A ‘C’ Project On :

ENCRYPTION

AND

DECRYPTION

DONE BY :

P.S.Varada Desikan (imt2016115)

Tanmay jain (imt2016023)

Ayush saraswat (imt2016123)

**INDEX**

|  |  |  |
| --- | --- | --- |
| S.NO | HEADING | PAGE NO. |
| 1 | PROBLEM STATEMENT (with explanation) | 3 |
| 2 | SOURCE CODE | 11 |
| 3 | SCREENSHOTS | 26 |
| 4 | FUTURE IMPROVEMENTS | 28 |
| 5 | BIBILIOGRAPHY | 28 |

**PROBLEM STATEMENT**

This project, written in c language deals with basic encryption and decryption and possible attacks in 4 different ways. The ciphers are namely Substitution Cipher, Caesar Cipher, Vernam Cipher and Vignere Cipher where the user is given the option to choose whether he/she wants to encrypt a given file, decrypt it considering the fact whether the decryption is done if the user has or doesn’t have the key. The following explains each of the 4 ciphers :

1. **Substitution Cipher :**

This is one of the most secure way of encrypting a document amongst the basic ciphers. It randomly generates a key such that every character gets a unique key and thus the encrypted is stored along with the key. Another point to be noted is that the key so generated is different every time the program is run. This makes the code reliable and thus makes the data secure. For eg if a key of ‘\’ is given to the letter ‘e’, then the next time the user runs the program, it is highly improbable for the letter ‘e’ to get the same key ‘\’. By this method, each character is “SUBSTITUTED” by another value.

Decryption with the key is the straight reverse method of what is done during substitution. Since the key is given, it checks and replaces each and every value of the encrypted file using the key and stores the decrypted file. Since the key is already inputted, all the 255 characters can be used to decrypt the given file. Unlike few other ciphers, it can both encrypt and decrypt any ASCII character and thus widespread applications.

An attack on substitution cipher is when the user wants to decrypt a given file but doesn’t have the key. Hence there is a generalization called the frequency table which tells us the order of the frequency of occurrence of each and every alphabet. It is true that this method of attacking is not efficient and according to researchers, such an attack is correct only 1.69% of the time. The order and its frequency goes as follows :

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **E** | **T** | **A** | **O** | **I** | **N** | **S** | **H** | **R** | **D** | **L** | **U** | **C** |
| 12.7 | 9.1 | 8.2 | 7.5 | 7.0 | 6.7 | 6.3 | 6.1 | 6.0 | 4.3 | 4.0 | 2.8 | 2.8 |
| **M** | **W** | **F** | **Y** | **G** | **P** | **B** | **V** | **K** | **X** | **J** | **Q** | **Z** |
| 2.4 | 2.4 | 2.2 | 2.0 | 2.0 | 1.9 | 1.5 | 1.0 | 0.8 | 0.2 | 0.2 | 0.1 | 0.1 |

Example : “defend the east wall of the castle” text to be encrypted

If this is the alphabetical order key “lgkarxjpbshtoduwnvyefizqmc”

Then the decrypted text is “arxrda epr rlye zltt ux epr klyetr”

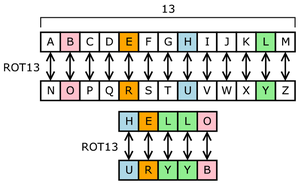
1. **Caesar Cipher :**

This method is one of oldest way of encrypting files. It can also be considered as a part of Substitution cipher in a very crude manner. It randomly generates a no. between 0 and 255 (both inclusive) and shifts each character by the no. and stores the character of the ASCII value of so created key. As in substitution cipher, each time, a random no. is generated. Since it just shifts and takes ASCII value into consideration, it can also take any of the 255 characters as input.

Decryption with key, shifts back each of the character by the no. of times the key is given. If the key is 5 and the ASCII of an example character is 3, it shifts back to 244. It is just the inverse operation and directly shifts back to its original value. It is not the most elegant of encrypting or decrypting a file, but can be considered as the easiest way of doing so.

An attack on Caesar cipher is one of the crudest methods of solving the attack. It runs through all the characters in the text and finds the character that occurs the most. It assumes that character is actually ‘e’ after decryption and assumes the key to be the ASCII of ‘e’ – ASCII of the character and thus changes every other character according to the given key.

Example :



1. **The Vernam Cipher.**

Also known as the ‘one time pad’, vernam cipher is mathematically the most secure cipher ever devised. All the ciphers are based on computational algorithms therefore in theory, only vernam cipher is the one which cannot be broken, given enough cipher text and time.

Using the vernam cipher.

* Vernam encryption is usually done with the help of a computer as it would take a lot amount of time to apply XOR operator through ASCII values.
* The text input or the message to be encrypted should be of the same length as the key is.
* No special characters or space is allowed in the in the vernam encryption, one reason is the space cannot be encrypted and the other is it can be easily compromised.
* A computer simplifies the process because the message and pad are encoded in binary. Each character is represented internally by a computer as a unique combination of zeros and ones called bits, for example the letter 'b' is composed of the eight bits '1100010'. This binary number is 98 in decimal. To encrypt the message each bit of each letter in the plaintext is combined with the corresponding letters' bit in the pad in sequence using a transformation called the bitwise exclusive or (abbreviated to XOR). This simply takes two bits as input and outputs a single bit according to the following scheme:

|  |  |  |
| --- | --- | --- |
| Input bits | | Output bit |
| Message | Pad |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* This operation is performed on each letter in sequence i.e. The first letter of the plaintext is XORed with the first letter of the pad to produce the first letter of the cipher text, then the second letter of the plaintext is XORed with the second letter of the pad to produce the second letter of the cipher text and so on.

Example

The principle of the Vernam Cipher is perhaps easier explained by looking at a message stored on a punched paper tape. In the example below, we want to transmit the word **HELLO** which is stored on the **plain text** tape. We also have a pre-recorded **key** tape, with a series of random characters; in this case the sequence **AXHJB**. The contents of the **plaintext** tape are now XOR-ed with the contents of the **key** tape. The result (KMIVE) is shown here as the **ciphertext** tape:

|  |
| --- |
| http://www.cryptomuseum.com/crypto/img/xor1.gif |
|  |
| *Mixing of the* ***plaintext*** *and the* ***key*** |

Now let us see what happens if we repeat this operation on the resulting **ciphertext** tape with the letters 'KMIVE'. In the illustration below, the **ciphertext** tape is on the left. It is XOR-ed with a copy of the original **key** tape (AXHJB), resulting in the original **plaintext**: 'HELLO'.

|  |
| --- |
| http://www.cryptomuseum.com/crypto/img/xor2.gif |
|  |
| *Mixing of the* ***ciphertext*** *and the* ***key*** |

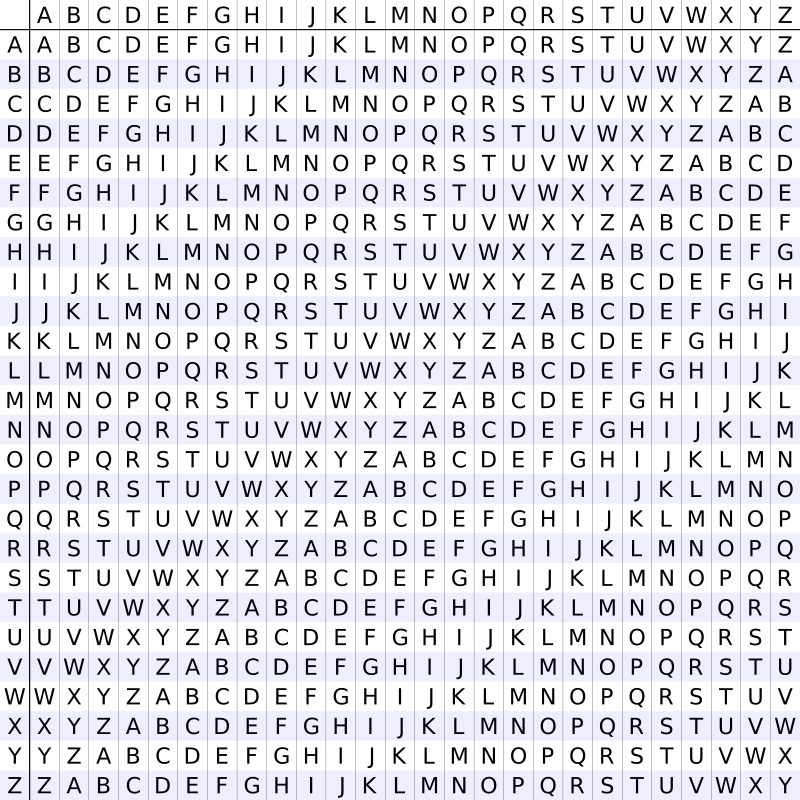
This process of applying the XOR-operation to **text** and **key** is often called **mixing**, and the cipher machines that use the Vernam principle, are therefore known as [mixers](http://www.cryptomuseum.com/crypto/mixer.htm). In the days when teletype systems were in widespread use, technicians were often so experienced that they could read a text directly from a paper tape, simply by looking at the holes and reading the bit patterns.

The same was often true for the maintenance engineers of OTT cipher machines. By taking a ciphertext tape, overlaying it with a key tape and then holding it against a bright light source, they were often able to 'read' the plaintext directly. This is illustrated in the above drawing, in which each half-transparent hole should be interpreted as as a binary '1'.

1. **THE VIGENERE CIPHER**

This cipher is a modification of the Caesar Cipher, which makes it more secure and stronger than the Caesar Cipher. It uses a series of Caesar Ciphers of different levels for encryption.

Basically, it uses a 26x26 table of alphabets with each row representing the possible Caesar Ciphers. That is, each row represents the 26 Caesar Ciphers possible with the 26 alphabets of the English language.



As shown in the image, each row has been shifted to the left by 1 element so as to generate all possible combination of Caesar Ciphers on the English Alphabet.

This table is called the Vigenere Square.

To encrypt, a repeated key is taken so that its length is equal to the length of the text to be encrypted. For eg, if the length of the text is 10 and the length of the key is 5. In this case, we repeat the key twice, so that now its length becomes 10.

If the length of the text is not a whole multiple of the length of the key, we eliminate the excess alphabets at the end of the key after repeating it the required number of the times. For e.g., if the length of the text is 10 and the length of the key is 4, we repeat the key thrice and then eliminate the last 2 alphabets so that now the length of the key becomes 10.

Now, as seen in the Vigenere Square, the alphabets of the English Language represent its rows and columns. Next comes the actual encryption part of the cipher. The first letter of the key and the text is taken and the element of the Vigenere Square corresponding to this row and column is the encrypted value for the first letter. Similarly, all the letters of the text are encrypted using the corresponding letters of the key. For example:

Consider the text: IAMBATMAN

And the key: RANG

Repeated key will be: RANGRANGR

Now,

I and R corresponds to element Z

A and A corresponds to element A

M and N corresponds to element Z

B and G corresponds to element H

A and R corresponds to element R

T and A corresponds to element T

M and N corresponds to element Z

A and G corresponds to element G

N and R corresponds to element E

So, the encrypted text will be ‘ZAZHRTZGE’!!!

For decryption with key, the same approach is followed as is followed while encrypting the text. The only difference is that, instead of knowing the row and column of the Vigenere Square and finding the element of the square, here the row and the element of the Vigenere Square are given and the column is needed.

The text is decrypted letter by letter and the original text is easily recovered. For example:

Consider the text: ZAZHRTZGE

And the key: RANG

Repeated key will be: RANGRANGR

Now,

Z and R corresponds to column I

A and A corresponds to column A

Z and N corresponds to column M

H and G corresponds to column B

R and R corresponds to column A

T and A corresponds to column T

Z and N corresponds to column M

G and G corresponds to column A

E and R corresponds to column N

So, the decrypted text will be ‘IAMBATMAN’!!!

The attacks on Vigenere Cipher are quite cumbersome as the same letter can be encoded with different letters and thus the frequency table method generally fails to decrypt the data.

This part of decryption without key is, however, not included in the code.

**SOURCE CODE**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<time.h>

char a[26][26];

int row(char c){

int i;

for(i=0;i<26;i++){

if(a[0][i]==c)

return(i);

}

}

int column(char c){

int i;

for(i=0;i<26;i++){

if(a[i][0]==c)

return(i);

}

}

int decrypt\_row(char c,int j){

int i;

for(i=0;i<26;i++){

if(a[i][j]==c)

return(i);

}

}

void vernam\_encrypt(){

int n;

//scanf("%d",&n);

char input[400], key[400],e\_m[400];

int i,j,value,x,y;

value =96;

for(i=0;i<26;i++){

value++;

for(j=0;j<26;j++){

a[i][j]=value++;

if(value == 123)

value =97;

}

}

FILE \*f1, \*f2, \*f3;

f1 = fopen("Text.txt","r");

f2 = fopen("Key.txt","r");

f3 = fopen("Encrypted.txt", "w");

fscanf(f1,"%s",input);

fscanf(f2,"%s",key);

n=strlen(input);

for(i=0;i<n;i++){

if(key[i]!='\0'){

x=row(key[i]);

y=column(input[i]);

e\_m[i]=a[y][x];

}

else

break;

if(input[i]==EOF)

break;

}

e\_m[i]=EOF;

fprintf(f3,"%s",e\_m);

fclose(f1);

fclose(f2);

fclose(f3);

}

void vernam\_decrypt(){

int n;

//char a[26][26];

char input[400],key[400],e\_m[400],d\_m[400];

int i,j,value,x,y;

value =96;

for(i=0;i<26;i++){

value++;

for(j=0;j<26;j++){

a[i][j]=value++;

if(value == 123)

value =97;

}

}

FILE \*f4, \*f5, \*f6;

f4 = fopen("Encrypted.txt","r");

f5 = fopen("Key.txt","r");

f6 = fopen("Decrypted.txt", "w");

fscanf(f4,"%s",e\_m);

fscanf(f5,"%s",key);

n=strlen(e\_m);

for(i=0;i<n;i++){

y=column(key[i]);

x=decrypt\_row(e\_m[i],y);

d\_m[i]=a[x][0];

}

fprintf(f6,"%s",d\_m);

fclose(f4);

fclose(f5);

fclose(f6);

}

void Sub\_encrypt(){

FILE \*fp, \*fp1, \*fp2;

fp=fopen("Text.txt", "r");

fp1=fopen("Encrypted.txt", "w");

fp2=fopen("Key.txt", "w");

char in;

int i, n, key[256];

for(i=0;i<=255;i++)

key[i]=0;

srand(time(NULL));

in=fgetc(fp);

while(in!=EOF){

int j=(int)in-1;

if(key[j]==0){

key[j]=rand()%255+1;

for(i=0;i<255;i++){

if(key[i]==key[j] && i!=j){

key[j]++;

if(key[j]==256)

key[j]=0;

i=0;

}

}

}

fputc(key[j], fp1);

in=fgetc(fp);

}

for(i=0;i<256;i++)

fputc((char)key[i], fp2);

fclose(fp1);

fclose(fp2);

fclose(fp);

}

void Sub\_decryptkey(){

FILE \*fp, \*fp1, \*fp2;

fp1=fopen("Encrypted.txt", "r");

fp2=fopen("Key.txt", "r");

fp=fopen("Decrypted.txt", "w");

char in;

int i, key[256], j;

for(i=0;i<256;i++){

key[i]=(int)fgetc(fp2);

}

fclose(fp2);

in=fgetc(fp1);

while(in!=EOF){

j=((int)in+256)%256;

for(i=0;i<256;i++){

if(key[i]==j){

fputc((char)(i+1),fp);

//printf("%c\n", (char)(i+1));

}

}

//fputc(key[(int)in-1],fp);

in=fgetc(fp1);

}

fclose(fp1);

//fclose(fp2);

fclose(fp);

}

void DecryptNo(){

FILE \*fp, \*fp1, \*fp2;

fp=fopen("Encrypted.txt", "r");

fp1=fopen("Decrypted.txt", "w");

char in, in1, out[26], tab[26]={'e', 't', 'a', 'o', 'i', 'n', 's', 'r', 'h', 'd', 'l', 'u', 'c', 'm', 'f', 'y', 'w', 'g', 'p', 'b', 'v', 'k', 'x', 'q', 'j', 'z'};

int i, a[26], x, l, r=0, j;

in=fgetc(fp);

in1=in;

for(i=0;i<26;i++){

a[i]=0;

out[i]='0';

}

while(in!=EOF){

in=toupper(in);

if((int)in>=65 && (int)in<91){

a[in-65]++;

}

in=fgetc(fp);

}

for(i=0;i<26;i++){

x=0;

for(j=0;j<26;j++){

if(a[j]>x && out[j]=='0'){

x=a[j];

l=j;

}

}

out[l]=tab[r];

r++;

}

printf("\n");

fclose(fp);

fp=fopen("Encrypted.txt", "r");

in=fgetc(fp);

while(in!=EOF){

in=toupper(in);

if((int)in>=65 && (int)in<91){

fputc(out[in-65], fp1);

}

else

fputc(in, fp1);

in=fgetc(fp);

}

fclose(fp);

fclose(fp1);

}

void Caesar\_encrypt(){

FILE \*fp, \*fp1, \*fp2;

fp=fopen("Text.txt", "r");

fp1=fopen("Encrypted.txt", "w");

fp2=fopen("Key.txt", "w");

char in, out;

int i, n, key;

srand(time(NULL));

key=rand()%256;

in=fgetc(fp);

while(in!=EOF){

fputc((key+(int)in)%256, fp1);

in=fgetc(fp);

}

fputc(key, fp2);

fclose(fp);

fclose(fp1);

fclose(fp2);

}

void Caesar\_decrypt(){

FILE \*fp, \*fp1, \*fp2;

fp=fopen("Decrypted.txt", "w");

fp1=fopen("Encrypted.txt", "r");

fp2=fopen("Key.txt", "r");

char in, out;

int i, n, key;

key=(int)fgetc(fp2);

in=fgetc(fp1);

while(in!=EOF){

out=(int)in-key;

if((int)in-key<0)

out+=256;

fputc(out,fp);

in=fgetc(fp1);

}

fclose(fp1);

fclose(fp2);

fclose(fp);

}

void Caesar\_decryptNo(){

FILE \*fp, \*fp1;

fp=fopen("Decrypted.txt", "w");

fp1=fopen("Encrypted.txt", "r");

char in, in1;

int a[26], i, x=0, l;

for(i=0;i<26;i++)

a[i]=0;

in=fgetc(fp1);

while(in!=EOF){

in1=toupper(in);

if((int)in1>=65 && (int)in1<91){

a[(int)in1-65]++;

}

in=fgetc(fp1);

}

for(i=0;i<26;i++){

printf("%d ", a[i]);

if(a[i]>x){

x=a[i];

l=i;

}

}

x=l-4;

fclose(fp1);

fp1=fopen("Encrypted.txt", "r");

in=fgetc(fp1);

while(in!=EOF){

in=toupper(in);

fputc(((int)in-65-x>=0? in-x:in+x), fp);

in=fgetc(fp1);

}

fclose(fp1);

fclose(fp);

}

void Vignere\_encrypt(){

char v\_sqr[26][26];

char temp[26]= {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

int i,j;

for(i=0;i<26;i++){

for(j=0;j<26;j++){

v\_sqr[i][j]=temp[(j+i)%26];

}

}

FILE \*fin, \*fout,\*fkey;

fkey=fopen("Key.txt","w");

fout=fopen("Encrypted.txt","w");

fin = fopen("Text.txt","r");

int n;

printf("Enter the length of the keyword: ");

scanf("%d",&n);

char keyword[n];

int g;

srand(time(NULL));

for (g=0;g<n;g++){

int r=rand()%26;

keyword[g]=temp[r];

fprintf(fkey,"%c",keyword[g]);

}

if (fin){

char str[1024];

while (fscanf(fin, "%s", str) != EOF){

int x,y,z;

char h[strlen(str)];

for (x=0; x<strlen(str); x++){

if(str[x]=='\0'){

break;

}

else{

y=(int)str[x];

z=(int)keyword[x % n];

h[x]=v\_sqr[y-65][z-65];

}

}

fprintf(fout,"%s",h);

}

}

else{

printf("No data found to encrypt");

}

fclose(fin);

fclose(fout);

fclose(fkey);

}

void Vignere\_decrypt(){

char v\_sqr[26][26];

char temp[26]= {'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

int i,j;

for(i=0;i<26;i++){

for(j=0;j<26;j++){

v\_sqr[i][j]=temp[(j+i)%26];

}

}

FILE \*fin, \*fout,\*fkey;

fin = fopen("Encrypted.txt","r");

fout = fopen("Decrypted.txt","w");

fkey = fopen("Key.txt","r");

char keyword[1024];

fscanf(fkey,"%s",keyword);

if (fin){

char str[1024];

while (fscanf(fin, "%s", str) != EOF){

int x,y,z;

char h[strlen(str)];

for (x=0;x<strlen(str);x++){

for (z=0;z<26;z++){

y=(int)keyword[x % strlen(keyword)];

if (v\_sqr[y-65][z] == str[x]){

h[x]=v\_sqr[0][z];

}

}

}

fprintf(fout,"%s",h);

}

}

}

int main(){

//Sub\_encrypt();

int x=1, x1, x2;

printf("Enter \n1 : Encryption\n2 : Decryption (if you have key)\n3 : Decryption (if you don't have key)\n4 : Instructions\n");

scanf("%d", &x);

switch(x){

case(1):{

printf("Enter\n1 : Substitution\n2 : Caesar\n3 : Vernam\n4 : Vigenere\n");

scanf("%d", &x1);

if(x1){

printf("Enter the text to be encrypted to 'Text.txt' file. Press 1 when done : ");

scanf("%d", &x2);

if(x2){

if(x1==1){

Sub\_encrypt();

printf("Your file has been successfully encrypted. Check 'Key.txt' for the key and 'Encrypted.txt' for the encrypted file");

break;

}

else if(x1==2){

Caesar\_encrypt();

printf("Your file has been successfully encrypted. Check 'Key.txt' for the key and 'Encrypted.txt' for the encrypted file");

break;

}

else if(x1==3){

vernam\_encrypt();

printf("Your file has been successfully encrypted. Check 'Encrypted.txt' for the encrypted file");

break;

}

else if(x1==4){

Vignere\_encrypt();

printf("Your file has been successfully encrypted. Check 'Key.txt' for the key and 'Encrypted.txt' for the encrypted file");

break;

}

else{

printf("Invalid Input!!!");

}

}

}

break;

}

case(2):{

printf("Enter\n1 : Substitution\n2 : Caesar\n3 : Vernam\n4 : Vigenere\n");

scanf("%d", &x1);

if(x1){

printf("Enter the text to be decrypted to 'Encrypted.txt' file and the key to 'Key.txt' file. Press 1 when done : ");

scanf("%d", &x2);

if(x2){

if(x1==1){

Sub\_decryptkey();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else if(x1==2){

Caesar\_decrypt();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else if(x1==3){

vernam\_decrypt();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else if(x1==4){

Vignere\_decrypt();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else{

printf("Invalid Input!!!");

}

}

}

break;

}

case(3):{

printf("Enter\n1 : Substitution\n2 : Caesar\n");

scanf("%d", &x1);

if(x1){

printf("Enter the text to be decrypted to 'Encrypted.txt' file. Press 1 when done : ");

scanf("%d", &x2);

if(x2){

if(x1==1){

DecryptNo();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else if(x1==2){

Caesar\_decryptNo();

printf("Your file has been successfully decrypted. Check 'Decrypted.txt' for the decrypted file");

break;

}

else{

printf("Invalid Input!!!");

}

}

}

}

case(4):{

FILE \*f1;

f1=fopen("Instructions.txt", "r");

char in;

in=fgetc(f1);

while(in!=EOF){

printf("%c", in);

in=fgetc(f1);

}

fclose(f1);

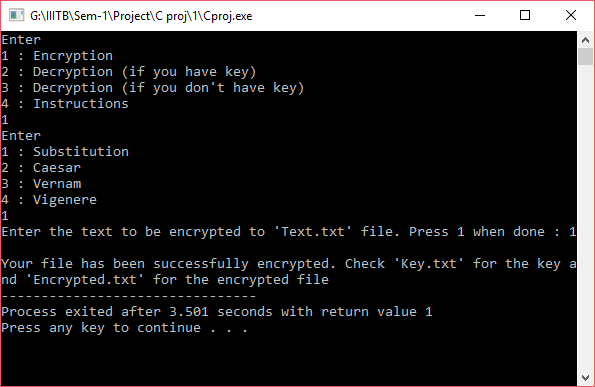
break;

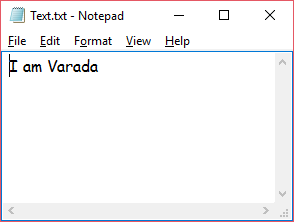
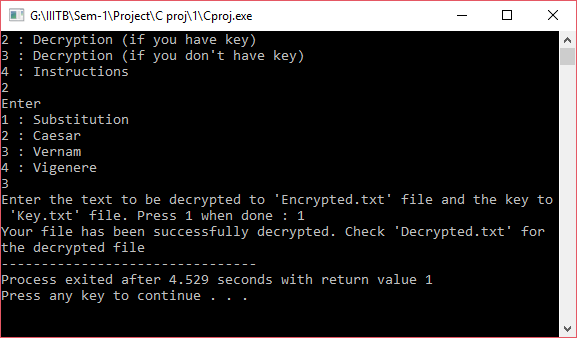
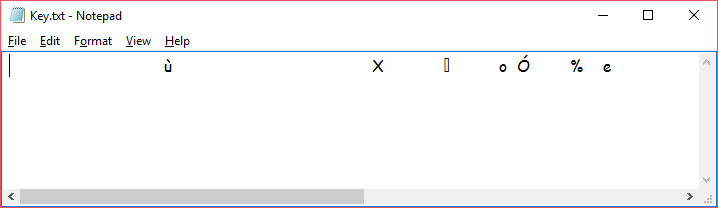
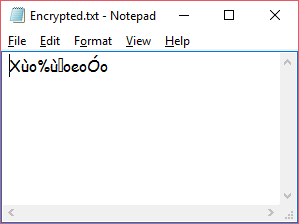
}

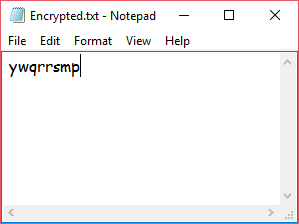
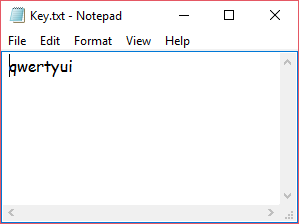
}

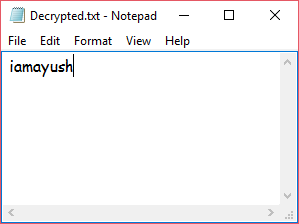
}

**SCREENSHOTS**





**FUTURE IMPROVEMENTS**

Substitution Cipher : The probability of the attack being right is low. It can be improved by checking with a dictionary.

Caesar Cipher : The probability of the attack being right is low. It can be improved by checking with a dictionary.

Vernam Cipher : Other characters can also be inputted if the code is modified. The junk character can be removed.

Vignere Cipher : Other characters can also be inputted. The junk character can be removed

**BIBILIOGRAPHY**

Stackoverflow.com

Wikipedia

practicalcryptography.com