Encypted Decrypted Files

MẬT MÃ VÀ AN NINH MẠNG

STUDENT: LÊ BÁ QUÂN – 1712817
TEACHER: NGUYỄN HỮU HIẾU

MỤC LỤC:

I. GIỚI THIỆU

Mã hoá là phương pháp giúp bảo vệ dữ liệu cá nhân nhạy cảm trên máy tính của bạn, cho dù bạn có gởi dữ liệu cho cá nhân, tổ chức nào đó qua mạng Internet, hay sao lưu dữ liệu cá nhân trên các máy chủ, Cloud,..., thì việc mã hoá sẽ ngăn chặn bất cứ ai có thể đọc được dữ liệu trước khi được sự cho phép của bạn.

Trong assignment này ta sẽ thực hiện một chương trình mã hóa để giữ cho các tập tin và thư mục trên máy tính của mình thật sự an toàn. Cụ thể là xây dựng chương trình mã hóa và giải mã các tập tin và thư mục sử dụng các giải thuật mã hóa được sử dụng phổ biến trong thực tế như DES, AES, RSA...

- *Những tính năng cơ bản cần có trong ứng dụng:*
- *Chương trình có khả năng mã hóa một tập tin bất kì như hình ảnh, âm thanh, văn bản, pdf...Sinh viên trình bày rõ trong báo cáo một số loại tập tin mà chương trình hỗ trợ mã hóa/giải mã.*
- *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 mực 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. Sử dụng các hàm Hash như MD5, SHA để chứng minh. tính toàn vẹn giữa tập tin gốc ban đầu được chọn và tập tin output của quá trình giải mã.*

II. TIẾN HÀNH THỰC HIỆN:

A. Cơ sở lý thuyết:

1. *Giải thuật mã hoá DES:*

a) 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.

b) Mô tả thuật toán:

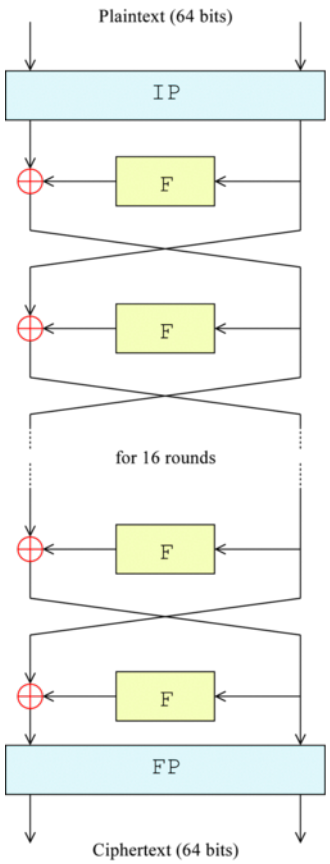


***Figure 1**: Cấu trúc giải thuật Feistel dùng trong DES*

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 bit.

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
2. permutation) với việc nhân đôi một số bit. Giai đoạn này được ký hiệu là E trong sơ đồ.
3. 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.
4. 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 khoá con:

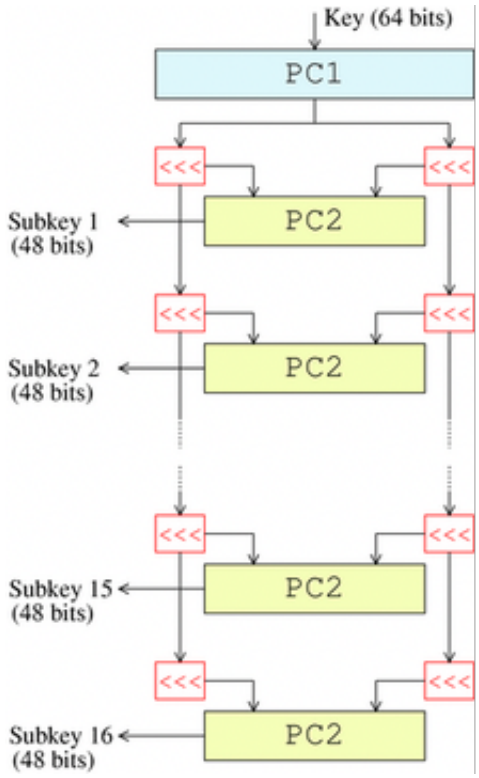


Figure 2: *Quá trình tạo khóa con dùng DES*

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.

2. Giải thuật mã hoá AES:

a) 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.

b) 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

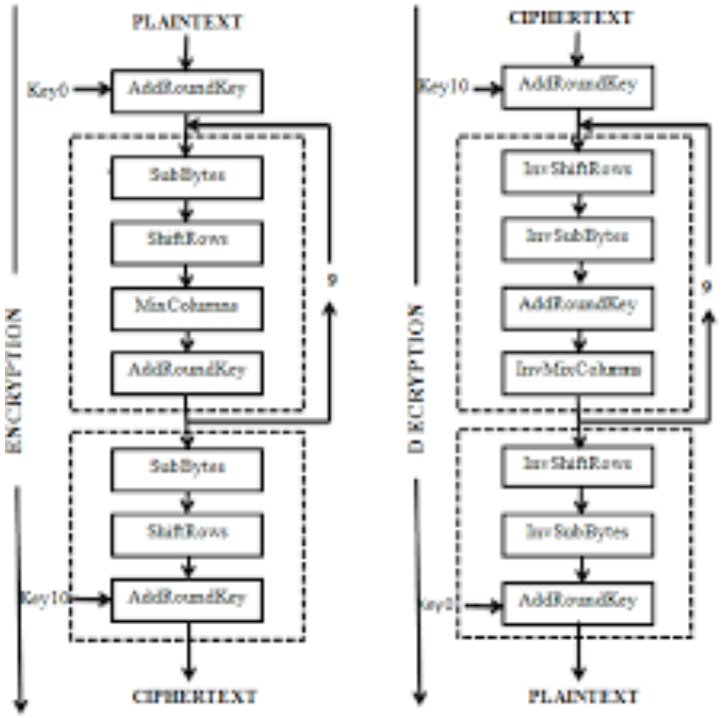


***Figure 3**: Quá trình mã hóa/giải mã dùng AES*

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. Hàm Hash SHA256:

a) Giới thiệu SHA-256:

SHA-256 là một trong những hàm băm kế tiếp cho SHA-1 và là một trong những hàm băm mạnh nhất hiện có. SHA-256 không phức tạp hơn nhiều so với mã SHA-1 và chưa bị xâm phạm theo bất kỳ cách nào. Khóa 256 bit làm cho nó trở thành đôi bạn tốt với AES.

b) Mô tả thuật toán:

SHA-256 là một thuật toán được mô tả cụ thể của thuật toán SHA -2 như 512 và gần đây là các phiên bản 224 bit, ngoài ra nó còn là sự phát triển thành công từ người tiền nhiệm SHA -1, bản thân SHA -256 là một sự cải tiến của SHA-0. Thuật toán SHA -2 được NSA phát triển để trả lời vấn đề bảo mật của SHA -1. Thuật toán này nhận đầu vào là một thông điệp có độ dài bit tối đa là hai lũy thừa sáu mươi tư và cho ra kết quả checksum có độ dài 256 bit. Có vẻ như SHA -256 đang ngày càng được sử dụng nhiều hơn để thay thế hàm băm MD5 cũ, thậm chí ngay cả SHA-1 cũng có mức độ bảo mật tốt hơn MD5. SHA-256 thực sự là sự thay thế tốt nhất vì sự cân bằng giữa kích thước dữ liệu và mức độ an toàn. Như những chức năng mã hóa khác của họ hàng nhà SHA.

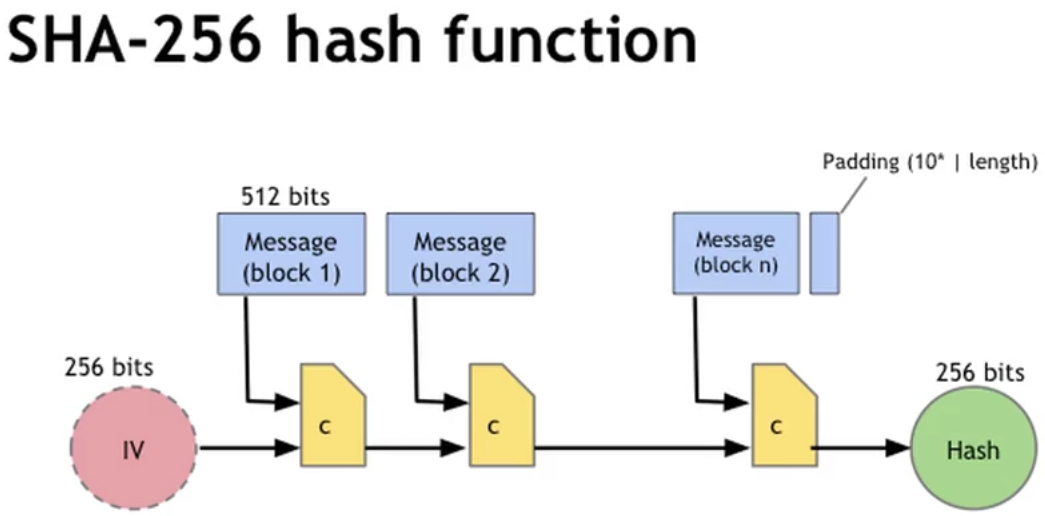


***Figure 4**: Quá trình mã hóa dùng hàm Hash SHA-256*

B. Ý tưởng thực hiện:

1. *Hiện thực 1 ứng dụng mã hoá, giải mã file trên môi trường .NET bằng ngôn ngữ C#.*
2. *Sử dụng màn hình Terminal để tiến hành thao tác với phần mềm.*
3. *Sử dụng 2 giải thuật mã hoá nổi tiếng là AES và DES được hỗ trợ từ thư viện Security.Cryptography.*
4. *Lựa chọn hàm băm SHA256 để kiểm tra tính toàn vẹn của File trước khi mã hoá và sau khi giải mã.*
5. *Chương trình sẽ tự động thực hiện công việc kiểm tra tính toàn vẹn bằng cách trước khi mã hoá một file, chương trình sẽ lưu lại chuỗi băm của file gốc. Khi file được giải mã, chương trình sẽ kiểm tra xem có tồn tại file chứa chuỗi băm của file gốc hay không, nếu có sẽ thực hiện việc kiểm tra tính toàn vẹn.*
6. *Có thể mã hoá cũng như giải mã 1 file nhiều lần mà không làm mất đi tính toàn vẹn của file ban đầu.*

C. Quy trình thực hiện:

1. *Hiện thực hàm mã hoá dùng giải thuật AES:*

```
        public static byte[] EncryptAES(byte[] input, byte[] key)
        {
            RijndaelManaged AES = new RijndaelManaged();
            MD5CryptoServiceProvider Hash_AES = new MD5CryptoServiceProvider();
            try
            {
                byte[] hash = new byte[32];
                byte[] temp = Hash_AES.ComputeHash(key);
                Array.Copy(temp, 0, hash, 0, 16);
                Array.Copy(temp, 0, hash, 15, 16);
                AES.Key = hash;
                AES.Mode = CipherMode.ECB;
                ICryptoTransform DESEncrypter = AES.CreateEncryptor();
                return DESEncrypter.TransformFinalBlock(input, 0, input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

2. *Hiện thực hàm giải mã dùng giải thuật DES:*

```
        public static byte[] DecryptAES(byte[] input, byte[] key)
        {
            System.Security.Cryptography.RijndaelManaged AES = new
System.Security.Cryptography.RijndaelManaged();
            System.Security.Cryptography.MD5CryptoServiceProvider Hash_AES = new
System.Security.Cryptography.MD5CryptoServiceProvider();
            try
            {
                byte[] hash = new byte[32];
                byte[] temp = Hash_AES.ComputeHash(key);
                Array.Copy(temp, 0, hash, 0, 16);
                Array.Copy(temp, 0, hash, 15, 16);
                AES.Key = hash;
                AES.Mode = System.Security.Cryptography.CipherMode.ECB;
                System.Security.Cryptography.ICryptoTransform DESDecrypter = AES.CreateDecryptor();
                return DESDecrypter.TransformFinalBlock(input, 0, input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

3. Hiện thực hàm mã hoá dùng giải thuật DES:

```
        public static byte[] EncryptDES(byte[] input, byte[] key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new MD5CryptoServiceProvider();
            try
            {
                byte[] byteHash;
                byteHash = hashMD5Provider.ComputeHash(key);
                desCryptoProvider.Key = byteHash;
                desCryptoProvider.Mode = CipherMode.ECB;
                return desCryptoProvider.CreateEncryptor().TransformFinalBlock(input, 0,
input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

4. Hiện thực hàm giải mã dùng giải thuật DES:

```
        public static byte[] DecryptDES(byte[] input, byte[] key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new MD5CryptoServiceProvider();
            try
            {
                byte[] byteHash;
                byteHash = hashMD5Provider.ComputeHash(key);
                desCryptoProvider.Key = byteHash;
                desCryptoProvider.Mode = CipherMode.ECB;
                return desCryptoProvider.CreateDecryptor().TransformFinalBlock(input, 0,
input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
```

5. Hiện thực hàm tạo và lưu trữ file chuỗi HASH của file trước khi mã hoá sử dụng hàm băm SHA-256:

```
        public static void SaveHash(byte[] data, string fileName)
        {
            SHA256 mySHA256 = SHA256.Create();
            byte[] hashValue = mySHA256.ComputeHash(data);
            string HashFile = @"Hash" + fileName + ".ecrypted";
            if (File.Exists(HashFile))
            {
                File.Delete(HashFile);
            }
            File.WriteAllBytes(HashFile, hashValue);
            Console.WriteLine("Save Hash File In : " + HashFile);
        }
```

6. Hiện thực hàm kiểm tra tính toàn vẹn của file sau khi giải mã sử dụng hàm băm SHA-256:

```
        public static void CheckIntegrity(byte[] data, string fileName)
        {
            SHA256 mySHA256 = SHA256.Create();
            byte[] hashValue = mySHA256.ComputeHash(data);
            string HashFile = @"Hash" + fileName + ".ecrypted";
            if (File.Exists(HashFile))
            {
                byte[] temp = File.ReadAllBytes(HashFile);
                if (ByteArrayCompare(hashValue,temp))
                {
                    Console.WriteLine("File Integrity");
                }
                else
                {
                    Console.WriteLine("File Changed");
                }
                Console.WriteLine("Original  File's Hash Key: " +
Encoding.ASCII.GetString(hashValue,0,hashValue.Length));
                Console.WriteLine("Decrypted File's Hash Key: " +
Encoding.ASCII.GetString(temp,0,temp.Length));
                File.Delete(HashFile);
            }
            else
            {
                Console.WriteLine("Can't File Hash File On Data");
            }
        }
```

7. Hiện thực hàm con kiểm tra 2 chuỗi byte có trùng nhau không:

```
        static bool ByteArrayCompare(byte[] a1, byte[] a2)
        {
            if (a1.Length != a2.Length)
                return false;

            for (int i=0; i<a1.Length; i++)
                if (a1[i]!=a2[i])
                    return false;
            return true;
        }
```

8. Source Code toàn bộ chương trình:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Security.Cryptography;

namespace Ass_1_With_C_
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome To En-Decrypt App:");

            InputFile:
            Console.WriteLine("Please Input Your File URL:");
            string fileURL = Console.ReadLine();
            if (File.Exists(fileURL)){}
            else
            {
                Console.WriteLine("File Doesn't Exist. Try Again:");
                goto InputFile;
            }
```

```
            byte[] filecontent = File.ReadAllBytes(fileURL);

            InputKey:
            Console.WriteLine("Now Input Your Key:");
            string keyURL = Console.ReadLine();
            if (File.Exists(keyURL)){}
            else
            {
                Console.WriteLine("File Doesn't Exist. Try Again:");
                goto InputKey;
            }
            byte[] key = File.ReadAllBytes(keyURL);

            ReInputOption:
            Console.WriteLine("Now Enter Your Selection:");
            Console.WriteLine("1. Encrypt with AES");
            Console.WriteLine("2. Decrypt with AES");
            Console.WriteLine("3. Encrypt with DES");
            Console.WriteLine("4. Decrypt with DES");
            string userinput = Console.ReadLine();
            if(userinput == "1")
            {
                File.WriteAllBytes(fileURL, EncryptAES(filecontent, key));
                Console.WriteLine("File " + fileURL + " Encrypted Successful Using AES");
                SaveHash(filecontent,fileURL);
            }
            else if(userinput == "2")
            {
                byte[] newTemp = DecryptAES(filecontent, key);
                File.WriteAllBytes(fileURL, newTemp);
                Console.WriteLine("File " + fileURL + " Decrypted Successful Using AES");
                CheckIntegrity(newTemp,fileURL);
            }
            else if(userinput == "3")
            {
                File.WriteAllBytes(fileURL, EncryptDES(filecontent, key));
                Console.WriteLine("File " + fileURL + " Encrypted Successful Using DES");
                SaveHash(filecontent,fileURL);
            }
            else if(userinput == "4")
            {
```

```
                byte[] newTemp = DecryptDES(filecontent, key);
                File.WriteAllBytes(fileURL, newTemp);
                Console.WriteLine("File " + fileURL + " Decrypted Successful Using DES");
                CheckIntegrity(newTemp,fileURL);
            }
            else
            {
                Console.WriteLine("Enter Wrong Selection!");
                goto ReInputOption;
            }
        }
        public static byte[] EncryptAES(byte[] input, byte[] key)
        {
            RijndaelManaged AES = new RijndaelManaged();
            MD5CryptoServiceProvider Hash_AES = new MD5CryptoServiceProvider();
            try
            {
                byte[] hash = new byte[32];
                byte[] temp = Hash_AES.ComputeHash(key);
                Array.Copy(temp, 0, hash, 0, 16);
                Array.Copy(temp, 0, hash, 15, 16);
                AES.Key = hash;
                AES.Mode = CipherMode.ECB;
                ICryptoTransform DESEncrypter = AES.CreateEncryptor();
                return DESEncrypter.TransformFinalBlock(input, 0, input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
        public static byte[] DecryptAES(byte[] input, byte[] key)
        {
            System.Security.Cryptography.RijndaelManaged AES = new
System.Security.Cryptography.RijndaelManaged();
            System.Security.Cryptography.MD5CryptoServiceProvider Hash_AES = new
System.Security.Cryptography.MD5CryptoServiceProvider();
            try
            {
                byte[] hash = new byte[32];
                byte[] temp = Hash_AES.ComputeHash(key);
```

```
                Array.Copy(temp, 0, hash, 0, 16);
                Array.Copy(temp, 0, hash, 15, 16);
                AES.Key = hash;
                AES.Mode = System.Security.Cryptography.CipherMode.ECB;
                System.Security.Cryptography.ICryptoTransform DESDecrypter = AES.CreateDecryptor();
                return DESDecrypter.TransformFinalBlock(input, 0, input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
        public static byte[] EncryptDES(byte[] input, byte[] key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new MD5CryptoServiceProvider();
            try
            {
                byte[] byteHash;
                byteHash = hashMD5Provider.ComputeHash(key);
                desCryptoProvider.Key = byteHash;
                desCryptoProvider.Mode = CipherMode.ECB;
                return desCryptoProvider.CreateEncryptor().TransformFinalBlock(input, 0,
input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
        public static byte[] DecryptDES(byte[] input, byte[] key)
        {
            TripleDESCryptoServiceProvider desCryptoProvider = new
TripleDESCryptoServiceProvider();
            MD5CryptoServiceProvider hashMD5Provider = new MD5CryptoServiceProvider();
            try
            {
                byte[] byteHash;
                byteHash = hashMD5Provider.ComputeHash(key);
                desCryptoProvider.Key = byteHash;
```

```
                desCryptoProvider.Mode = CipherMode.ECB;
                return desCryptoProvider.CreateDecryptor().TransformFinalBlock(input, 0,
input.Length);
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public static void SaveHash(byte[] data, string fileName)
        {
            SHA256 mySHA256 = SHA256.Create();
            byte[] hashValue = mySHA256.ComputeHash(data);
            string HashFile = @"Hash" + fileName + ".ecrypted";
            if (File.Exists(HashFile))
            {
                File.Delete(HashFile);
            }
            File.WriteAllBytes(HashFile, hashValue);
            Console.WriteLine("Save Hash File In : " + HashFile);
        }
        public static void CheckIntegrity(byte[] data, string fileName)
        {
            SHA256 mySHA256 = SHA256.Create();
            byte[] hashValue = mySHA256.ComputeHash(data);
            string HashFile = @"Hash" + fileName + ".ecrypted";
            if (File.Exists(HashFile))
            {
                byte[] temp = File.ReadAllBytes(HashFile);
                if (ByteArrayCompare(hashValue,temp))
                {
                    Console.WriteLine("File Integrity");
                }
                else
                {
                    Console.WriteLine("File Changed");
                }
                Console.WriteLine("Original  File's Hash Key: " +
Encoding.ASCII.GetString(hashValue,0,hashValue.Length));
```

```
                Console.WriteLine("Decrypted File's Hash Key: " +
Encoding.ASCII.GetString(temp,0,temp.Length));
                File.Delete(HashFile);
            }
            else
            {
                Console.WriteLine("Can't File Hash File On Data");
            }

        }
        static bool ByteArrayCompare(byte[] a1, byte[] a2)
        {
            if (a1.Length != a2.Length)
                return false;

            for (int i=0; i<a1.Length; i++)
                if (a1[i]!=a2[i])
                    return false;
            return true;
        }
    }
}
```

III. PHÂN TÍCH VÀ KẾT LUẬN:

A. Phân tích các giải thuật:

1. Giải thuật mã hoá 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 2120 là thành công dù tấn công này chưa thể thực hiện trong thực tế.

2. Giải thuật mã hoá 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, K1, 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 {EK }{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) sẽ bị phá vỡ dễ dàng hơn, vì việc áp dụng DES nhiều lần sẽ không làm tăng thêm độ an toàn của DES.

B. 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.

1. Ưu điểm của chương trình:

- Thực hiện được 2 giải thuật mã hóa: AES và DES
- Chương trình có khả năng mã hóa tất cả các file như: text, png, word,...
- Sử dụng hàm hash SHA-256 để kiểm tra tính toàn vẹn
- Chương trình có độ chính xác cao khi mã hóa/giải mã

2. Nhược điểm của chương trình:

- Chưa có giao diện sử dụng thân thiện mà phải thao tác qua Terminal
- Chương trình chỉ có chức năng cơ bản là mã hoá và giải mã
- Chưa thực hiện được các tính năng nâng cao bổ sung như mã hoá toàn bộ tập tin, …
- Muốn mã hoá file nào phải đưa file đó trực tiếp vào Folder chứa Source Code

IV. HƯỚNG PHÁT TRIỂN:

- Phát triển phần UX,UI cho ứng dụng
- Bổ sung thêm các giải thuật mã hoá khác cho ứng dụng
- Thêm thanh trạng thái mô tả quá trình

V. THAM KHẢO:

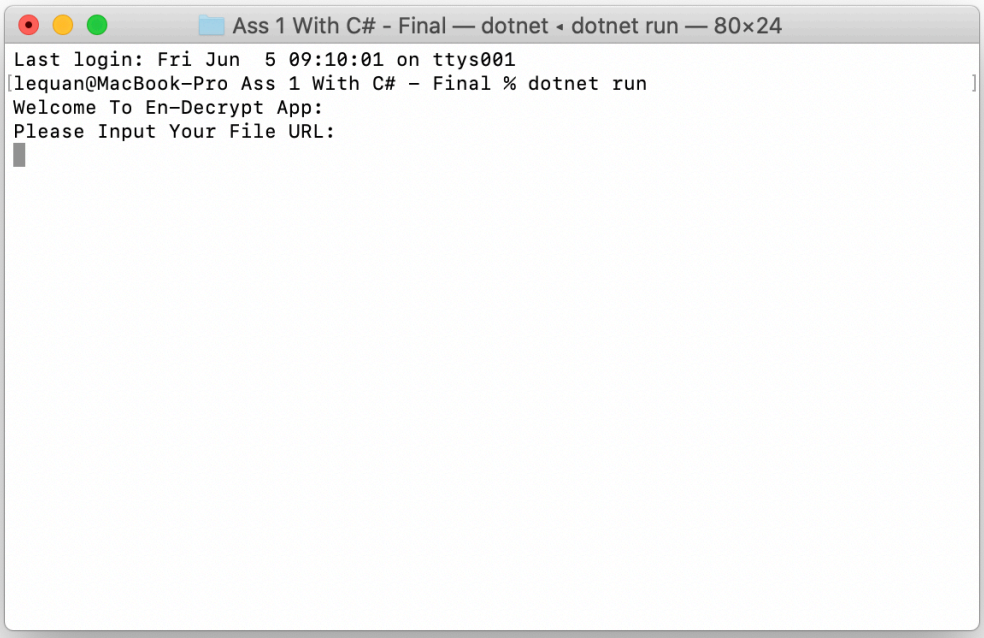
1. https : //docs.microsoft.com/en−us/dotnet/api/system.security.cryptography.des?view = netcore−3.1
2. https://en.wikipedia.org/wiki/DataEncryptionStandard
3. https : //docs.microsoft.com/en−us/dotnet/api/system.security.cryptography.aes?view = netcore−3.1
4. https://vi.wikipedia.org/wiki/AdvancedEncryptionStandard
5. https://medium.com/@pkmar437/blockchain-for-layman-part-2-a8984fda0acc
6. https://www.stdio.vn/articles/sha-256-va-sha-512-627

VI. PHỤ LỤC:

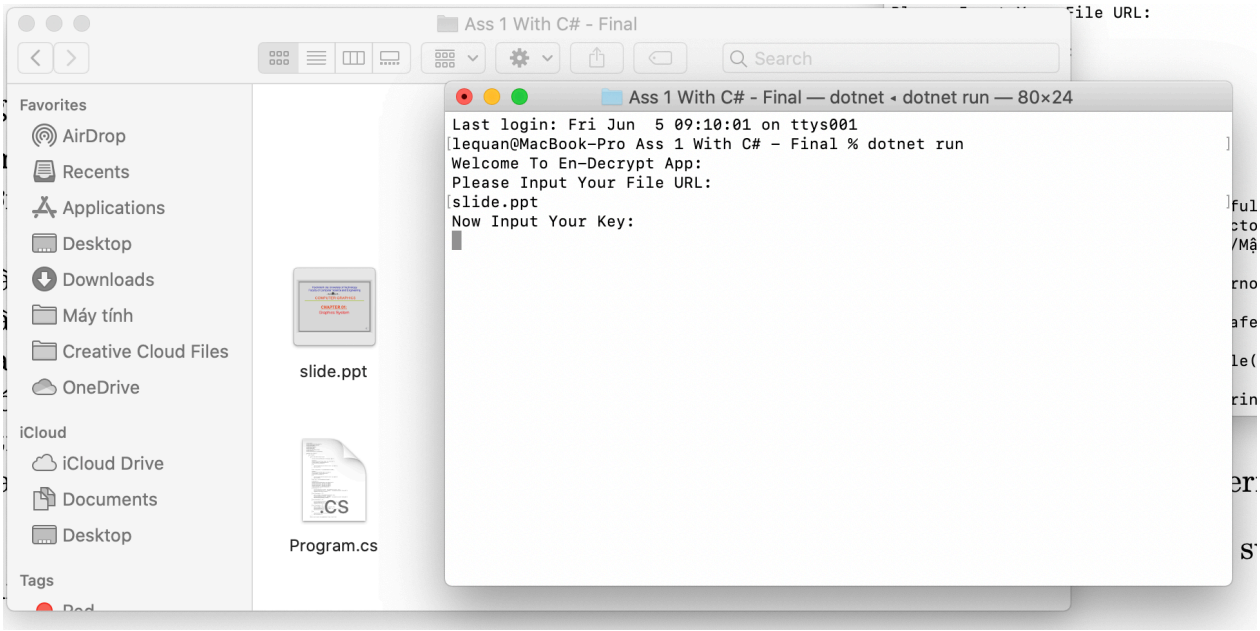
A. Thao tác với chương trình:

Mở cửa sổ Terminal vào thư mục chứa Source Code

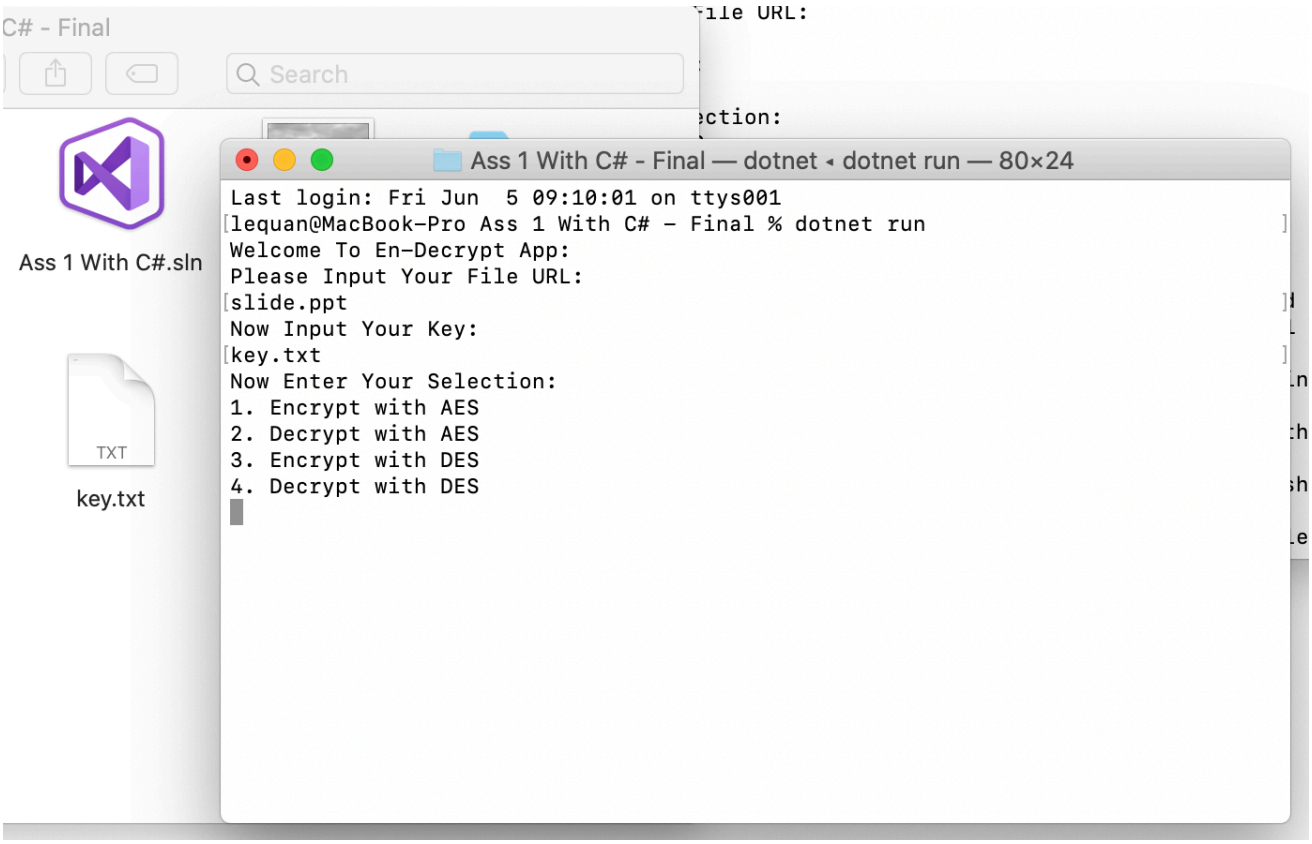
Nhập lệnh **dotnet run**



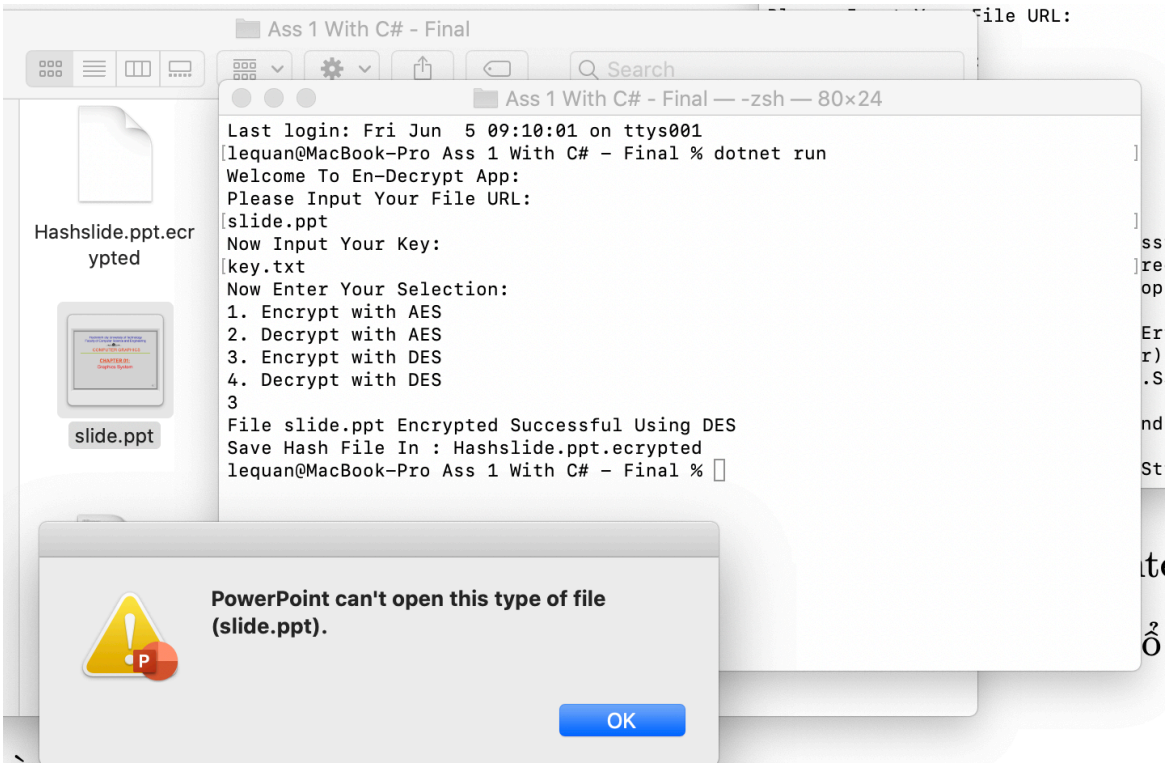
Nhập tên file, ví dụ **slide.ppt**



Nhập tên key file, ví dụ **key.txt**

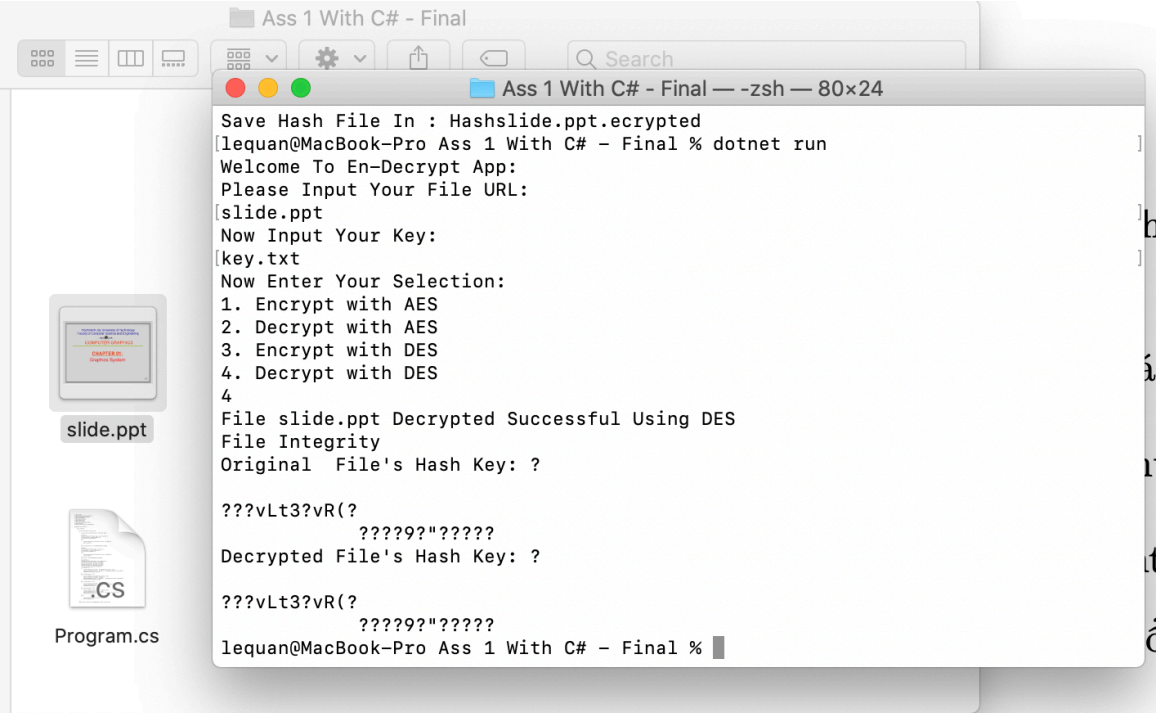


Nhập số tương đương với giải thuật muốn thực hiện, ví dụ nhập **3** là Encrypt with DES



Chương trình đã mã hoá thành công file slide.ppt, đồng thời sinh ra file hash Hashslide.ppt.encrypted để lưu giữ hàm băm của file ban đầu.

Thực hiện các bước như trên nhưng thay vì chọn mã hoá, ta chọn **4** – Decrypted with DES, ta nhận được kết quả:



Chương trình đã giải mã thành công file, đồng thời kiểm tra tính toàn vẹn của file dựa vào chuỗi băm đã được tạo ở bước mã hoá. Chương trình cũng tiến hành in ra 2 chuỗi băm cũng như xoá file hash được sinh ra ở trên.

B. Video demo:

https://drive.google.com/file/d/1-EzeXtE_bWQKK94zu0sOhdTfOam-mUyo/view?usp=sharing