ENCRYPTION TECHNIQUES

Submitted in Partial Fulfillment of the Recruitment for the Degree of

Bachelor of Technology

In

Computer Science & Engineering

2020

Submitted to

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

NOIDA (U.P)

Submitted by

HARIT MOHAN (19103129)
KARTIK DHOLAKIA (19103135)
SPARSH GUPTA (19103138)

Under the guidance of

ANKITA WADHWA

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

YEAR – 2020

DECLARATION

We hereby declare that the project report entitled- “Encryption Techniques” which is being submitted as Mini Project of 3rd Semester in Computer Science & Engineering to Jaypee Institute of Information Technology (JIIT), Sector-62, Noida is an authentic record of our genuine work done under the guidance of Prof. Ankita Wadhwa, Department of Computer Science Engineering Technology, JIIT, Noida.

HARIT MOHAN

(19103129)

KARTIK DHOLAKIA

(19103135)

SPARSH GUPTA

(19103138)

Date :- 13/12/2020

Place :- Noida

ACKNOWLEDGEMENT

With our sincere regards, we wish to acknowledge our indebtedness and gratitude for the contributions of people who helped us at every stage of the project.

We are very much like to express our gratitude and profound thanks to our project guide Prof. Ankita Wadhwa, Department of Computer Science Engineering, JIIT, Noida for her kind approval of the project, sustained guidance, invaluable suggestions, and constant encouragement without which it would not have been possible for us to complete this project.

TABLE OF CONTENTS

Introduction / Problem Statement

In today's world of ubiquitous computers and networks, it's hard to overstate the value of encryption. Quite simply, encryption keeps you safe. Encryption protects your financial details and passwords when you bank online. It protects your cell phone conversations from eavesdroppers. If you encrypt your laptop — and I hope you do — it protects your data if your computer is stolen. It protects your money and your privacy. Encryption protects the identity of dissidents all over the world. It's a vital tool to allow journalists to communicate securely with their sources, NGOs to protect their work in repressive countries, and attorneys to communicate privately with their clients.

So, we have made a project in which we use various encryption techniques, which encrypts our messages into audio files, image files and also encodes our messages using various ciphers and encryption techniques, build within a clean and elegant UI made using Qt and written in C++. Only the person having our software can correctly decrypt the message and use it for beneficial usage. We also have used the OOPs functionality of C++ for restricting access to the private and protected members from outside access.

OOPs and Concepts Used

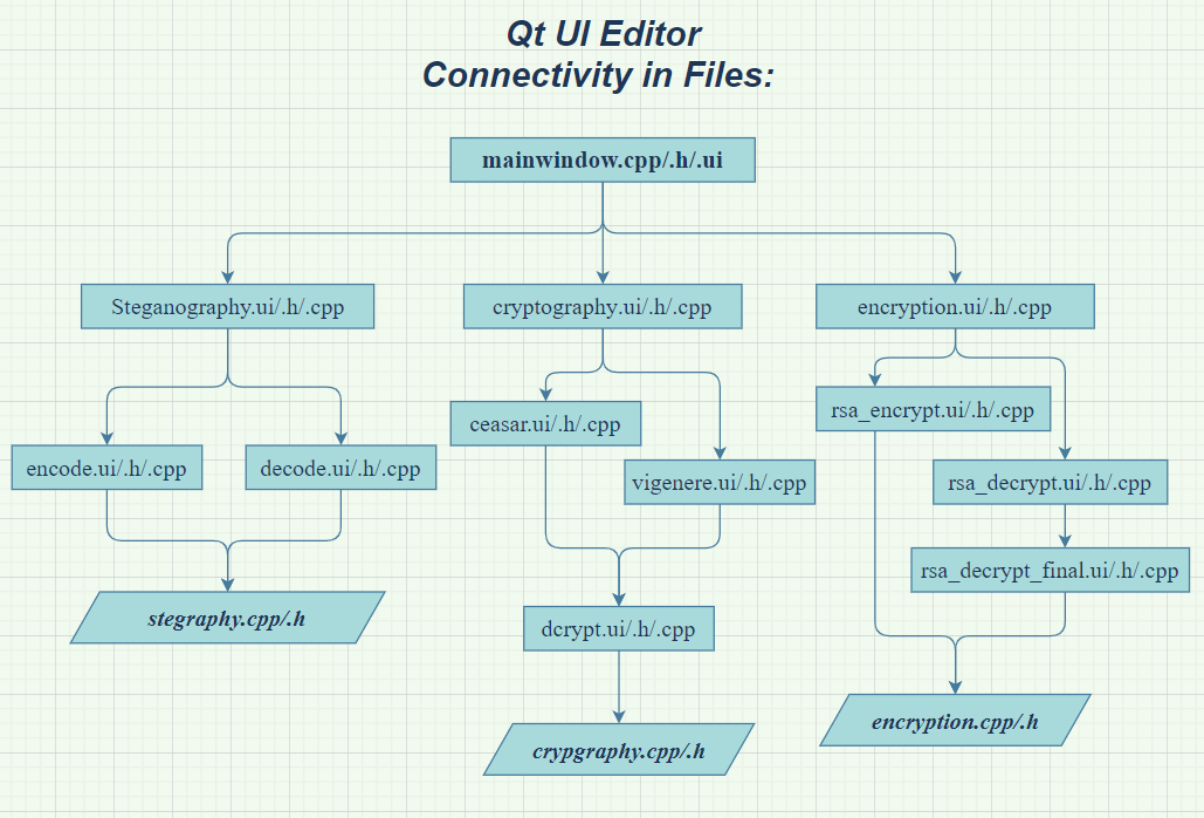
1. Virtual functions
2. Inheritance
3. Abstract classes
4. Polymorphism
5. Function overloading
6. Dynamic Binding
7. File Handling
8. Hashing

Requirements

1. OS - Windows 7 / 8 / 10
2. RAM - 256 MB
3. Disk Space – 100 MB
4. Processor – Intel core 2 duo

Detailed Design of the project:

Design of Qt Files:



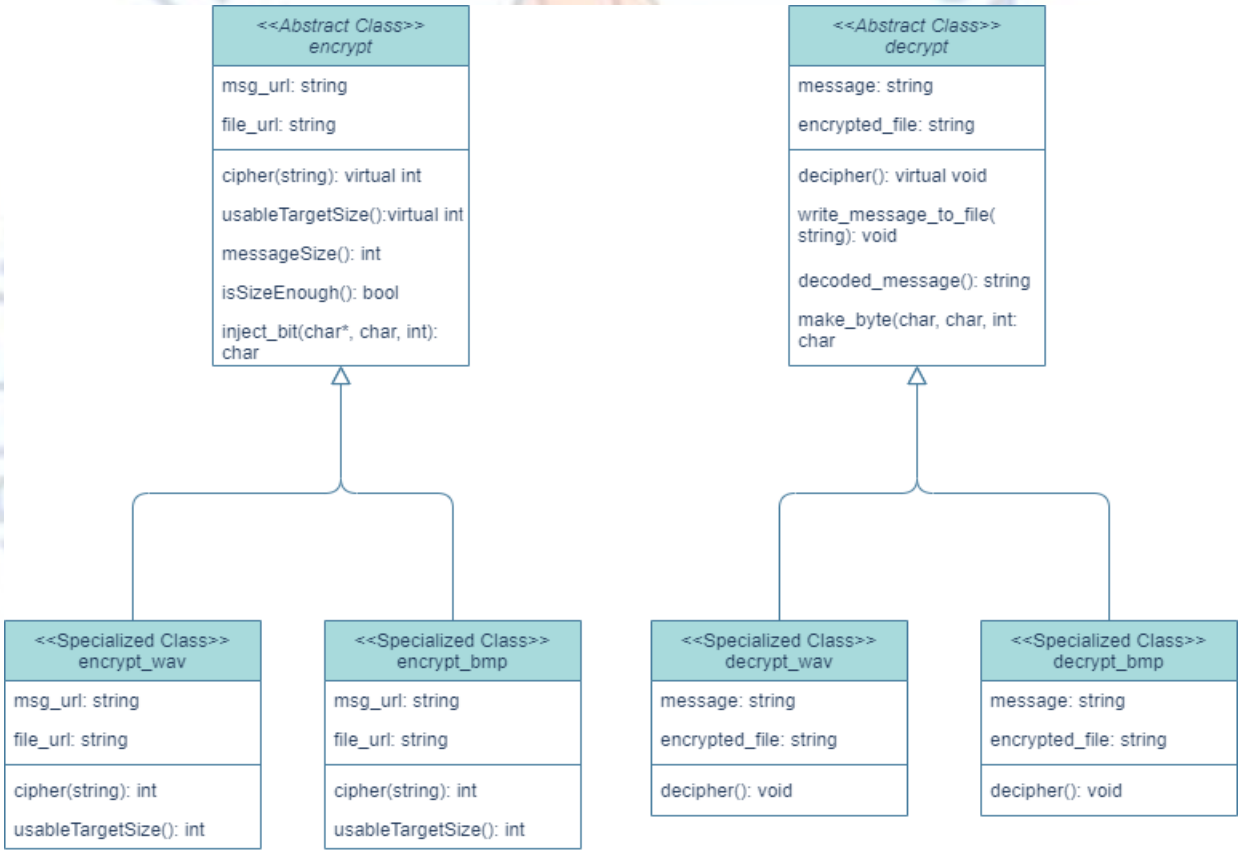
*All the files except the last files have a .ui, .h, and .cpp files.

All the .ui files contain forms which are displayed in our software when the program runs and the .cpp files related to those.

Forms enable the user to interact with the code written in the header and cpp files (provided at last), and perform the different operations for encryption, decryption, etc.

Design of stegraphy.h and stegraphy.cpp:

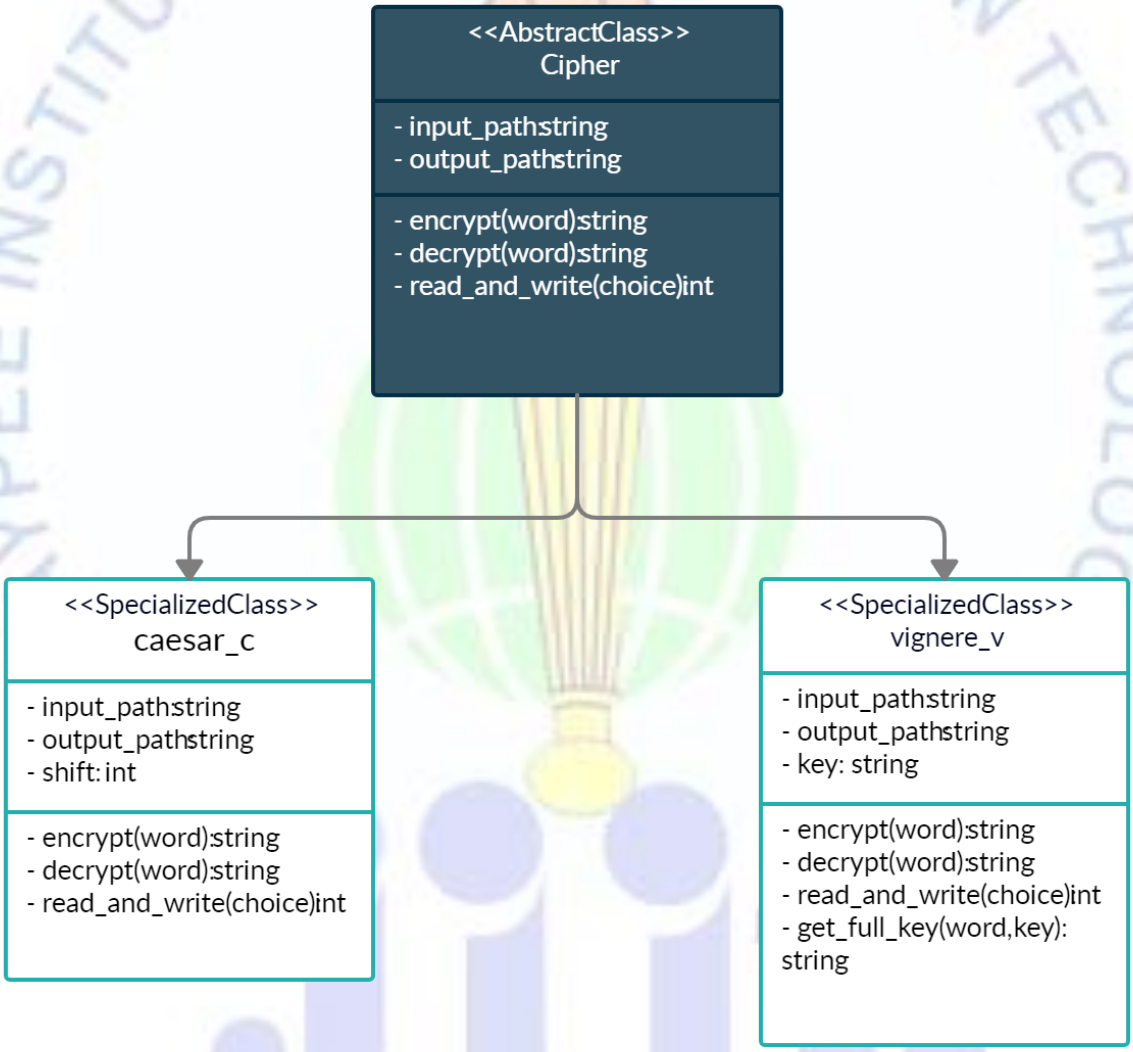
These files contain the classes and their implementation to encode the text inside of an image and an audio file. The Class Diagram is as follows:



*We have used an external library file cimg.h for processing of bmp image files.

Design of crypgraphy.h and crypgraphy.cpp:

These files contain classes and their implementation to encrypt the text message using Caesar and Vignere ciphers. The class diagram is given below:



Implementation Details:

Function definitions of:

int encrypt_wav::usableTargetSize()

```
{
    int size(0);
    ifstream file(file_url,ios::binary|ios::in);
    file.seekg(44);
    char c;
    while(!file.eof()) {
        size++;
        file>>c;
    }
    size/=8;                        //one byte stored in LSBs of 8 bytes
    return size/(SKIP+1);           //see the encryption
}
```

int *encrypt_bmp*::usableTargetSize()

```
{
    bytecimg target(file_url.c_str());
    bytecimg::iterator it = target.begin();
    long c(0);
    while(it!=target.end()){
        it++;c++;
    }
    return c;       // In bytes...
}
```

int *encrypt_wav*::cipher(*string filename*="copy")

```
{
    ifstream target(file_url,ios::binary|ios::in);
    ifstream message(msg_url,ios::in);
    ofstream encrypted(filename,ios::binary|ios::out);
    char t[40],c,temp[8];
    string m("");
    if(!target.is_open() && !message.is_open() && !encrypted.is_open())
    {
        return 1;   //Run as administrator
    }

    //read the whole message and store it in a string
```

```
    while(message >> noskipws >> c){
        m += c;
    }
    m+='\0';
    message.close();

    //deal with header
    target.read(t,40);
    encrypted.write(t,40);
    target.read(t,4);
    encrypted.write(t,4);
    int size_target,k(0),size_message(m.size());


    //Start encoding by inserting 1 byte of message in 8 bytes(LSB(s)) of targ
et
    //and writing the resultant byte on encrypted file.
    //Assumption: Target size is big enough to insert every message byte.
    while(!target.eof()) {
        c=m[k++];                           //1 byte of message stored in char
        for(int i(0); i<8; i++){            //read 8 bytes of target for 1 byt
e of message
            target.read(t,SKIP);
            encrypted.write(t,SKIP);        //skip k bytes for better data hid
ing and for no loss in audio quality
            target.read(t,1);
            t[0] = inject_bit(t,c,7-
i);     //returns an edited byte and stores it
            encrypted.write(t,1);
        }

        if(k>=size_message)
            break;
    }

    if(target.eof() && k < m.size()){
        remove(filename.c_str());
        return 2;       //File size error
    }

    //store the remaining file
    while(!target.eof()) {
        target.read(t,1);
        encrypted.write(t,1);
    }
    target.close();
    encrypted.close();
    return 0;
}
```

int *encrypt_bmp*::cipher(*string filename*="copy")

```
{
    bytecimg target(file_url.c_str());
    ifstream m(msg_url, ios::in);
    if(!m.is_open()){
        return 1; // File open error
    }
    bytecimg::iterator it = target.begin();
    string msg;
    char c;
    while(m>>noskipws>>c){
        msg+=c;
    }
    msg+=(unsigned char)26;
    int k(0);
    while(1){
        if(k>=msg.size()-1){
            break;
        }
        c = msg[k++];
        for(int i(0); i<8; i++){
            if(c>>i & 1){
                *it = *it | 1;
            }
            else{
                *it = *it & ~1;
            }
            it++;
        }
        if(it == target.end()){
            // File size error
            return 2;
        }
    }
    target.save(filename.c_str());
    return 0;
}
```

void *decrypt_wav*::decipher()

```
{
    ifstream encrypted(encrypted_file, ios::binary|ios::in);

    //skip the header
    encrypted.seekg(44);

    cout<<"\nDecrypting...";
    char c(0),temp[6];
    while(1) {
        c=0;
        //cout<<"Here";
        for(int i(0); i<8; i++) {
            //cout<<encrypted.tellg()<<endl;
            encrypted.read(temp,SKIP);          //skip 'K' just like before
            encrypted.read(temp,1);
            c = make_byte(temp[0], c, i);
        }
        if(encrypted.eof() || c=='\0')
            break;
        message += c;

    }
    encrypted.close();
    // cout<<"\nDone.";
}
```

void *decrypt_bmp*::decipher()

```
{
    bytecimg encoded(encrypted_file.c_str());
    unsigned char c(0);
    int bit(0);
    for (bytecimg::iterator it = encoded.begin(); it != encoded.end() && c!=(u
nsigned char)26; ++bit, ++it) {
        c |= (*it & 1) << bit;
            if (bit == 7) {
                message += c;
                bit = -1;
                c = 0;
            }
    }
}
```

Hash function used in Caesar's encrypt function:

```
        if (isupper(word[i]))
            result += char(int(word[i] + shift%26 - 65)%26 + 65);
        else
            result += char(int(word[i] + shift%26 - 97)%26 + 97);
```

And Caesar's decrypt function:

```
        if (isupper(word[i])) {
            if (word[i] - shift%26 - 65 < 0)
                result += char(26 - (shift%26 - (word[i] - 65)) + 65
);
            else
                result += char(word[i] - shift%26);
        }
        else {
            if (word[i] - shift%26 - 97 < 0)
                result += char(26 - (shift%26 - (word[i] - 97)) + 97
);
            else
                result += char(word[i] - shift%26);
        }
```

Hash function used in Vignere's encrypt function:

```
if (isupper(word[i]))
    enc_word += (char)(((int)word[i]-'A' + (int)temp_key[i]-
'A') % 26) + 'A';
else
    enc_word += (char)(((int)word[i]-'a' + (int)temp_key[i]-
'a') % 26) + 'a';
```

And in Vignere's decrypt function:

```
if (isupper(word[i]))
    dec_word += (char) ((((int)word[i]-'A' - ((int)temp_key[i]-
'A')) + 26) % 26) + 'A';
else
    dec_word += (char) ((((int)word[i]-'a' - ((int)temp_key[i]-
'a')) + 26) % 26) + 'a';
```

Function isSizeEnough() compares usableTargetSize() and messageSize() and returns if former is greater or equal to the latter and vice-versa.

Function usableTargetSize() iterates through the file and counts and returns the number of usable bytes which can be used to inject the data.

Function messageSize() simply reads the message file and counts number of bytes of the file and returns that in int.

Function encrypt::inject_bit() returns an edited byte of the audio or image file for insertion in the edited file.

Function decoded_message() simply returns message string.

Function write_message_to_file() takes file path and simply saves the message in that file.

Function decrypt::make_byte() collects bits, and adds them until 1 byte is reached and then it is returned and stored in the message file.

Function get_full_key() of Vignere class extends/trims the length of Key to match its length with the length of a particular word.

Function read_and_write() of Cipher Class reads each word from the file, encrypts/decrypts it using the encrypt(word: string) function or decrypt(word: string) and then writes on a new file.

Function int log_power() of rsa class works for both the encryption and decryption functions. It converts plain text to cypher text as plain text to power of some co prime number of the private key, and it same does with the encrypted text raised to coprime number of the public key.

Function rabin_miller() is a probabilistic function to find prime numbers quickly.

Function generate_prime() generate a prime number using rabin_miller() and log_power().

Function gcd() finds the gcd of two numbers.

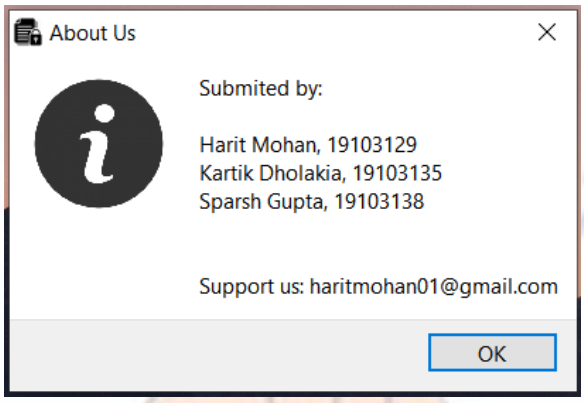
Function generate_coprime() finds the co prime number related to the public and the private key.

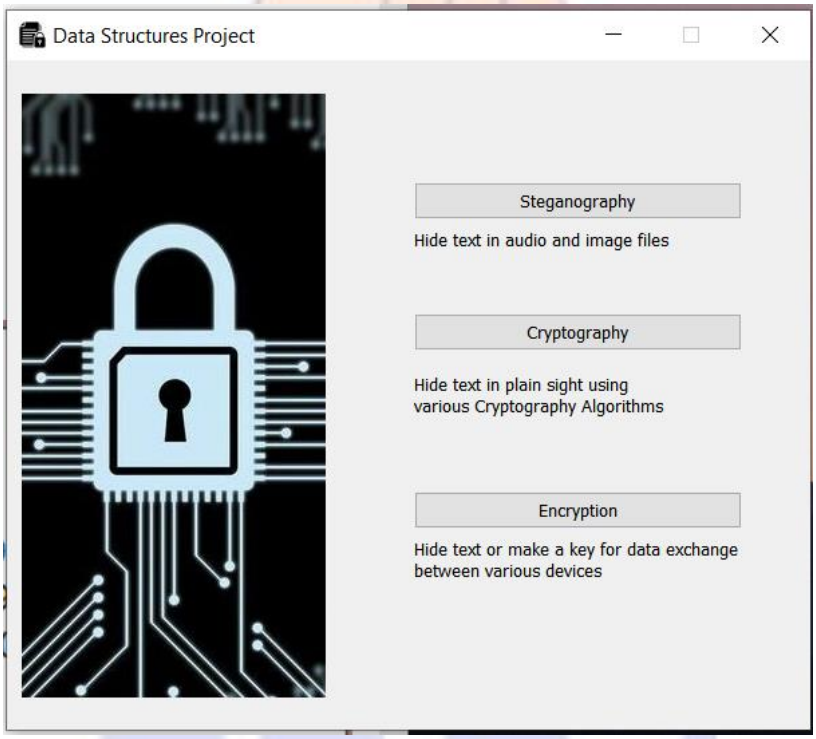
Function modular_inverse() finds 1 % phi(n).

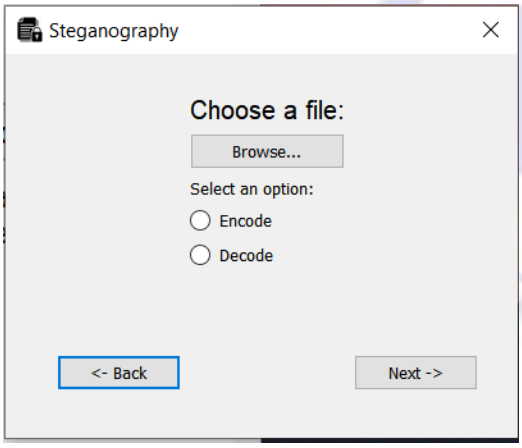
Function rsa_encrypt() encrypts the content of the file

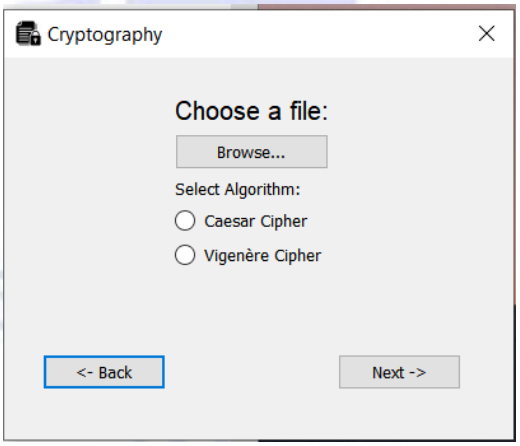
Function rsa_decrypt() decrypts the content of the encrypted file

Testing


About Us
Submited by:
Harit Mohan, 19103129
Kartik Dholakia, 19103135
Sparsh Gupta, 19103138
Support us: haritmohan01@gmail.com
OK

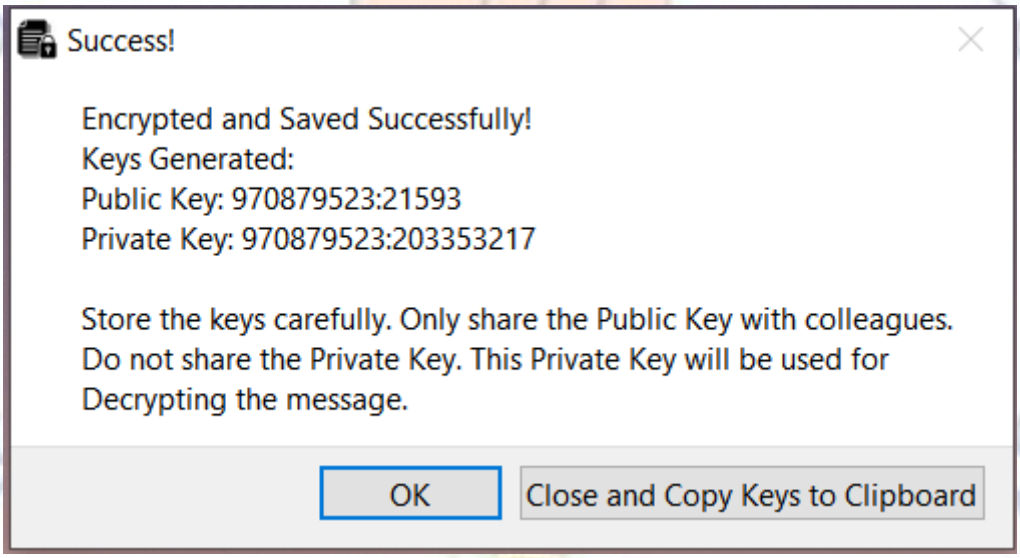

Data Structures Project
Steganography
Hide text in audio and image files
Cryptography
Hide text in plain sight using
various Cryptography Algorithms
Encryption
Hide text or make a key for data exchange
between various devices

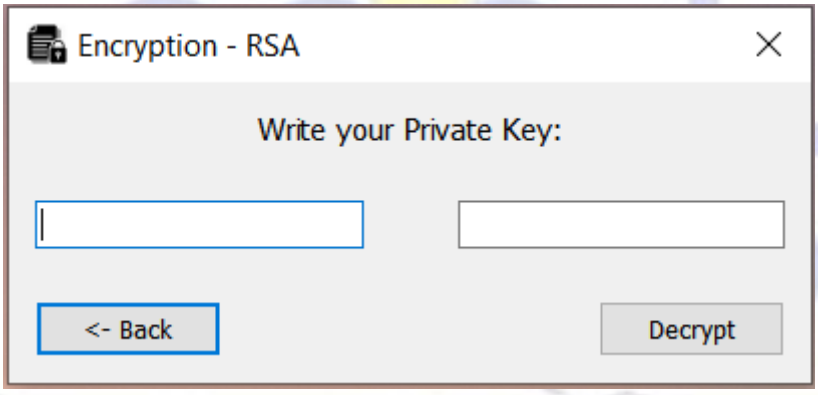

Steganography
Choose a file:
Browse...
Select an option:
Encode
Decode
<- Back
Next ->


Cryptography
Choose a file:
Browse...
Select Algorithm:
Caesar Cipher
Vigenère Cipher
<- Back
Next ->


Encryption - RSA
Choose a file:
Browse...
Select:
Encrypt
Decrypt
<- Back
Next ->


Encryption - RSA
Choose:
I Have A Public Key:
Generate A Key For Me
Back
Next ->


Success!
Encrypted and Saved Successfully!
Keys Generated:
Public Key: 970879523:21593
Private Key: 970879523:203353217
Store the keys carefully. Only share the Public Key with colleagues.
Do not share the Private Key. This Private Key will be used for
Decrypting the message.
OK
Close and Copy Keys to Clipboard


Encryption - RSA
Write your Private Key:
<- Back
Decrypt


JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY
jiit
विद्या तत्व ज्योतिसम: