ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

MẬT MÃ VÀ ANH NINH MẠNG

Đề tài

Ứng dụng mã hóa dữ liệu

GVHD: Nguyễn Hữu Hiếu
SV: Nguyễn Trần Lê Minh - 1511003
Lê Duy Thanh - 1512990
Nguyễn Xuân Nam - 1512098

TP. HỒ CHÍ MINH, THÁNG 03/2019

Mục lục

Mã hóa là phương pháp bảo vệ dữ liệu cá nhân nhạy cảm trên máy tính của bạn. Việc mã hóa còn ngăn chặn bất cứ ai đọc dữ liệu của bạn khi bạn gửi thông tin qua mạng hay đồng bộ lên máy chủ, cloud,...

Trong bài tập lớn này, nhóm sẽ thực hiện một số giải thuật mã hóa dễ cho các tập tin trong máy được an toàn.

1 Giới thiệu đề tài

Nhóm đã hiện thực ba giải thuật mã hóa phổ biến đó là RSA, DES và Steganography.

Với RSA, nhóm đọc dữ liệu cần mã hóa (plaint text) từ tệp văn bản (*.txt file - text file).
Với Steganography, nhóm đọc dữ liệu từ text file trộn vào một ảnh mang.
Với DES, nhóm có thể mã hóa các tệp cơ bản như text, mp4, pdf, png,... bằng key có sẵn từ tệp.
Để chứng minh plaint text trùng với dữ liệu được giải mã, nhóm sử dụng hàm băm để đối chiếu hai tệp.

2 Cơ sở lý thuyết

2.1 Giải thuật DES

DES, viết tắt của **Data Encryption Standard**, là một block cipher, nghĩa là nó xử lí từng khối plaintext đầu vào với một kích thước nhất định (như 64 bits,...) và trả về các khối đã được mã hóa tương ứng với cùng kích thước. Do đó, kết quả DES là một hoán vị của 64 bits đầu vào. DES là một thuật mã hóa đối xứng, sử dụng cùng một key cho cả quá trình mã hóa và giải mã. Cụ thể với việc mã hóa DES ECB 64 bits, có thể chia làm hai bước chính sau:

2.1.1 Tạo 16 subkeys với độ dài 48 bits

Key 64 bits sẽ được hoán vị thông qua bảng PC-1. Ta có thể hiểu bit thứ 57 của key trở thành bit thứ nhất, bit thứ 49 của key trở thành bit thứ hai,... sau khi được hoán vị bởi bảng PC-1. Từ key 64 bits ban đầu, ta có key hoán vị 56 bits K+. Chia K+ thành hai phần 28 bits, C_0 là

PC-1

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

Hình 1: Bảng PC1.

nửa trái và D_0 là nửa phải. Ta tạo ra 16 cặp khối C_n và D_n, $1 \leq n \leq 16$. Mỗi cặp C_n và D_n được tính từ cặp C_{n-1} và D_{n-1} với việc "dịch trái" từng bit trong mỗi khối, các bit di chuyển

qua trái, trừ bit đầu được đưa về cuối khối. Nhìn hình trên, ta có thể hiểu C_3 và D_3 được nhận

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Hình 2: Bảng dịch bit ở từng vòng lặp.

từ C_2 và D_2 thông qua dịch trái 2 bits, hay C_{16} và D_{16} được nhận từ C_{16} và D_{16} thông qua dịch trái 1 bit. Bây giờ chúng ta có 16 cặp C_nD_n, thông qua bảng PC-2 ta sẽ tạo ra 16 subkeys với độ dài mỗi key là 48 bits. Tương tự như PC-1, ta có thể hiểu như sau: bit đầu tiên của K_n là bit thứ 14 của C_nD_n, bit thứ hai của K_n là bit thứ 17 của C_nD_n,...

2.1.2 Mã hóa từng khối 64 bits dữ liệu

Ta dùng bảng IP để hoán vị 64 bits dữ liệu đầu vào M. Như hình dưới, bit thứ 58 của M trở thành bit thứ nhất của IP, bit thứ 50 của M trở thành bit thứ hai của IP,... Kế tiếp, ta chia IP thành hai nửa: nửa trái L_0 32 bits, và nửa phải R_0 cũng với độ dài 32 bits. Chúng ta sẽ trải qua 16 vòng lặp để xác định giá trị L_{16} và R_{16} theo công thức truy hồi:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus F(R_{n-1}, K_n)$$

Để tính hàm F, trước tiên ta sẽ mở rộng R_{n-1} từ 32 bits thành 48 bits. Bằng việc sử dụng lặp lại vài bits trong R_{n-1}, ta có bảng thể hiện hàm E biến 32 bits đầu vào thành 48 bits đầu ra như sau: Tiếp theo, ta thực hiện phép XOR từng bit $E(R_{n-1})$ với key K_n. Bởi vì ban đầu ta đã mở rộng 32 bits thành 48 bits nên bây giờ cần thu gọn lại. Kết quả phép toán XOR trên là 48 bits, hiểu cách khác chính là 8 nhóm 6 bits.

$$K_n \oplus E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8$$

PC-2

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

Hình 3: Bảng PC2.

IP

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

Hình 4: Bảng IP

E BIT-SELECTION TABLE

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

Hình 5: Bảng E.

Ta dùng 8 bảng S lần lượt cho từng nhóm đó, biến 6 bits đầu vào thành 4 bits đầu ra. Như vậy, hiển nhiên ta đã chuyển được kết quả 48 bits từ phép tính XOR về 32 bits. Cụ thể, chẳng hạn giá trị $S_1(B)$ được tính như sau: Bit đầu và bit cuối của B được ghép lại thành một số thập thân **i** trong đoạn [0;3]. Bốn bits chính giữa còn lại của B là một số thập phân **j** trong đoạn [0;15]. Ta tìm giá trị hàng thứ **i**, cột thứ **j** trong bảng S_1 thì đó chính là giá trị của $S_1(B)$.

S1

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

S2

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

S3

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

S4

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

S5

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

S6

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

S7

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

S8

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

Hình 6: 8 bảng S

Bước cuối cùng của hàm F là hoán vị 32 bits bằng bảng P. $F = P(S_1(B_1)S_2(B_2)...S_8(B_8))$

P

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

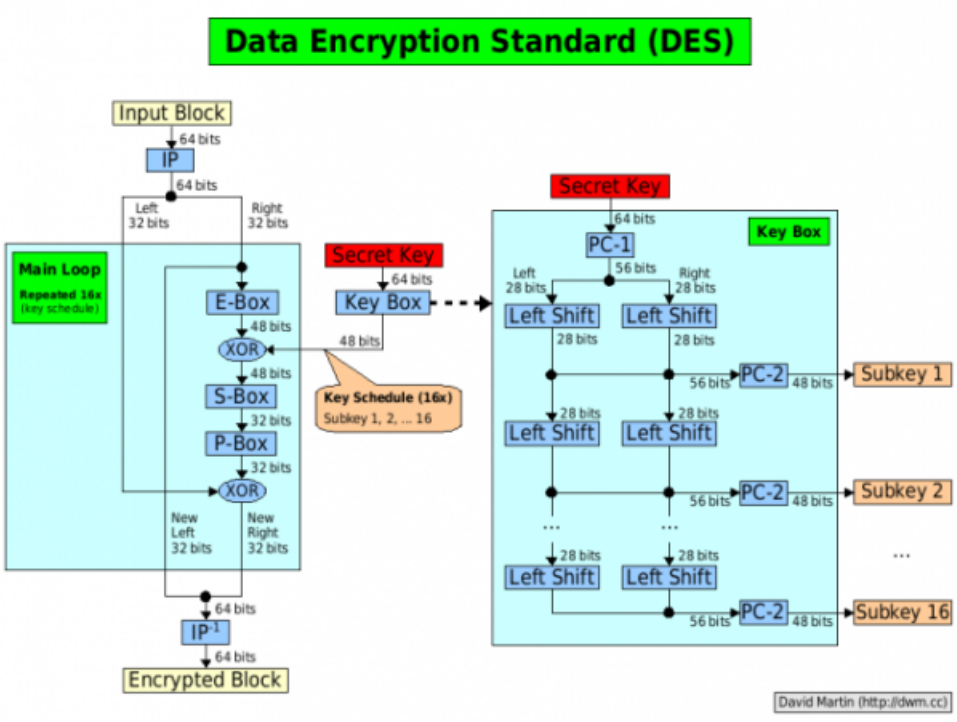
Hình 7: Bảng P.

Có giá trị hàm F, ta thực hiện phép XOR với L_{n-1} ta được giá trị R_n. Và sau 16 vòng lặp ta được giá trị L_{16} và R_{16}. Ta đảo ngược thứ tự của chúng trong khối 64 bits, trở thành $R_{16}L_{16}$ và hoán vị lần cuối với bảng IP^{-1}:

IP^{-1}

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

Hình 8: Bảng ngược của IP.



Hình 9: Sơ đồ khối của DES

2.2 Giải thuật RSA

RSA, viết tắt tên của 3 tác giả **Rivest–Shamir–Adleman**, là một thuật mã hóa bất đối xứng. RSA sử dụng một biểu thức với số mũ. Plaintext được mã hóa theo khối, với mỗi khối có giá trị nhị phân nhỏ hơn một số n. Nghĩa là, kích thước khối phải nhỏ hơn hoặc bằng $log_2(n)+1$. Thực tế, kích thước khối là i bits, trong đó $2^i < n \leq 2^{i+i}$. Mã hóa và giải mã có dạng sau, với M là plaintext còn C là ciphertext.

$$C = M^e \text{ mod n}$$
$$M = C^d \text{ mod n} = (M^e)^d \text{ mod n} = M^{ed} \text{ mod n}$$

Cả người gửi và người nhận đều phải biết giá trị của n. Người gửi biết giá trị của e và chỉ người nhận biết giá trị của d. Do đó, đây là thuật toán mã hóa công khai với khóa chung là PU = e,n và khóa riêng là PR = d,n. Để thuật toán này đạt yêu cầu cho mã hóa khóa công khai, các yêu cầu sau phải được đáp ứng:

- Có thể tìm các giá trị của e, d và n sao cho M^{ed} mod n = M cho tất cả M < n.
- Việc tính toán M^e mod n và C^d mod n tương đối dễ dàng cho tất cả các giá trị của M < n.
- Không thể xác định d dù biết e và n.

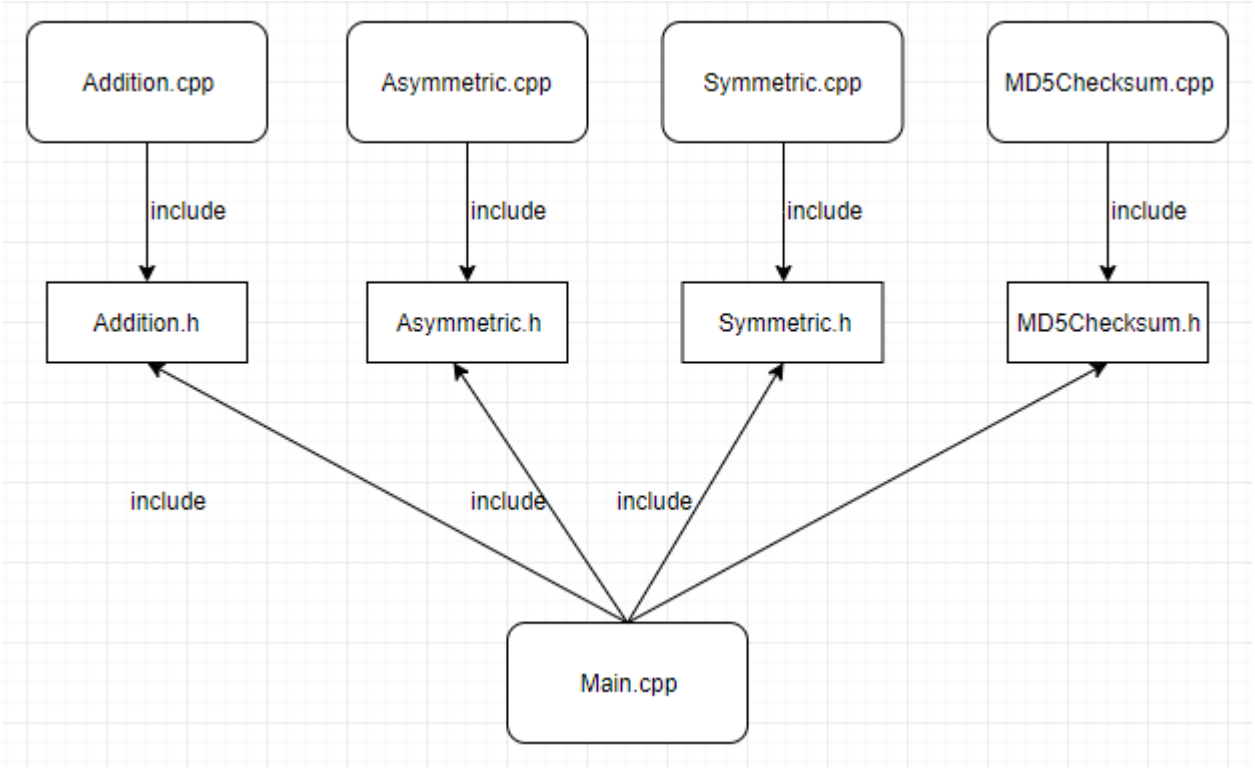
Việc thực hiện RSA được mô tả qua các bước sau:

- **Người nhận sinh key:**
  Chọn hai số p, q nguyên tố cùng nhau.
  Tính giá trị n = pq.
  Tính giá trị $\phi(n) = (p-1)(q-1)$.
  Chọn số nguyên e < $\phi(n)$ sao cho chúng nguyên tố cùng nhau.
  Tìm d thỏa $d \equiv e^{-1} \pmod{\phi(n)}$.
  Khóa công khai: PU = e,n
  Khóa riêng tư: PR = d,n
- **Người gửi mã hóa:** Sử dụng khóa công khai của người nhận để mã hóa plaintext M < n. Ciphertext: C = M^e mod n
- **Người nhận giải mã:** Người nhận dùng khóa riêng tư của mình để giải mã ciphertext C. Plaintext: M = C^d mod n

3 Cấu trúc của ứng dụng

Cấu trúc của ứng dụng được miêu tả trong hình 10: Chương trình gồm 5 mô đun chính là:

- Symmetric: Chứa các hàm mã hóa đối xứng tập tin sử dụng giải thuật DES.
- Asymmetric: Chứa các hàm mã hóa bất đối xứng tập tin sử dụng giải thuật RSA.
- Addition: Chứa các hàm mã hóa tập tin sử dụng kỹ thuật giấu tin (Steganography).
- MD5: Hàm MD5 sử dụng cho việc so sánh hai tệp.
- Main: Đọc các đối số và gọi các hàm phù hợp trong bốn thư viện trên.



Hình 10: Các mô đun của chương trình

3.1 Symmetric

Giải thuật mã hóa đối xứng nhóm sử dụng là DES với ECB mode. Trong tệp "Symmetric.h":

```
char *stringFromFile(char filename[], const char type[]);
```

Trả về chuỗi giá trị trong tệp filename.
type: chế độ đọc tệp ("r", "rb", ...).
Lưu ý: Cần giải phóng vùng nhớ cho con trỏ trả về do có sử dụng malloc trong hàm này.

```
unsigned char *DES_encrypt(EVP_CIPHER_CTX *en, unsigned char *plaintext, int *plain_l
```

Mã hó plaintext với độ dài plain_len theo giải thuật được quy định trong (*en). Trả về giá trị của chuỗi mã hóa. Lưu ý: Cần giải phóng vùng nhớ cho con trỏ trả về do có sử dụng malloc trong hàm này.

```
char *DES_decrypt(EVP_CIPHER_CTX *de, unsigned char *ciphertext, int *cipher_len);
```

Ngược với hàm trên, hàm này giải mã ciphertext với độ dài cipher_len.
Giá trị của (*de) quy định giải thuật sử dụng để giải mã.
Giá trị trả về là plaintext.
Lưu ý: Cần giải phóng vùng nhớ cho con trỏ trả về do có sử dụng malloc trong hàm này.

3.2 Asymmetric

Giải thuật mã hóa bất đối xứng nhóm chọn là RSA.Trong tệp "Asymmetric.h":

```
RSA * create_RSA(RSA *keypair, int pem_type, char *file_name);
```

Được sử dụng để tạo ra cặp key (private và public key) cho việc mã hóa và giải mã.
Key được lưu trong tệp file_name.

```
int public_encrypt(int flen, unsigned char* from, unsigned char *to, RSA* key, int pa
```

Sử dụng public key để mã hóa chuỗi (*from) với độ dài flen và lưu chuỗi mã hóa vào (*to). Trả về độ dài của chuỗi mã hóa.

```
int private_decrypt(int flen, unsigned char* from, unsigned char *to, RSA* key, int p
```

Sử dụng private key để giải mã chuỗi (*from) với độ dài flen và lưu vào chuỗi (*to).
Trả về giá trị độ dài của plaintext.

3.3 Addition

Giải thuật mã hóa ngoài bài giảng nhóm sử dụng là kỹ thuật che giấu tập tin. Trong tệp "Addition.h":

```
void steganography_encode(char* inputfile, Mat image, char*outputfile);
```

Đọc tệp inputfile dưới dạng bit.
Sử dụng phép tính "and" bit cho mỗi bit trong inputfile ứng với giá trị bit cuối trong mỗi kênh màu, pixels của ảnh image. Từ đó thu được một ảnh có sai khác rất nhỏ so với ảnh gốc.
Ghi ảnh kết quả vào tệp outputfile.

```
void steganography_decode(Mat image, char*outputfile);
```

Từ ảnh image, rút ra thông điệp và lưu nó vào outputfile.

3.4 MD5

Trong tệp "MD5Checksum.h":

```
unsigned char *getMd5Hash(unsigned char *data, size_t dataLen, int *mdLen);
```

Trả về giá trị Md5Hash của data với độ dài dataLen và độ dài của Md5 được lưu trong mdLen.
Lưu ý: Cần giải phóng vùng nhớ của con trỏ trả về.

3.5 Main

Hàm main chịu trách nhiệm đọc khác đối số khi chạy chương trình. Từ đó quyết định thực hiện giải thuật mã hóa nào. Nhóm có một số định nghĩa cho đối số đầu tiên như sau:

```
#define SYMMETRIC_ENCODE        10
#define SYMMETRIC_DECODE        11
#define RSA_CREATE                      20
#define RSA_ENCODE                      21
#define RSA_DECODE                      22
#define STREGAN_ENCODE          30
#define STREGAN_DECODE          31
#define MD5_CHECKSUM            40
```

Từ đây, với mỗi tùy chọn của đối số đầu tiên mà người dùng có thể lựa chọn tạo key, giải mã, mã hóa hay so sánh hai tệp.

Ngoài ra, trong hàm main, nhóm có gọi hàm clock() trước và sau khi thực hiện chương trình để đo thời gian mã hóa và giải mã.

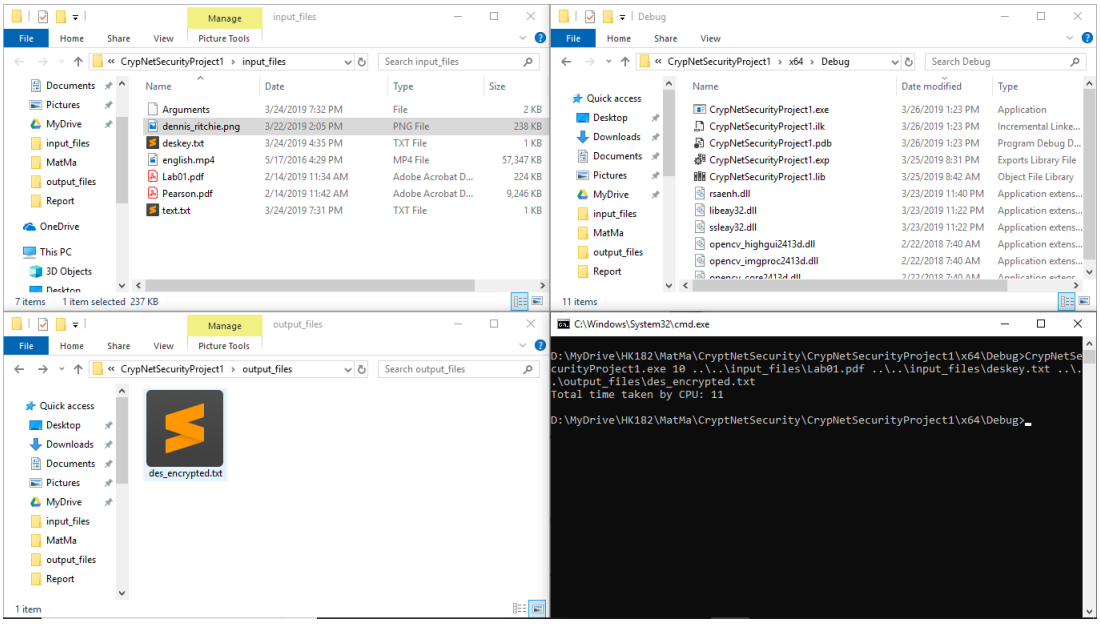
4 Chạy thử chương trình

4.1 Mã hóa DES

Chạy chương trình với command:
$File.exe 10 inputfile inputkeyfile outputfile
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.
- 10 : để lựa chọn thực thi mã hóa DES.
- inputfile: đường dẫn đến tệp cần mã hóa.
- inputkeyfile: đường dẫn đến tệp chứa key mã hóa.
- outputfile: đường dẫn đến tệp kết quả.

Ví dụ hình 11



Hình 11: Mã hóa DES

Từ hình 11, ta thấy mất 11ms để mã hóa tệp Lab01.pdf và tệp mã hóa (des_encrypted.txt) được tạo.

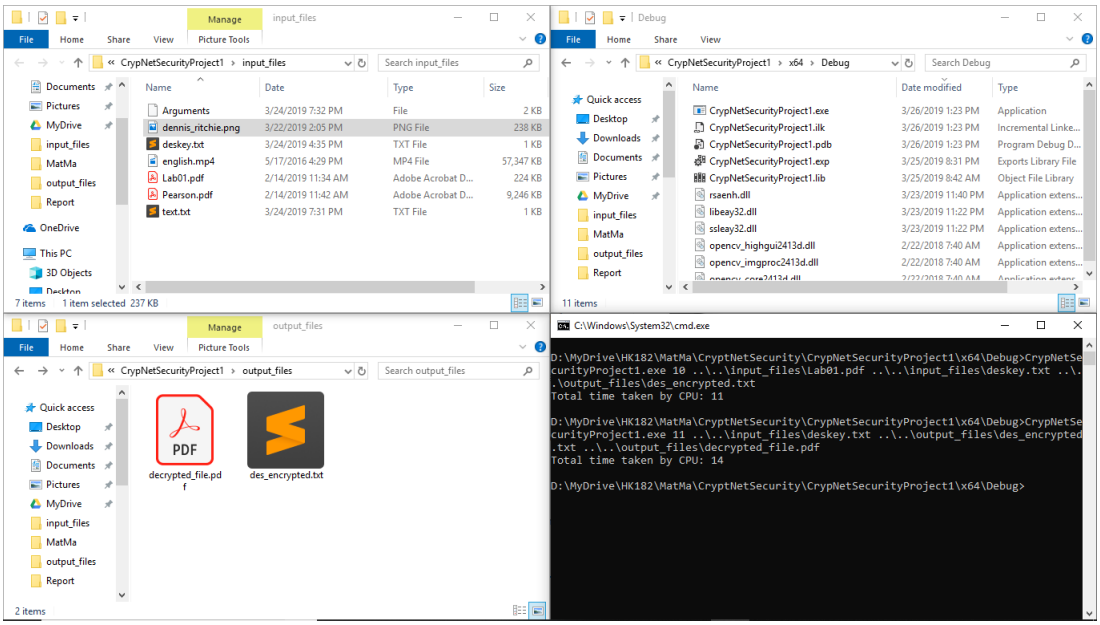
4.2 Giải mã DES

Chạy chương trình với command:
$File.exe 11 inputkeyfile inputfile outputfile
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.

- 11 : để lựa chọn thực thi giải mã DES.
- inputfile: đường dẫn đến tệp cần giải mã.
- inputkeyfile: đường dẫn đến tệp chứa key mã hóa.
- outputfile: đường dẫn đến tệp kết quả.

Ví dụ hình 12 Từ hình 12, ta thấy tệp des_encrypted.txt được giải mã trong vòng 14ms và



Hình 12: Giải mã DES

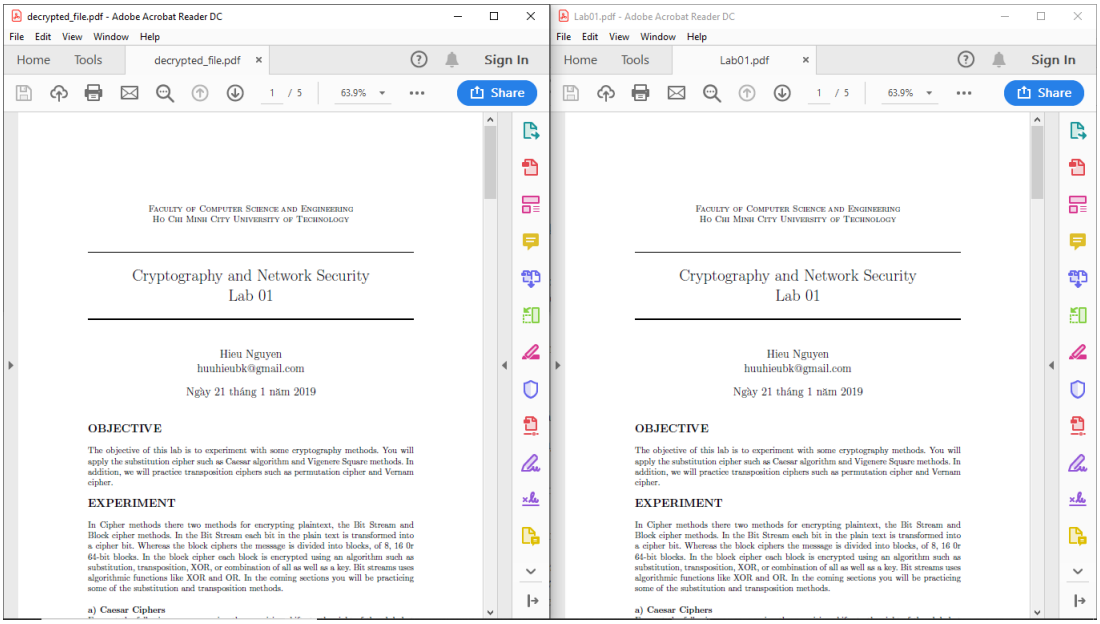
tạo ra tệp giải mã decrypted_file.pdf. Kết quả của giải thuật DES đã tạo ra tệp giải mã và được miêu tả ngắn gọn trong hình 13.

4.3 Tạo khóa RSA

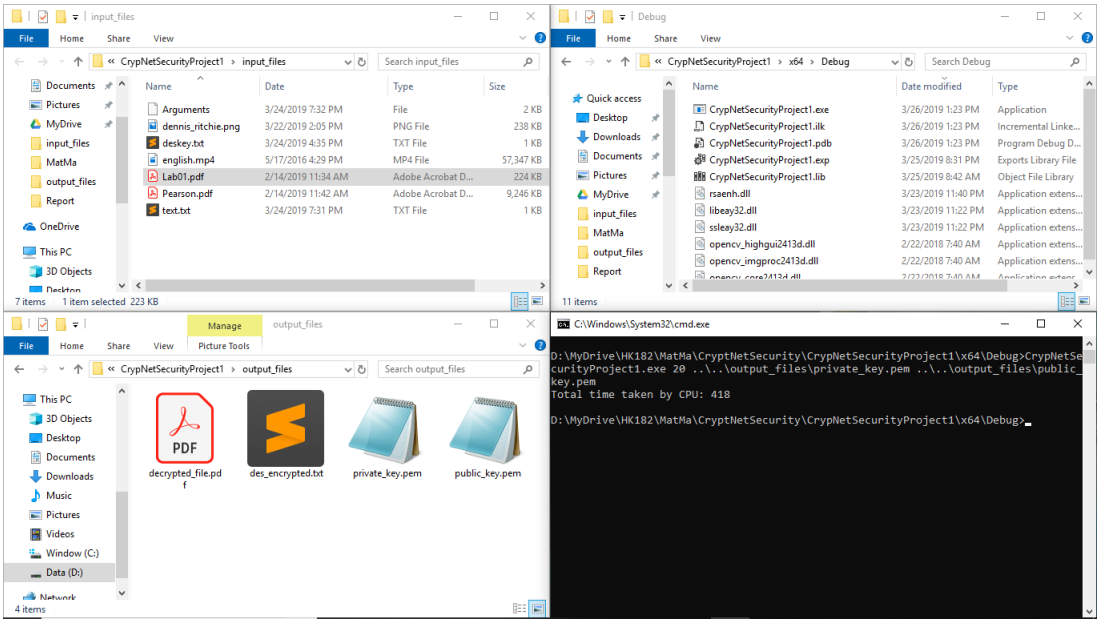
Chạy chương trình với command:
$File.exe 20 privateouput publicoutput
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.
- 20 : để lựa chọn thực thi tạo khóa bí mật và công khai.
- privateouput: đường dẫn đến tệp chứa khóa bí mật.
- publicoutput: đường dẫn đến tệp chứa khóa công khai.
- outputfile: đường dẫn đến tệp kết quả.

Ví dụ hình 14. Mất 418ms để tạo ra cặp khóa bí mật và công khai. Hai khóa trên ghi vào hai tệp: private_key.pem và public_key.pem.



Hình 13: Kết quả của giải thuật DES



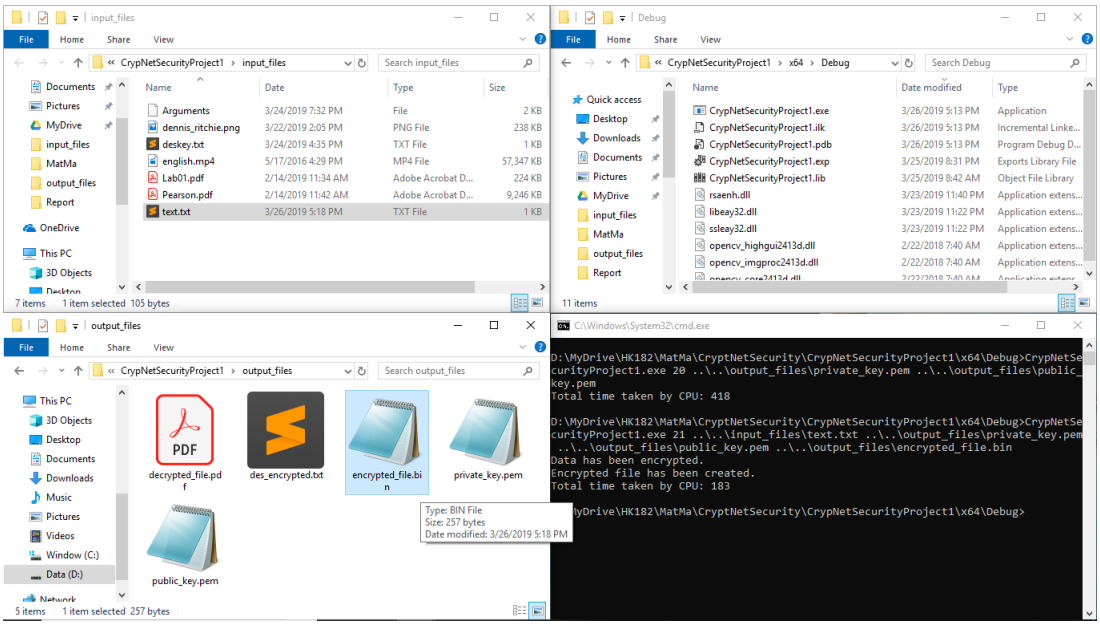
Hình 14: Tạo cặp khóa công khai và bí mật

4.4 Mã hóa RSA

Chạy chương trình với command:
$File.exe 21 inputfile privateouput publicoutput outputfile
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.
- 21 : để lựa chọn thực thi tạo khóa bí mật và công khai.
- inputfile: đường dẫn đến tệp cần mã hóa.
- privateouput: đường dẫn đến tệp chứa khóa bí mật.
- publicoutput: đường dẫn đến tệp chứa khóa công khai.

Ví dụ hình 15. Mất 183ms để mã hóa tệp text.txt. Kết quả mã hóa được ghi xuống tệp encrypted_file.bin



Hình 15: Tạo cặp khóa công khai và bí mật

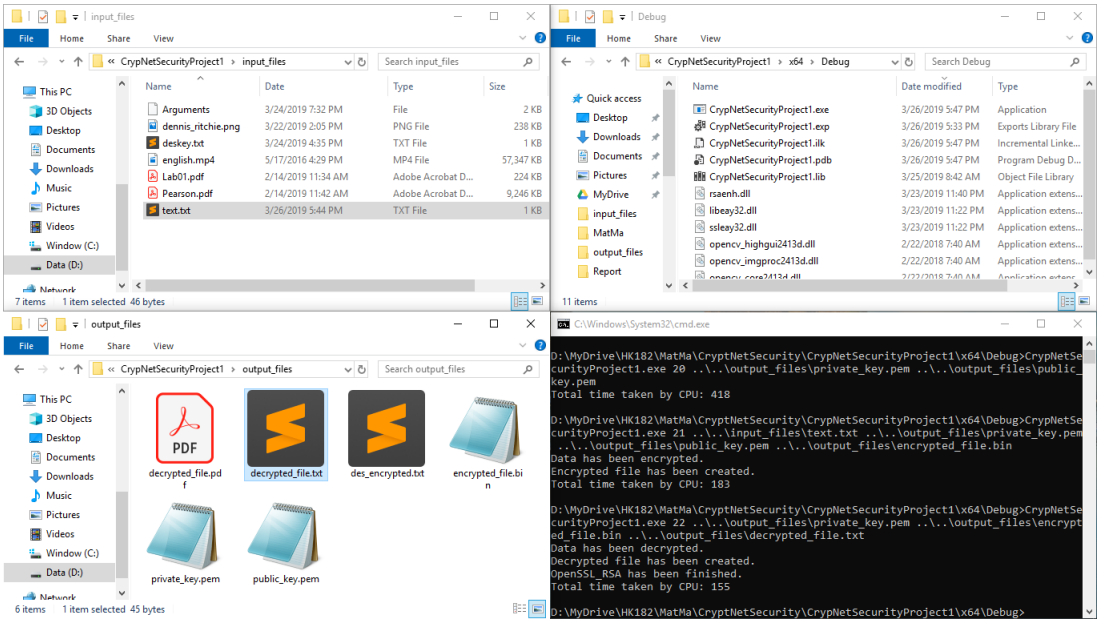
4.5 Giải mã RSA

Chạy chương trình với command:
$File.exe 22 privateinput inputfile outputfile
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.
- 22 : để lựa chọn thực thi mã hóa tệp theo giải thuật RSA.
- privateinput: đường dẫn đến tệp chứa khóa bí mật.

- inputfile: đường dẫn đến tệp cần giải mã.
- outputfile: đường dẫn đến tệp kết quả của phép giải mã.

Ví dụ hình 16. Mất 155ms để giải mã tệp encrypted_file.bin. Kết quả mã hóa được ghi xuống tệp decrypted_file.txt.
Hình 17 thể hiện trực quan so sánh giữa tệp ban đầu và tệp sau mã hóa.



Hình 16: Tạo cặp khóa công khai và bí mật

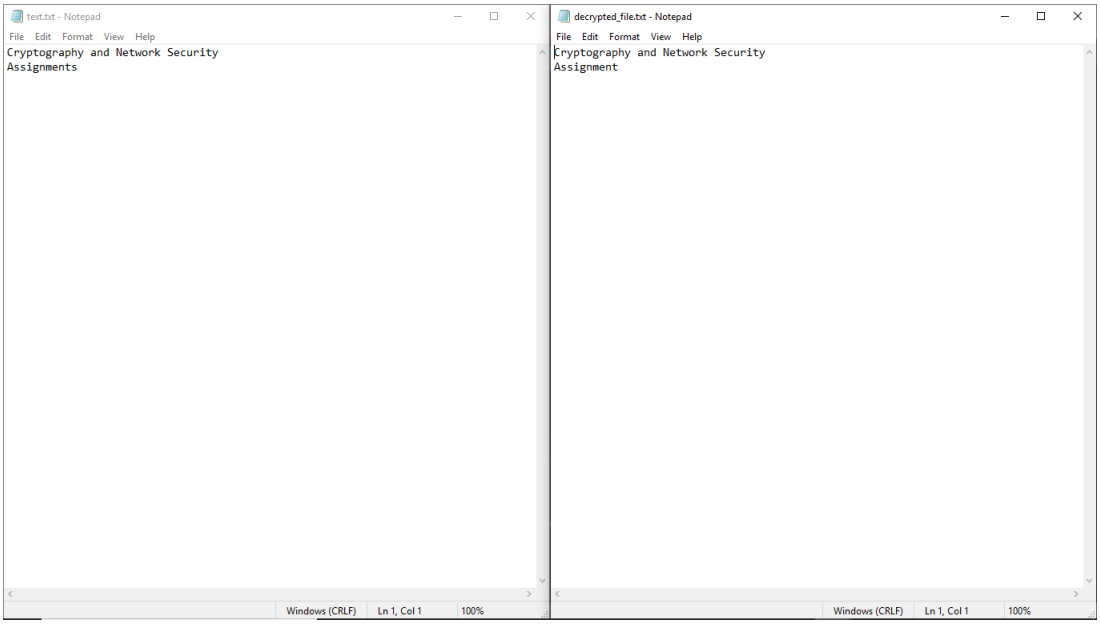
4.6 Kỹ thuật giấu tin

Chạy chương trình với command sau để gắn thông điệp vào ảnh mang:
$File.exe 30 inputfile1 inputfile2 outputfile
Trong đó:

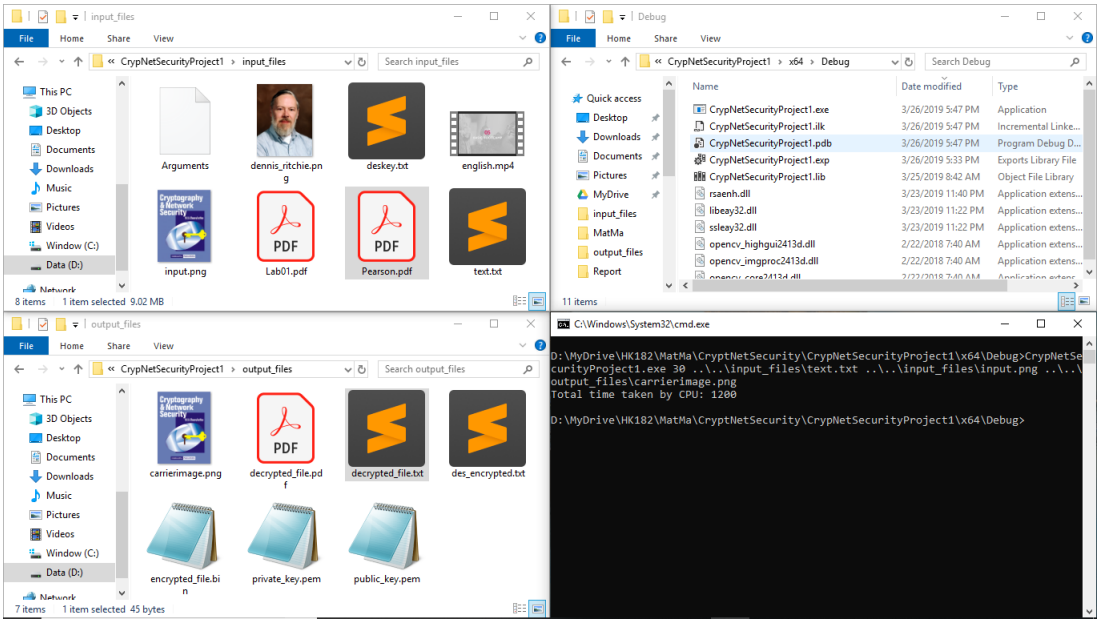
- File.exe : tệp thực thi được tạo từ MSVC.
- inputfile1: đường dẫn đến tệp chứa thông điệp.
- inputfile2: đường dẫn đến tệp chứa ảnh mang.
- outputfile: đường dẫn đến tệp kết quả của phép giấu tin.

Ví dụ hình 18
Chạy chương trình với command sau để gắn thông điệp vào ảnh mang:
$File.exe 30 inputfile outputfile
Trong đó:

- File.exe : tệp thực thi được tạo từ MSVC.
- inputfile: đường dẫn đến tệp chứa ảnh mang thông điệp.



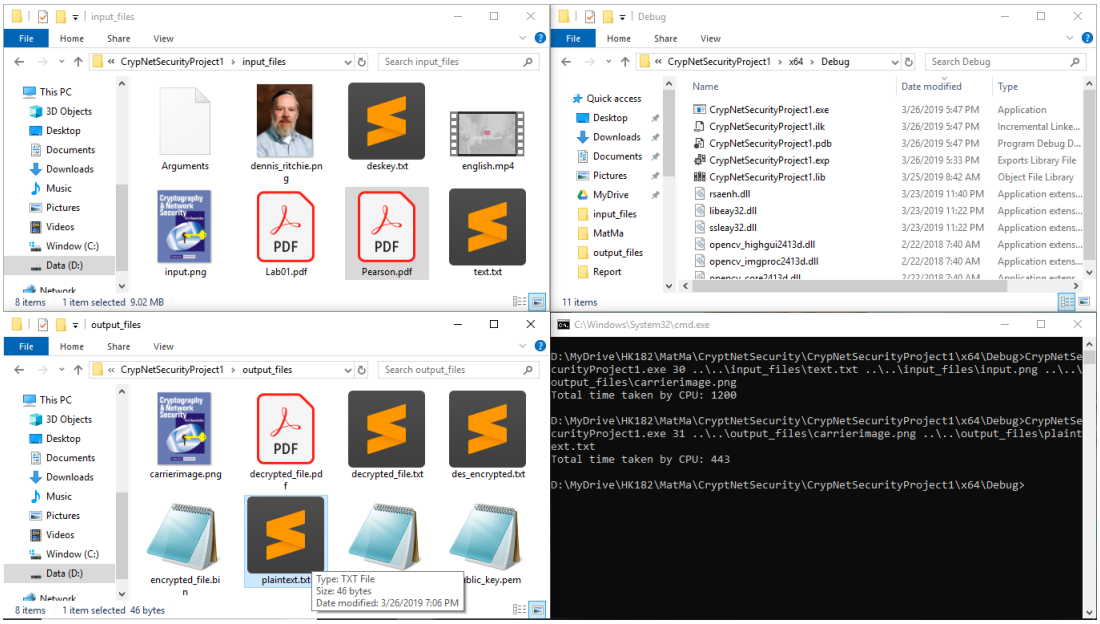
Hình 17: Tạo cặp khóa công khai và bí mật



Hình 18: Gắn thông điệp vào ảnh mang

- outputfile: đường dẫn đến tệp giải mã.

Ví dụ hình 19



Hình 19: Lấy thông điệp từ ảnh mang

Kết quả của quá trình giải mã và mã hóa được thể hiện qua hình 20

5 Hướng dẫn vận hành trên MSVC

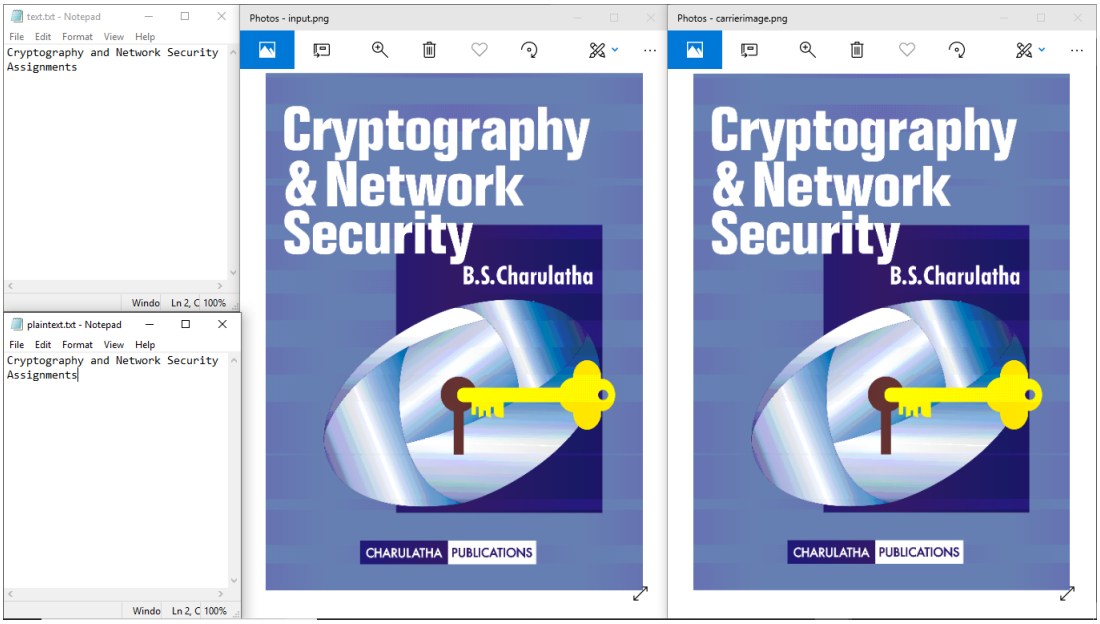
Bước 1: Cài đặt MSVC.
Bước 2: Cài đặt opencv.
Bước 3: Cài đặt opensll.
Bước 4: Tạo project trên MSVC.
Bước 5: Thiết lập đường dẫn đến thư viện opencv và openssl như hình 21 22 và 23
Bước 6: Tải mã nguồn tại urlhttps://github.com/nxnam714/CryptNetSecurity.git và gom vào project.

6 Phân tích và kết luận

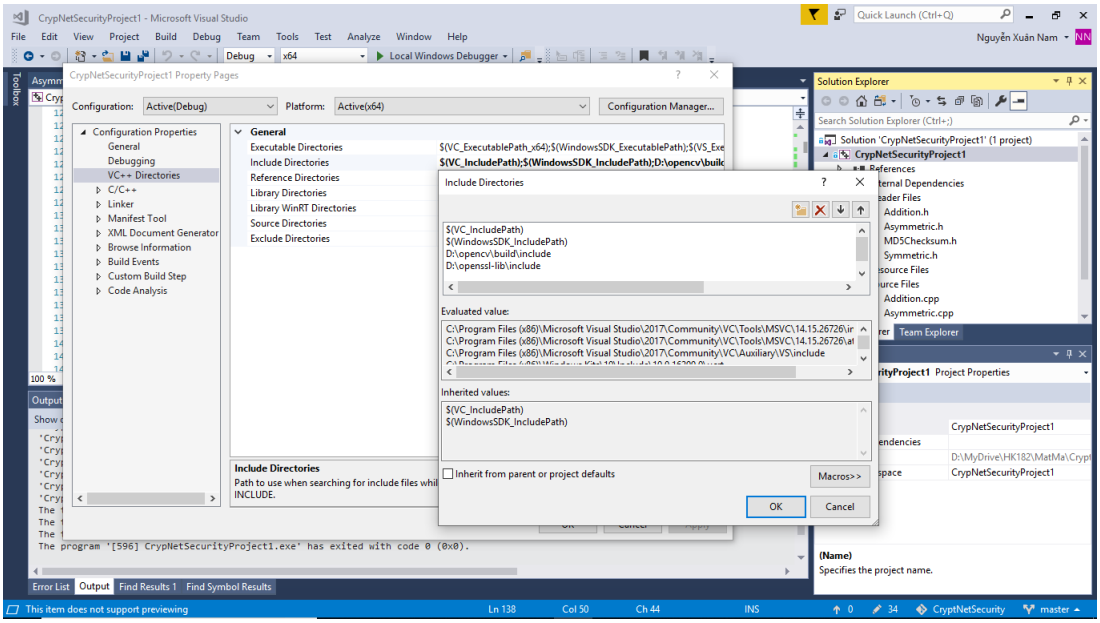
Ứng dụng có thể chạy tốt ba giải thuật mã hóa.

- Kết quả giải mã là đồng nhất với dữ liệu đưa vào dựa vào hàm kiểm tra MD5.
- Có thể mã hóa các tệp cơ bản như pdf, png, txt,...
- Thời gian thực thi nhanh.

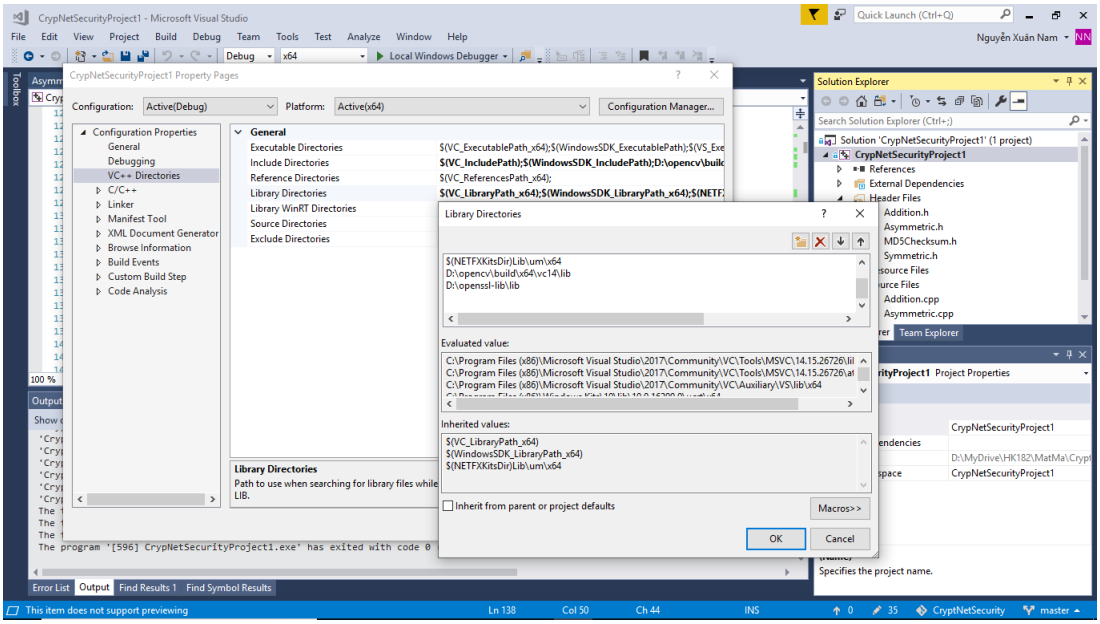
Tuy nhiên còn một số khuyết điểm:



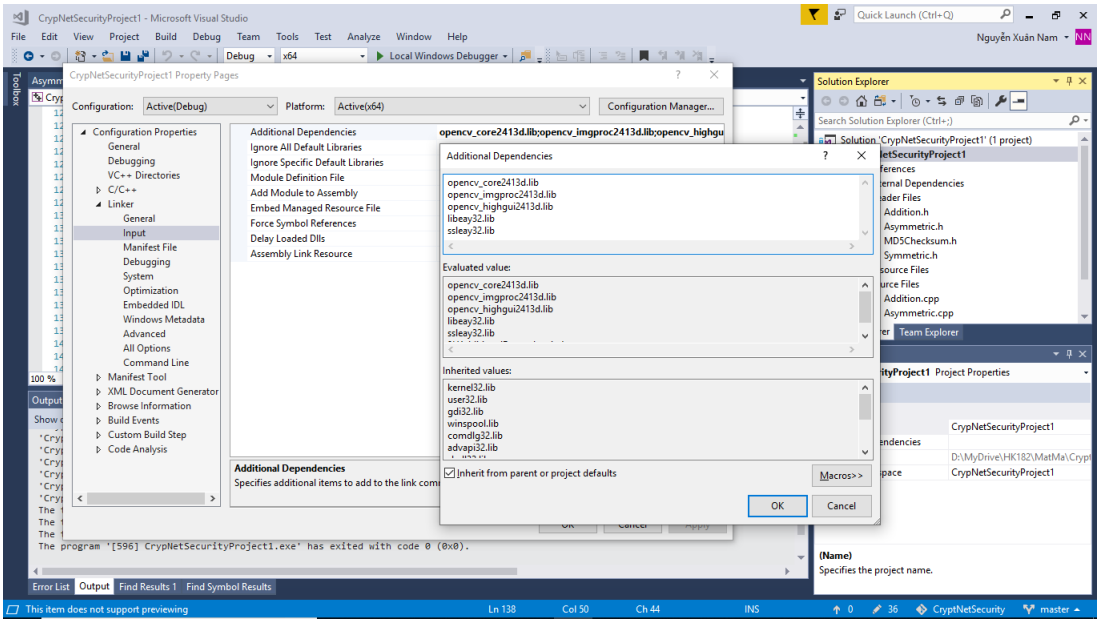
Hình 20: So sánh input và output của kỹ thuật giấu tin



Hình 21: Lấy thông điệp từ ảnh mang



Hình 22: Lấy thông điệp từ ảnh mang



Hình 23: Lấy thông điệp từ ảnh mang

- Tính thực tế của ứng dụng chưa cao, chỉ có thể sử dụng cho việc mã hóa các tệp trong máy. Đồng thời việc bảo khóa bí mật chưa được quan tâm đến.
- Ứng dụng không có giao diện nên tạo khó khăn cho việc sử dụng.
- Không có thanh trạng thái báo cho người dùng biết quá trình mã hóa đang diễn ra.

Hướng phát triển ứng dụng:

- Khắc phục các nhược điểm được nêu ra.
- Kết hợp các giải thuật mã hóa đã hiện thực để tao ứng dụng nhắn tin an toàn.

7 Phân công công việc

Khối lượng công việc được chia đều cho các thành viên trong nhóm như sau:

- Nguyễn Trần Lê Minh: Hiện thực khối DES.
- Lê Duy Thanh: Hiện thực khối RSA.
- Nguyễn Xuân Nam: Hiện thực khối MD5 và Steganography.

Tài liệu

[1] Thư viện openssl, https://www.openssl.org/docs/man1.0.2/

[2] Sách giáo khoa môn học: Stallings, William - Cryptography and network security principles and practice (2017, Pearson).