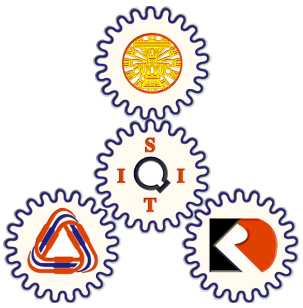


SCHOOL OF INFORMATION, COMPUTER, AND COMMUNICATION TECHNOLOGY

SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY

THAMMASAT UNIVERSITY

ITS335 IT Security

Assistant Prof. Dr. Nyuyen Duy Hung

Submitted by

Teerapat Satitporn 6122790114

Semester 2/2020

Contents

Overview

Cryptography is the conversion of regular messages into secret messages, where secret messages are not understood by anyone other than the interlocutor to whom we want to communicate.

Cryptography is achieved by mathematically using data and keys, which are random numbers, into encrypted data. We call this method “encryption” and call the encrypted data “ciphertext”. On the other hand, when we need to read data, it is mathematically processed again by the encrypted data and the key to getting the original data back. We call this method “decryption” and call the original data “plaintext”.

In this project, I will use Python to implement file encryption and decryption. I have applied the knowledge I learned in the IT Security course such as the RSA algorithm, encryption algorithm, and various cipher techniques to make this project successful.

Design

Caesar Cipher

Caesar Cipher or Shift Cipher is one of the simplest encryption techniques. This cipher will shift plaintext letters K positions to the right. In this project, I will not use mod 26 to give the coding a chance to get more characters besides the A-Z alphabets. The implementation of the original caesar cipher and the caesar cipher used in this project is shown below.

Original Caesar Cipher
Plaintext P = [p1, p2, . . . , pn] where pi ∈ [0, . . . , 25]
Ciphertext C = [c1, c2, . . . , cn] where ci = (pi + K)mod 26

In Project Caesar Cipher
Plaintext P = [p1, p2, . . . , pn]
Ciphertext C = [c1, c2, . . . , cn] where ci = (pi + K)

Rail-Fence Cipher

Rail-Fence Cipher is a form of classical transposition cipher. This cipher will write plaintext letters in diagonals over K rows, and ciphertext will obtain the plaintext letters by reading the letters row-by-row. The implementation of the rail-fence cipher is shown below.

Operation of Rail-Fence Cipher
If K = 3
Plaintext P = [p1, p2, . . . , pn]

p1	p4	p7	...		
	p2	p5	p8	...	
		p3	p6	p9	...

Ciphertext C = [c1, c4, c7, …, c2, c5, c8, …, c3, c6, c9, …, cn]

Long-Code Cipher

Long-Code Cipher is a cipher that I have personally invented. I created this cipher to make the ciphertext have a longer length than plaintext. This algorithm will change the plaintext letter to two ciphertext letters. As the result, The length of the ciphertext will be 2x longer than the plaintext. The implementation of the long-code cipher is shown below.

Operation of Long-Code Cipher

Define R as a random number between (1, pj)
Define j where j = floor(i/2) if i mod 2 = 1, j = i/2 if i mod 2 = 0
Plaintext P = [p1, p2, . . . , pn]
Ciphertext C = [c1, c2, . . . , cn] where ci = R if i mod 2 =1,
where ci = pj - R if i mod 2 = 0

RSA Algorithm

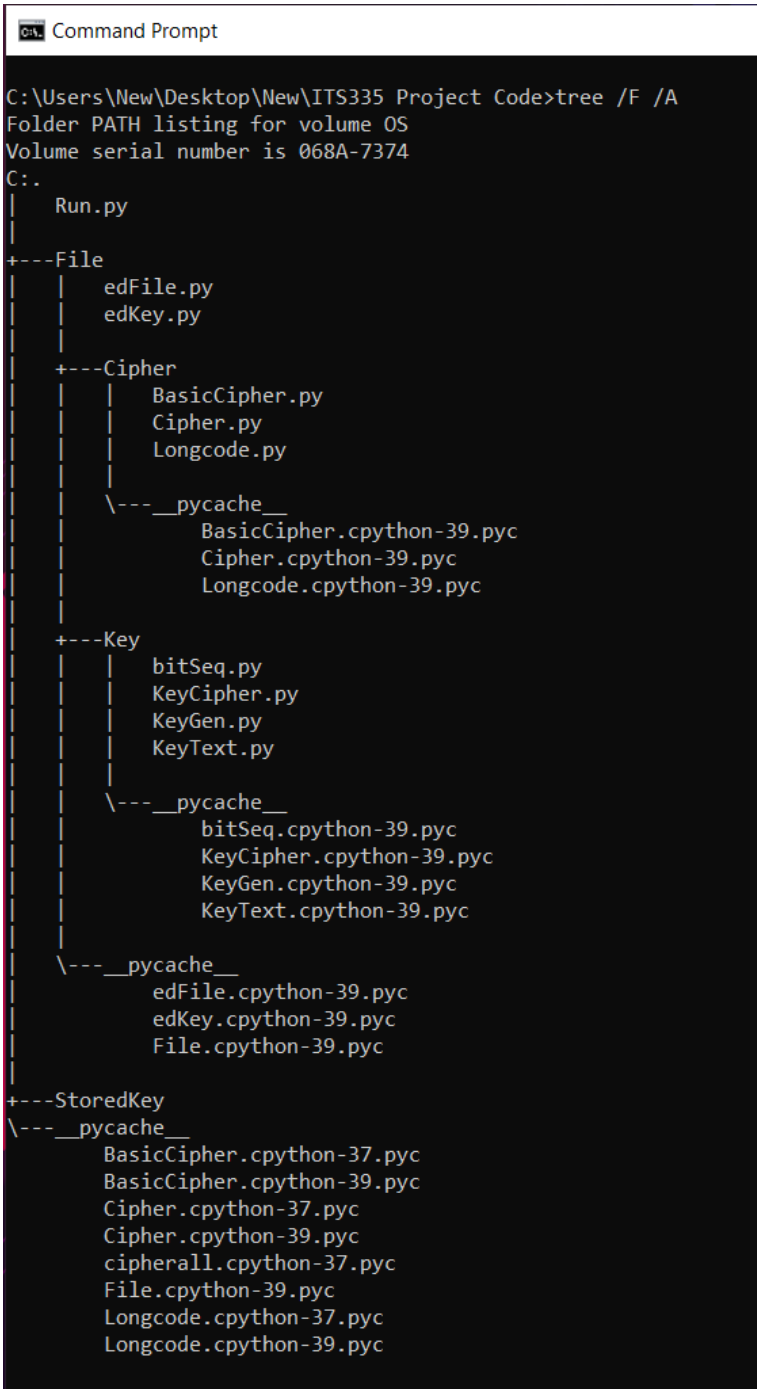
RSA or Rivest–Shamir–Adleman is an asymmetric cryptographic algorithm that is widely used for secure data transmission. The name RSA comes from the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who invented and publicly described the algorithm in 1977. RSA is an algorithm to make the key secure by generating the public key and private key. The implementation of the RSA algorithm is shown below.

Operation of RSA Algorithm

- Define prime number p and q
- n =p*q
- phi(n) = (p-1)*(q-1)
- Define number e which e < n and gcd(e,n) = 1
- Define number d which e*d mod phi(n) =1
- Public Key = (e,n), Private Key = (d,n)
- Let m be the value obtained from the hash function
- Encryption: c = (m^e) mod n
- Decryption: m = (c^d) mod n

Implementation

Structure



Code

There are 10 Python code files used to encrypt data in this project.

1. BasicCipher.py

BasicCipher.py contains the basic cipher techniques that I study in the class including Caesar cipher and rail-fence cipher. This file contains a code called "BasicCipher" to combine Caesar cipher and rail-fence cipher into a single command.

```
import math

def en_Caesar(text, key):
    message = ""
    i = 0
    for i in range(len(text)):
        num = ord(text[i])
        num += key
        message += chr(num)
        i += 1
    return message

def de_Caesar(text, key):
    message = ""
    i = 0
    for i in range(len(text)):
        num = ord(text[i])
        num -= key
        message += chr(num)
        i += 1
    return message

def en_Railfence(text, key):
    message = ""
    i = 0
    for i in range(key):
        j = 0
        for j in range(len(text)):
            if j%key == i:
                message += text[j]
            j += 1
        i += 1
    return message
```

```
def de_Railfence(text, key):
    message = ""
    i = 0
    col = math.ceil(len(text)/key)
    for i in range(col):
        j = 0
        row = key
        count = 0
        pos = i
        if len(text)%key != 0:
            if i == col-1:
                row = len(text)%key
            for j in range(row):
                message += text[pos]
                if count < len(text)%key:
                    pos += col
                    count += 1
                else:
                    pos += col-1
                j += 1
        else:
            for j in range(row):
                message += text[pos]
                pos += col
                j += 1
        i += 1
    return message

def en_BasicCipher(text, key1, key2):
    return en_Railfence(en_Caesar(text,key1),key2)

def de_BasicCipher(text, key1, key2):
    return de_Caesar(de_Railfence(text,key2),key1)
```

2. Longcode.py

Longcode.py contains the long-code cipher and repeat-long-code cipher. The repeat-long-code cipher is the algorithm used to repeat the long-code cipher according to the number received from the key. therefore, the length of the ciphertext will be 2^n longer than plaintext when we use the repeat-long-code cipher.

```
import random

def en_Longcode(text):
    message = ""
    for i in range(len(text)):
        r = random.randint(0, ord(text[i]))
        a = chr(ord(text[i]) - r)
        b = chr(r)
        message += a+b
    return message

def de_Longcode(text):
    message = ""
    i = 0
    while i < len(text):
        a = ord(text[i])
        b = ord(text[i+1])
        message += chr(a+b)
        i += 2
    return message

def en_RepeatLongcode(text, key):
    message = text
    for i in range(key):
        message = en_Longcode(message)
    return message

def de_RepeatLongcode(text, key):
    message = text
    for i in range(key):
        message = de_Longcode(message)
    return message
```

3. Cipher.py

Cipher.py contains all cipher techniques in this project into one command. Keys that are random numbers from 100-400 are divided into 3 sub-keys.

```
from BasicCipher import en_BasicCipher, de_BasicCipher
from Longcode import en_RepeatLongcode, de_RepeatLongcode

def en_Cipher(text, key):
    key = str(key)
    key1 = int(key[0]) + int(key[1]) + int(key[2])
    key2 = (int(key) % 8) + 2
    key3 = (int(key[0]) % 2) + 2
    return en_RepeatLongcode(en_BasicCipher(text, key1, key2), key3)

def de_Cipher(text, key):
    key = str(key)
    key1 = int(key[0]) + int(key[1]) + int(key[2])
    key2 = (int(key) % 8) + 2
    key3 = (int(key[0]) % 2) + 2
    return de_BasicCipher(de_RepeatLongcode(text, key3), key1, key2)
```

4. KeyGen.py

KeyGen.py contains the algorithms to generate public keys and private keys to make the RSA encryption. In PUKeyGen, I write the code to make a list of all possible values of e and randomly choose one of all values in the list.

```
from math import *
import random

def primeInRange(x, y):
    prime_list = []
    for n in range(x, y):
        isPrime = True

        for num in range(2, n):
            if n % num == 0:
                isPrime = False

        if isPrime:
            prime_list.append(n)

    return prime_list

def PUKeyGen(key):
    prime = primeInRange(10,31)

    p = random.choice(prime)
    q = random.choice(prime)
    n = p*q

    while n < key:
        p = random.choice(prime)
        q = random.choice(prime)
        n = p*q

    phiOfn = (p-1)*(q-1)

    E = []

    for i in range(1,phiOfn):
        if gcd(i,phiOfn)==1:
            E.append(i)

    e = random.choice(E)

    PUkey = (e,n)

    return PUkey,phiOfn
```

```
def PRKey(PUkey,phiOfn):
    e,n = PUkey

    for i in range(1, phiOfn):
        if i*e%phiOfn==1:
            d = i
            break

    PRkey = (d,n)

    return PRkey
```

5. bitSeq.py

bitSeq.py contains all RSA bit sequences, padding, and block algorithms to make the RSA encryption and decryption algorithms.

```
from math import *

def bitSeq2PlainBlocks(bitSeq,plainBlockSize):
    bitSeq = bitSeq + "1"
    if len(bitSeq)%plainBlockSize != 0:
        bitSeq = bitSeq + "0" * (plainBlockSize - len(bitSeq)%plainBlockSize)
    plainBlocks = []
    noOfblocks = int(len(bitSeq)/plainBlockSize)
    for i in range(0,noOfblocks):
        plainBlocks.append(bitSeq[i*plainBlockSize:(i+1)*plainBlockSize])
    return plainBlocks

def removePadding(bitSeq):
    indexOfOne = len(bitSeq)-1
    while bitSeq[indexOfOne]=="0":
        indexOfOne = indexOfOne -1
    return bitSeq[0:indexOfOne]

def blocks2numberSeq(blocks):
    numSeq = []
    for b in blocks:
        numSeq.append(int(b,2))
    return numSeq

def numberSeq2Blocks(numSeq, bsize):
    blocks = []
    for num in numSeq:
        block = bin(num)
        block = block[2:]
        if len(block)<bsize:
            block = "0"*(bsize-len(block)) + block
        blocks.append(block)
    return blocks
```

```
def rsaEncrypt(key, plainBitSeq):
    (e,n) = key
    plainBlockSize = floor(log(n,2))
    cipherBlockSize =  plainBlockSize + 1
    plainBlocks = bitSeq2PlainBlocks(plainBitSeq,plainBlockSize)
    plainNumSeq = blocks2numberSeq(plainBlocks)

    cipherNumSeq = []
    for plainNum in plainNumSeq:
        cipherNum = plainNum**e % n
        cipherNumSeq.append(cipherNum)
    cipherBlocks = numberSeq2Blocks(cipherNumSeq,cipherBlockSize)
    cipherBitSeq = ""
    for b in cipherBlocks:
        cipherBitSeq = cipherBitSeq + b
    return cipherBitSeq

def rsaDecrypt(key, cipherBitSeq):
    (d,n) = key
    plainBlockSize = floor(log(n,2))
    cipherBlockSize =  plainBlockSize + 1
    cipherBlocks = []
    numOfCipherBlocks = floor(len(cipherBitSeq)/cipherBlockSize)
    for i in range(0,numOfCipherBlocks):
        cipherBlocks.append(cipherBitSeq[i*cipherBlockSize: (i+1)*cipherBlockSize])
        cipherNumSeq = blocks2numberSeq(cipherBlocks)

    plainNumSeq = []
    for cipherNum in cipherNumSeq:
        plainNum = cipherNum**d % n
        plainNumSeq.append(plainNum)
    plainBlocks = numberSeq2Blocks(plainNumSeq,plainBlockSize)
    plainBitSeq = ""
    for pb in plainBlocks:
        plainBitSeq = plainBitSeq + pb
    return removePadding(plainBitSeq)
```

6. KeyText.py

KeyText.py is a code that contains two algorithms named “EncryptKeyText” and “DecryptKeyText”. The EncryptKeyText is an algorithm to encrypt the key and make it to be a text that contains the public key p and n, phi(n), and the binary encrypted key. The DecryptKeyText is an algorithm to change the key text back to the real key and send it to the system.

```
from KeyGen import *
from bitSeq import *

def EncryptKeyText(key):
    keygen = PUKeyGen(key)
    PU = keygen[0]
    phiOfn = keygen[1]

    binkey = rsaEncrypt(PU, bin(key))

    keytext = 'TE' + str(PU[0]) + 'N' + str(PU[1]) + 'P' + str(phiOfn) + 'K' + binkey

    return keytext

def DecryptKeyText(keytext):
    e = ''
    n = ''
    phiOfn = ''
    key = ''

    count = 0

    for i in range(len(keytext)):
        if keytext[i].isnumeric():
            if count == 1:
                e += keytext[i]
            elif count == 2:
                n += keytext[i]
            elif count == 3:
                phiOfn += keytext[i]
            elif count == 4:
                key += keytext[i]
        else:
            if keytext[i] == 'E':
                count = 1
            elif keytext[i] == 'N':
                count = 2
            elif keytext[i] == 'P':
                count = 3
            elif keytext[i] == 'K':
                count = 4

    PR = PRKey((int(e), int(n)), int(phiOfn))
    realkey = int(rsaDecrypt(PR,key), 2)

    return realkey
```

7. KeyCipher.py

KeyCipher.py is a code that contains the commands to encrypt and decrypt key text to make the key text more complex when stored in the system. The cipher that I use to encrypt and decrypt the key text is also the same as the cipher that I use to encrypt and decrypt the data.

```
import math
import sys
sys.path.append("./../Cipher")

from BasicCipher import en_BasicCipher, de_BasicCipher
from Longcode import en_RepeatLongcode, de_RepeatLongcode

def en_KeyCipher(keytext, key):
    key = int(key)
    key1 = (key % 10) + 2
    key2 = (key % 8) + 2
    key3 = (key % 2) + 2
    return en_RepeatLongcode(en_BasicCipher(keytext, key1, key2), key3)

def de_KeyCipher(keytext, key):
    key = int(key)
    key1 = (key % 10) + 2
    key2 = (key % 8) + 2
    key3 = (key % 2) + 2
    return de_BasicCipher(de_RepeatLongcode(keytext, key3), key1, key2)
```

8. edFile.py

edFile.py contains the command that combines all data encryption and decryption algorithms into a single command name “Encrypt” and “Decrypt”. The “Encrypt” and “Decrypt” commands contain the code to make the xor between the byte array of data and the key mod with 256 to make the ciphertext more complicated.

```
import sys
sys.path.append("./Cipher")

from Cipher import *

def Encrypt(filename, key):
    file = open(filename, 'rb')
    data = file.read()
    file.close()

    k = key%256

    data = bytearray(data)
    for i, v in enumerate(data):
        data[i] = v^k

    data = data.hex()
    c_data = en_Cipher(data, key)

    c_data = c_data.encode()

    file = open(filename, 'wb')
    file.write(c_data)
    file.close()

def Decrypt(filename, key):
    file = open(filename, 'rb')
    data = file.read()
    file.close()

    data = data.decode()

    m_data = de_Cipher(data, key)

    k = key%256

    data = bytearray.fromhex(m_data)
    for i, v in enumerate(data):
        data[i] = v^k

    file = open(filename, 'wb')
    file.write(data)
    file.close()
```

9. edKey.py

edKey.py contains the command that combines all key encryption and decryption algorithms into a single command.

```
import sys
sys.path.append("./Cipher")
sys.path.append("./Key")

from KeyText import *
from KeyCipher import *

def EncryptKey(key, num):
    return en_KeyCipher(EncryptKeyText(key), num)

def DecryptKey(keytext, num):
    return DecryptKeyText(de_KeyCipher(keytext, num))
```

10. Run.py

Run.py is a code used to run the program for encrypting and decrypting the file. Run.py also has the code to store the encrypted key text in the system.

```
import random
import sys
sys.path.append("./File/Cipher")
sys.path.append("./File/Key")
sys.path.append("./File")
sys.path.append("./StoredKey")

from edFile import *
from edKey import *

loop = True

while loop:
    print('1.Encrypt 2.Decrypt Other-->Exit')
    menu = input('Type number to choose menu: ')

    if menu == '1':
        filename = input('Type the file name you want to encrypt: ')

        key = random.randint(100,400)
        keytext = EncryptKey(key, len(filename))

        Encrypt(filename,key)

        filekey = open('StoredKey/'+filename+'key.txt', 'wb')
        filekey.write(keytext.encode())
        filekey.close()

        print('Encryption Success')
        print()

    elif menu == '2':
        filename = input('Type the file name you want to decrypt: ')

        filekey = open('StoredKey/'+filename+'key.txt', 'rb')
        keytext = filekey.read()
        filekey.close()

        key = DecryptKey(keytext.decode(), len(filename))

        Decrypt(filename,key)

        print('Decryption Success')
        print()
```

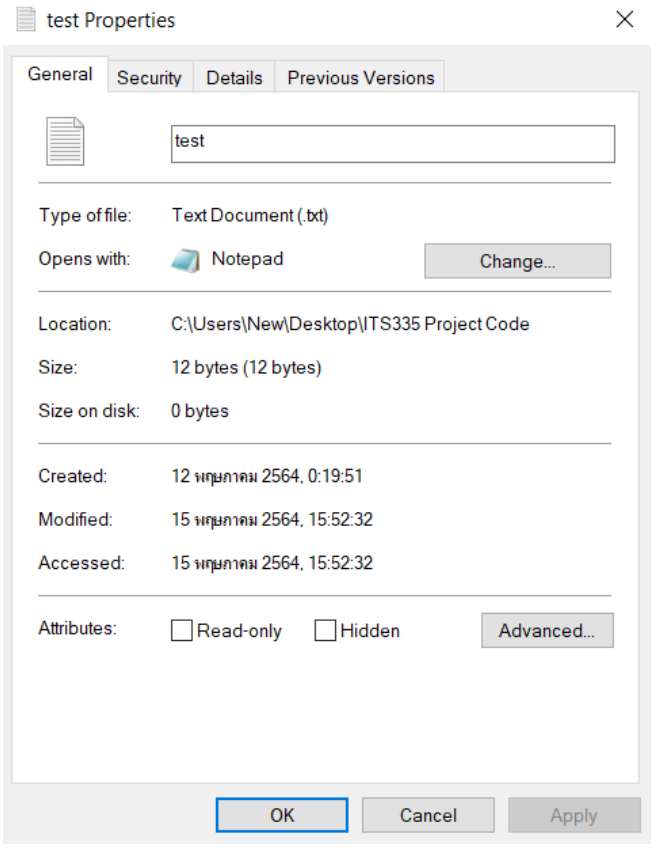
```
        filekey = open('StoredKey/'+filename+'key.txt', 'w')
        filekey.write('')
        filekey.close()

    else:
        print('Exit')
        loop = False
```

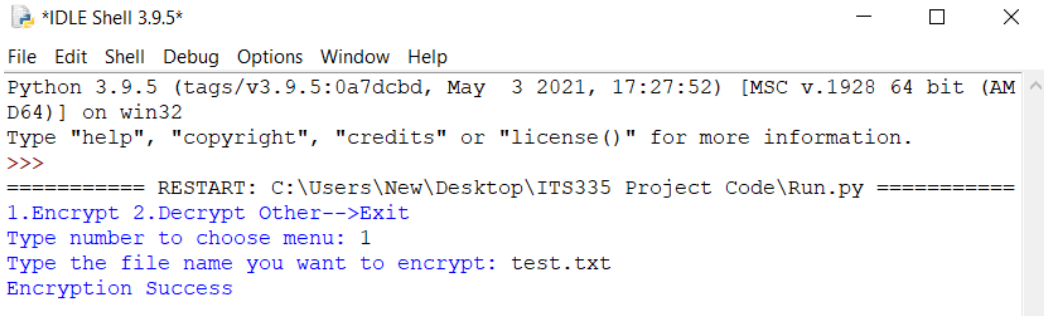
Result

Encrypted File

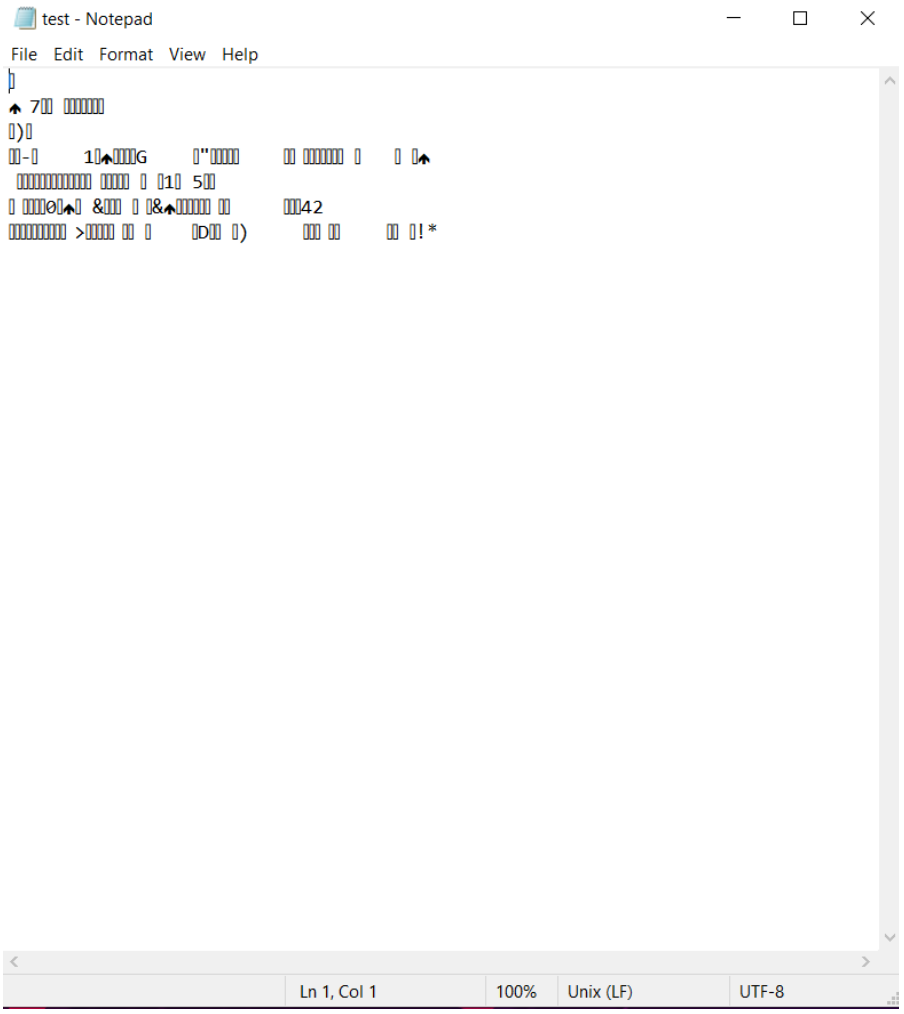
First, I start with trying to encrypt the text file named “test.txt”. The text file is the simplest file to use for testing the encryption and decryption system. In this file, I write the word “Hello World!” that is shown in the picture below.

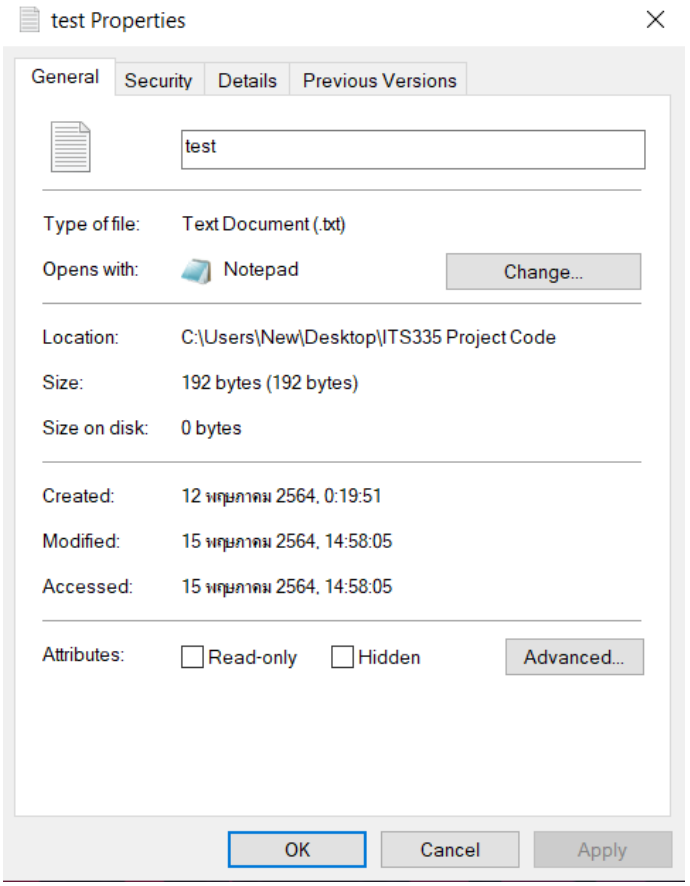


Then, I will encrypt the file by typing the command in run.py that is shown in the picture below.

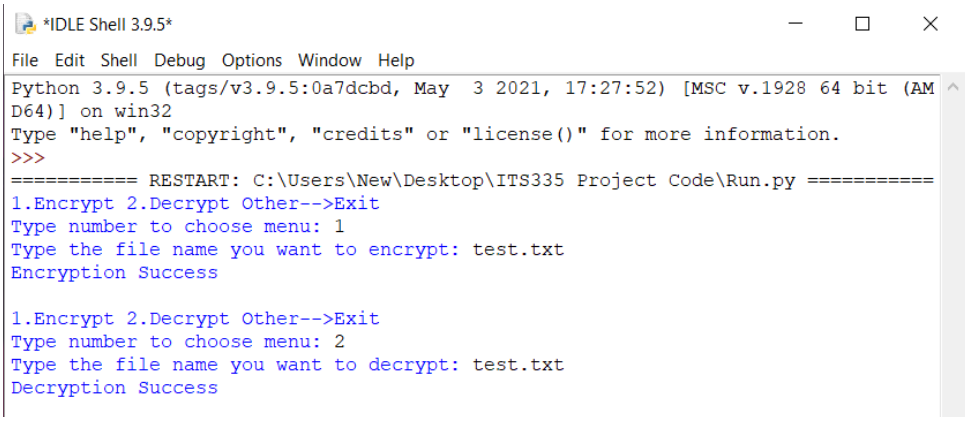


This file takes less than one second to encrypt because the file size is very small. Then, when we open the file “test.txt” again, we will see the word “Hello World!” was changed as the picture that is shown below.

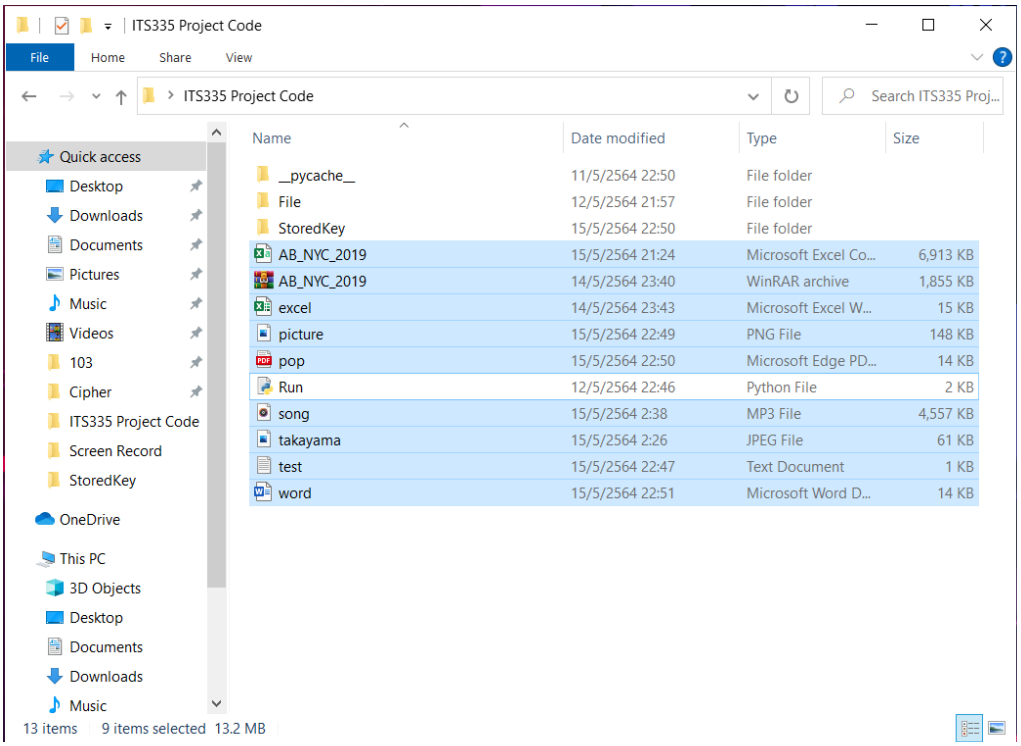

test - Notepad
File Edit Format View Help
Ln 1, Col 1
100%
Unix (LF)
UTF-8



From the picture above, we will see that the size of the encrypted file is larger than the original file. Then, I will decrypt the file back to the original by typing the command in run.py that is shown in the picture below. we will see the word “Hello World!” in the file “test.txt” again.

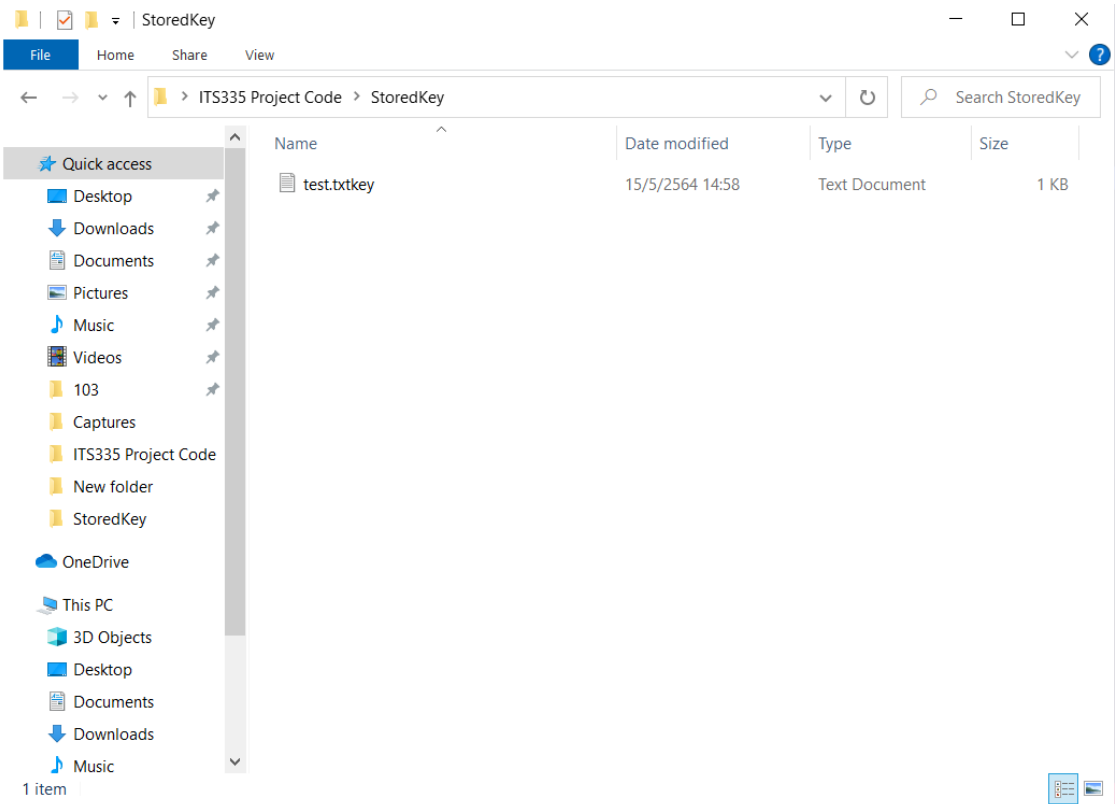


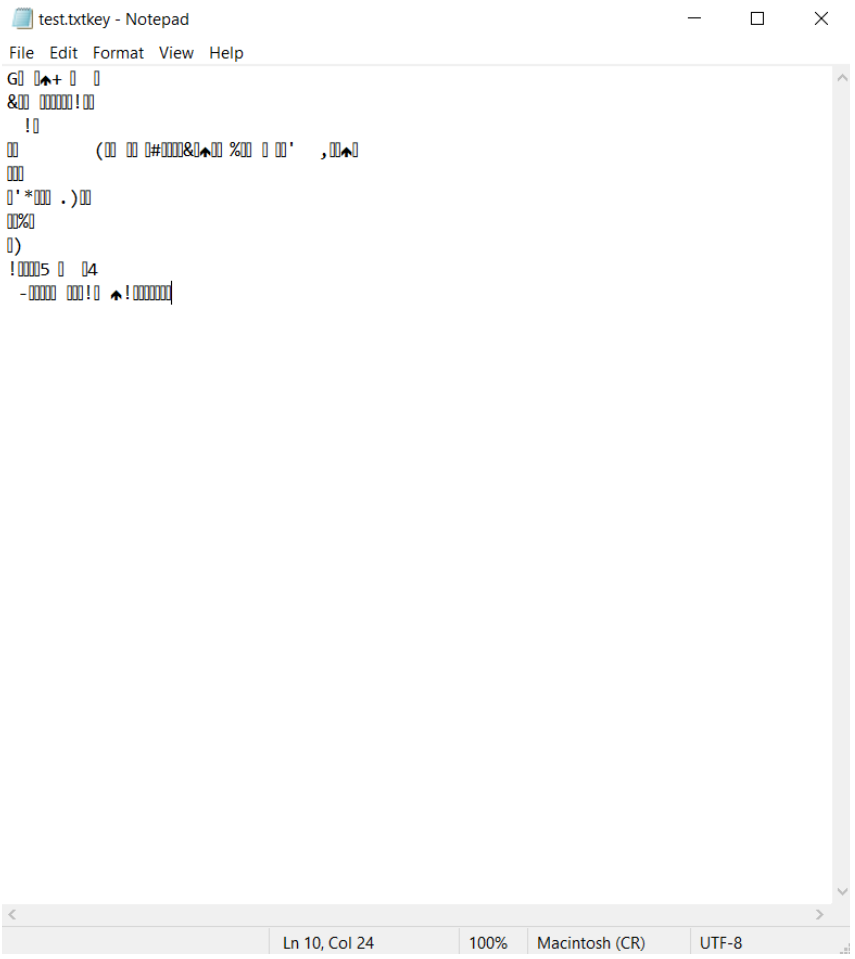
This program can also encrypt other types of files such as pdf, docx, mp3, jpeg, png etc. that are shown in the picture below.

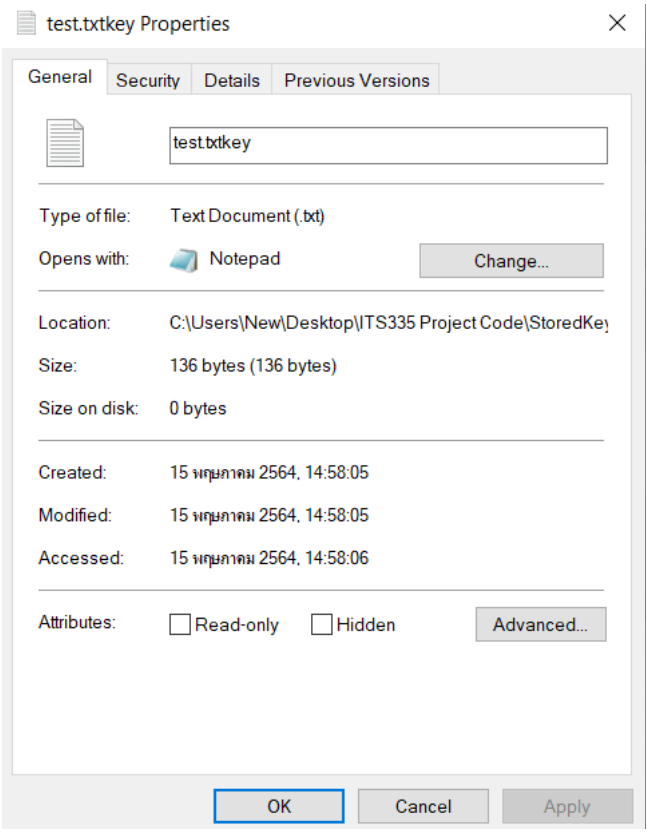


Stored Key

The encrypted key text will be written in a .txt file with the name of the encrypted file appended with the word “key” and the file will be stored in the folder name “StoredKey”. These three pictures show you the location, encrypted key text, and the size of the file named “test.txtkey.txt” which stored the key of the encrypted file named “test.txt”.




test.txtkey - Notepad
File Edit Format View Help
Ln 10, Col 24
100%
Macintosh (CR)
UTF-8


test.txtkey Properties
General
Security
Details
Previous Versions
test.txtkey
Type of file: Text Document (.txt)
Opens with: Notepad
Change...
Location: C:\Users\New\Desktop\ITS335 Project Code\StoredKey
Size: 136 bytes (136 bytes)
Size on disk: 0 bytes
Created: 15 พฤษภาคม 2564, 14:58:05
Modified: 15 พฤษภาคม 2564, 14:58:05
Accessed: 15 พฤษภาคม 2564, 14:58:06
Attributes: Read-only Hidden
Advanced...
OK
Cancel
Apply

Limitations

Security

From my research on the internet. The RSA algorithm that I use to encrypt the key can be attacked by using random number generators or guessing the d number.

Limit File Size

The largest file that I try to encrypt is “AB_NYC_2019.csv”. The size of this file is 6.75 MB. I am not guaranteed that my code can encrypt a file that has a size larger than 6.75 MB.

Computation Speed/Time

The speed of encryption and decryption in this project will depend on two reasons. The first reason is the file size. The computation speed will be slower when we encrypt a large file. For example, we will use less than one second to encrypt “test.txt” which has a size of only 12 bytes. On the other hand, we need to use more than 15 minutes to encrypt “AB_NYC_2019.csv” which has a size of up to 6.5 Mbs. The second reason is the key that is used for repeat-long-code cipher. Although I set the code to not repeat the long-code cipher more than four times, that is enough to increase the computation time of the file encryption.

Conclusion

In conclusion, I have made a tool that can encrypt and decrypt the files by using three existing algorithms and one personally created algorithm including Caesar cipher, rail-fence cipher, long-code cipher, and RSA algorithm to make the cryptography tool in this project. This tool can encrypt and decrypt various types of files including text, pdf, music, excels, words, etc. However, this tool cannot guarantee that it can encrypt a file that the size is larger than 6.75 MB.