            byte[] encrypt = desCryptoProvider.CreateEncryptor().
                TransformFinalBlock(source, 0, source.Length);
            return encrypt;
        }
        public static byte[] DecryptDES(byte[] input, string key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
                TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new
                MD5CryptoServiceProvider();

            byte[] byteHash;
```

```
33
34              byteHash = hashMD5Provider.ComputeHash(Encoding.UTF8.GetBytes(key))
                    ;
35              desCryptoProvider.Key = byteHash;
36              desCryptoProvider.Mode = CipherMode.ECB; //CBC, CFB
37
38              byte[] decrypt = desCryptoProvider.CreateDecryptor().
                    TransformFinalBlock(input, 0, input.Length);
39              return decrypt;
40          }
41      }
42  }
```

Check hash MD5

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7  using System.Security.Cryptography;
8
9  namespace Ass1EDCS
10 {
11     public class MD5CheckSum
12     {
13         public static string MD5Check(byte[] input)
14         {
15             MD5 test = MD5.Create();
16             byte[] arCode = test.ComputeHash(input);
17             string strMD5 = "";
18             foreach (byte by in arCode)
19             {
20                 strMD5 += by.ToString("x2");
21             }
22             return strMD5;
23         }
24     }
25 }
```

4 Phân tích và kết luận

4.1 Phân tích các giải thuật

4.1.1 Giải thuật mã hóa AES

Thời gian mã hóa nhanh, giải thuật đơn giản, có 16 chu trình giống nhau trong quá trình xử lý

Nếu các kỹ thuật tấn công được cải thiện thì AES có thể bị phá vỡ vì ranh giới giữa số chu trình của thuật toán và số chu trình bị phá vỡ quá nhỏ. Tháng 10 năm 2005, Adi Shamir và 2 nhà nghiên cứu khác có một bài tấn công minh họa một vài dạng khác. Với tấn công thực hiện 2^{120} là thành công dù tấn công này chưa thể thực hiện trong thực tế.

4.1.2 Giải thuật mã hóa DES

DES có tính chất đặc thù: $E_K(P) = C \Leftrightarrow E_{\overline{K}}(\overline{P}) = \overline{C}$

E_K là bản mã hóa của E với khóa K. $\overline{x}$ là phần bù của x theo từng bít (1 thay bằng 0 và ngược lại). P và C là văn bản rõ (trước khi mã hóa) và văn bản mã (sau khi mã hóa). Do tính bù, ta có thể giảm độ phức tạp của tấn công duyệt toàn bộ xuống 2 lần (tương ứng với 1 bít) với điều kiện là ta có thể lựa chọn bản rõ.

Ngoài ra DES còn có 4 khóa yếu (weak keys). Khi sử dụng khóa yếu thì mã hóa (E) và giải mã (D) sẽ cho ra cùng kết quả: $E_K(E_K(P)) = P$ hoặc $E_K = D_K$

Bên cạnh đó, còn có 6 cặp khóa nửa yếu (semi-weak keys). Mã hóa với một khóa trong cặp, K_1, tương đương với giải mã với khóa còn lại K_2: $E_{K1}(E_{K2}(P)) = P$ hoặc $E_{K2} = D_{K1}$

Tuy nhiên có thể dễ dàng tránh được những khóa này khi thực hiện thuật toán, có thể bằng cách thử hoặc chọn khóa một cách ngẫu nhiên. Khi đó khả năng chọn phải khóa yếu là rất nhỏ.

Thời gian mã hóa nhanh, giải thuật đơn giản, có 16 chu trình giống nhau trong quá trình xử lý

DES đã được chứng minh là không tạo thành nhóm. Nói một cách khác, tập hợp EK (cho tất cả các khóa có thể) theo phép hợp thành không tạo thành một nhóm hay gần với một nhóm (Campbell and Wiener, 1992). Vấn đề này vẫn còn để mở và nếu như không có tính chất này thì DES có thể bị phá vỡ dễ dàng hơn và việc áp dụng DES nhiều lần (ví dụ như trong Triple DES) sẽ không làm tăng thêm độ an toàn của DES.

Kết luận

AES đang là tiêu chuẩn của giải thuật mã hóa và giải mã. Nhưng với những điểm yếu phân tích ở trên trong

tương lai không xa nó có thể bị phá vỡ. Chúng ta cần nghiên cứu để tối ưu hóa AES khắc phục những hạn chế của nó.

Hiện nay DES được xem là không đảm bảo an toàn cho nhiều ứng dụng, do giải thuật này đó độ dài khóa quá nhỏ (56bits). Thuật toán cải tiến từ DES được tin tưởng an toàn là Triple DES (thực hiện DES 3 lần), nhưng trênlý thuyết, nếu tập hợp $\{E_K\}\{E_K\}$ (cho tất cả các khóa có thể) theo phép hợp thành không tạo thành một nhóm hay gần với một nhóm (Campbell and Wiener, 1992) sẽ bị phá vỡ dễ dàng hơn, vì việc áp dụng DES nhiều lần sẽ không làm tăng thêm độ an toàn của DES.

4.2 AES và DES trong chương trình

- Trong quá trình demo chương trình, khi mã hóa/giải mã các file lớn thì AES có thời gian ngắn hơn so với DES. (Với các file nhỏ thì độ chênh lệch về thời gian không nhiều)
- Chương trình mã hóa có tính chính xác cao với các file vùa và nhỏ. Tuy nhiên, do chưa có điều kiện để kiểm tra trên các file lớn nên không thể kết luận chính xác nhất về vấn để này.

Ưu điểm của chương trình

- Thực hiện được 2 giải thuật mã hóa: AES và DES
- Chương trình có khả năng mã hóa hầu hết các file như: text, png, word,...
- Sử dụng hàm hash MD5 để kiểm tra tính toàn vẹn
- Chương trình có độ chính xác cao khi mã hóa/giải mã.

Nhược điểm của chương trình

- Chương trình chỉ dừng lại ở mức cơ bản
- Chưa hiện thực các chức năng nâng cao như bài tập lớn đưa ra

5 Hướng phát triển

- Có thể bổ sung các chức năng nâng cao như bài tập lớn đã đưa ra
- Bổ sung thêm các giải thuật mã hóa khác mà hiện nay đang được sử dụng
- Thêm thanh trạng thái mô tả quá trình
- Tìm hiểu thêm về cơ chế bảo mật hiện nay đang được sử dụng
- Tìm hiểu thêm về cơ chế hàm một chiều để tăng tính bảo mật cho khóa

6 Tham khảo

1. $https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.des?view=netcore-3.1$
2. $https://en.wikipedia.org/wiki/Data_Encryption_Standard$
3. $https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes?view=netcore-3.1$
4. $https://vi.wikipedia.org/wiki/Advanced_Encryption_Standard$

7 Phụ lục

Hướng dẫn chạy chương trình

- Giao diện chương trình

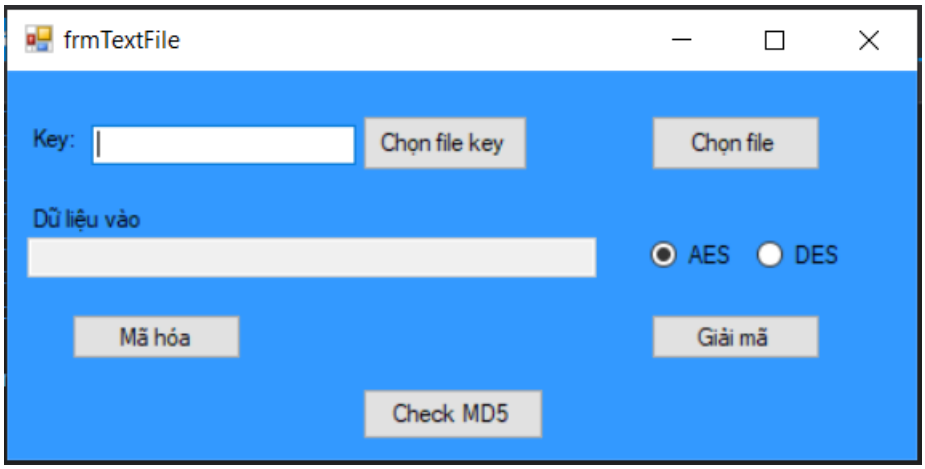

Figure 5: *Giao diện chính*

- Có thể chọn kiểu mã hóa bằng cách click chọn một trong hai RadioButton AES hoặc DES.

- Giả sử chọn thuật toán AES

- Người dùng có thể nhập trực tiếp khóa vào, hoặc nhấp vào button "Chọn file key" để chọn file text chứa key
- Một bảng chọn hiện ra, ta chọn file text chứa file key. Chọn file xong, nhấp OK

- Người dùng nhấp button "Chọn file", sau đó một một bảng hiện lên, người dùng chọn file cần mã hóa/giải mã. Khi đó đường dẫn của file được hiện thị trong textBox "Dữ liệu vào".

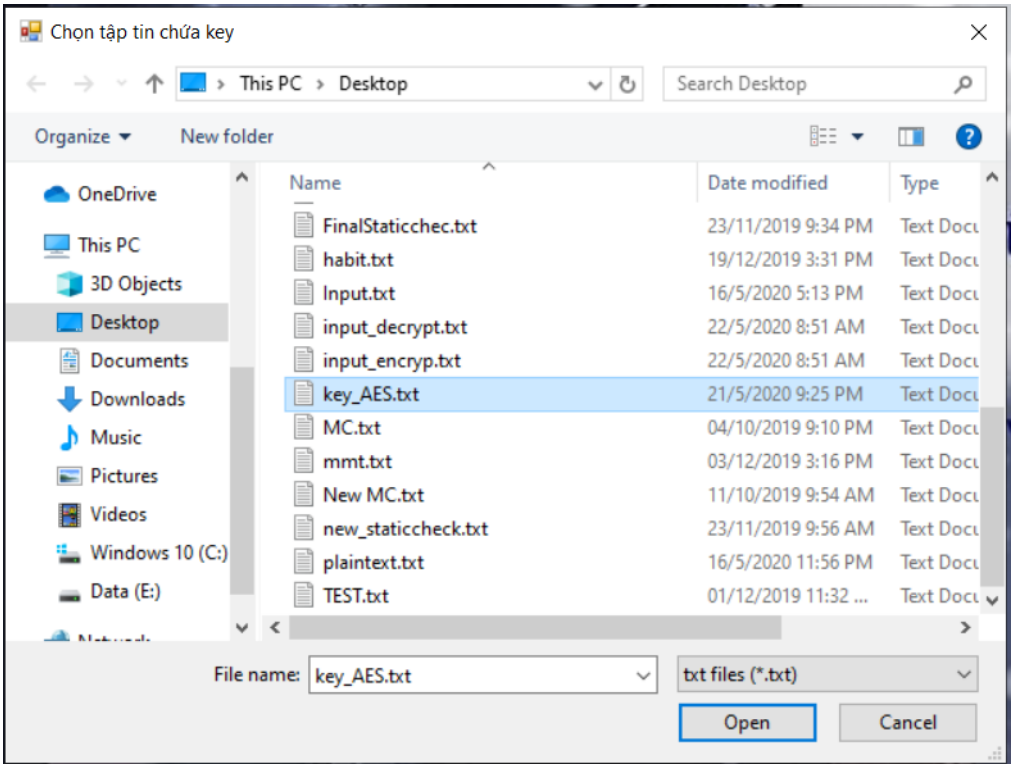

Figure 6: *Chọn file text chứa key*

- Người dụng nhấp chọn button "Mã hóa" hoặc button "Giải mã" để mã hóa/giải mã.

- Khi đó một bảng sẽ hiện lên, người dụng chọn tên file cần lưu,chọn đưởng dẫn lưu file. Khi đó chương trình sẽ tự động lưu file giải mã/mã hóa theo đuôi file của file đầu vào.

- Khi giải mã file, nếu khóa không đúng thì chương trình sẽ hiện thị thông báo, yêu cầu nhập lại khóa (hoặc chọn lại file chứa khóa).

- Check hash MD5: người dùng nhập chọn button "Check MD5 trên cửa số giao diện". Khi đó cửa số mới hiện ra. Người dùng chọn file nguồn và file đích để tiến hành kiểm tra. Sau đó, chọn Check MD5. Khi đó 2 textBox chứa code hash MD5 của file nguồn và file đích sẽ hiện ra, đồng thời có bảng thông báo "Giải mã và mã hóa thành công, file input và file output là giống nhau". Còn không sẽ hiện ra thông báo "Giải mã thất bại".

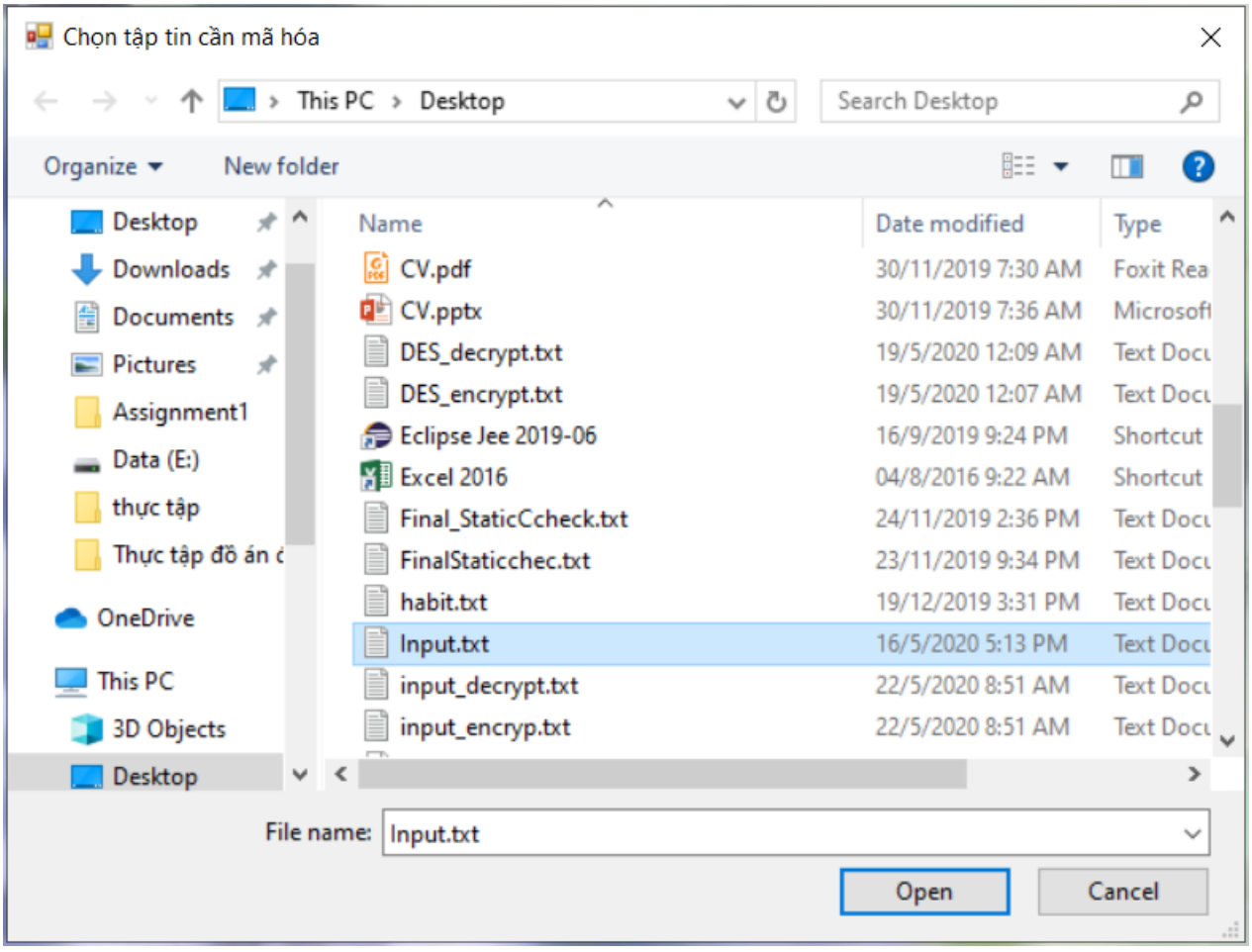

Figure 7: *Chọn file muốn mã hóa/giải mã*

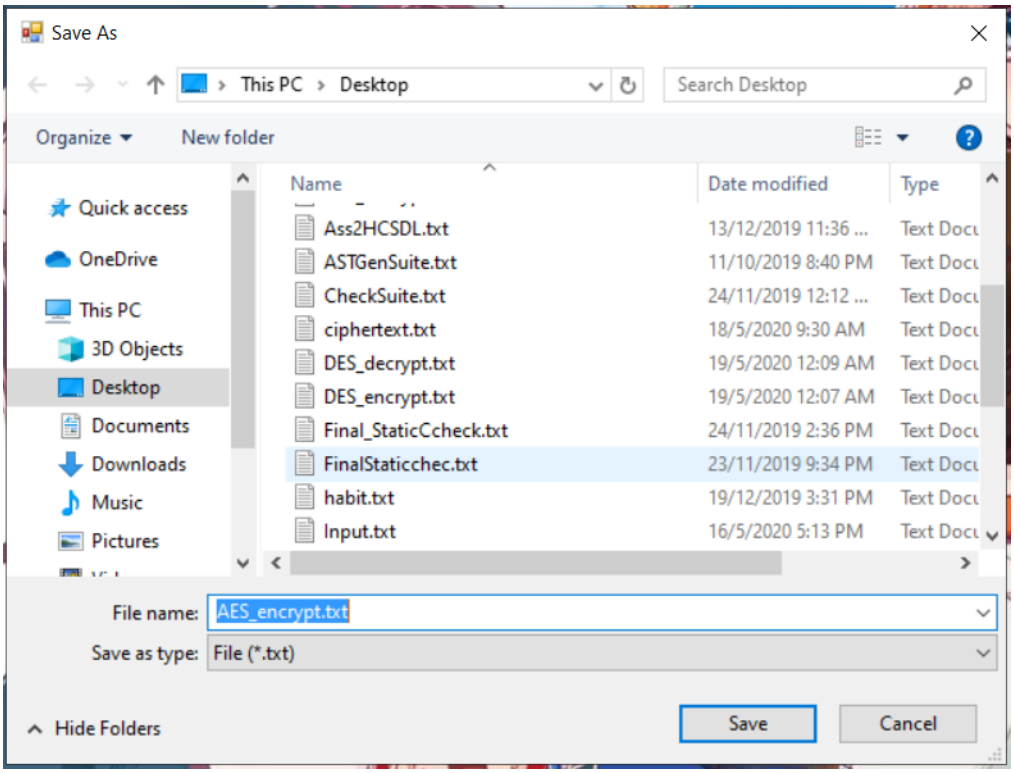

Figure 8: *Lưu file đã má hóa/giải mã*

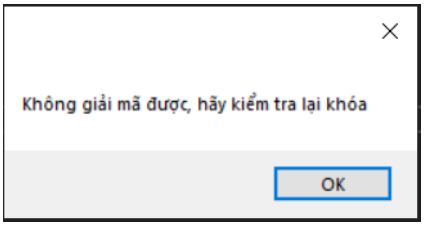

Figure 9: *Lỗi sai khóa khi giải mã*

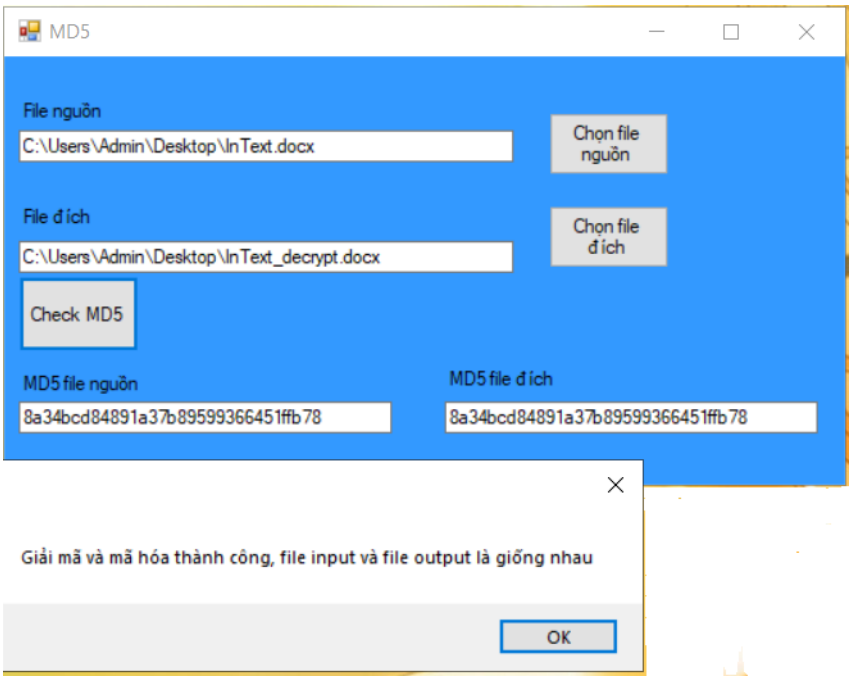

Figure 10: *Check MD5*