**PRACTICAL REPORT**

**ON**

**PPSCSP204: CRYPTOGRAPHY AND CRYPTANALYSIS**

**SUBMITTED BY**

**ABHIJEET NAIKWADI**

**ROLL NO: 14**

**SUBMITTED TO**

**Ms. POOJA PANDEY**

**MSc. (COMPUTER SCIENCE) SEM - III  
2022 – 2023**



**CONDUCTED AT**

**CHIKITSAK SAMUHA’S**

**S. S. & L.S. PATKAR COLLEGE OF ARTS & SCIENCE**

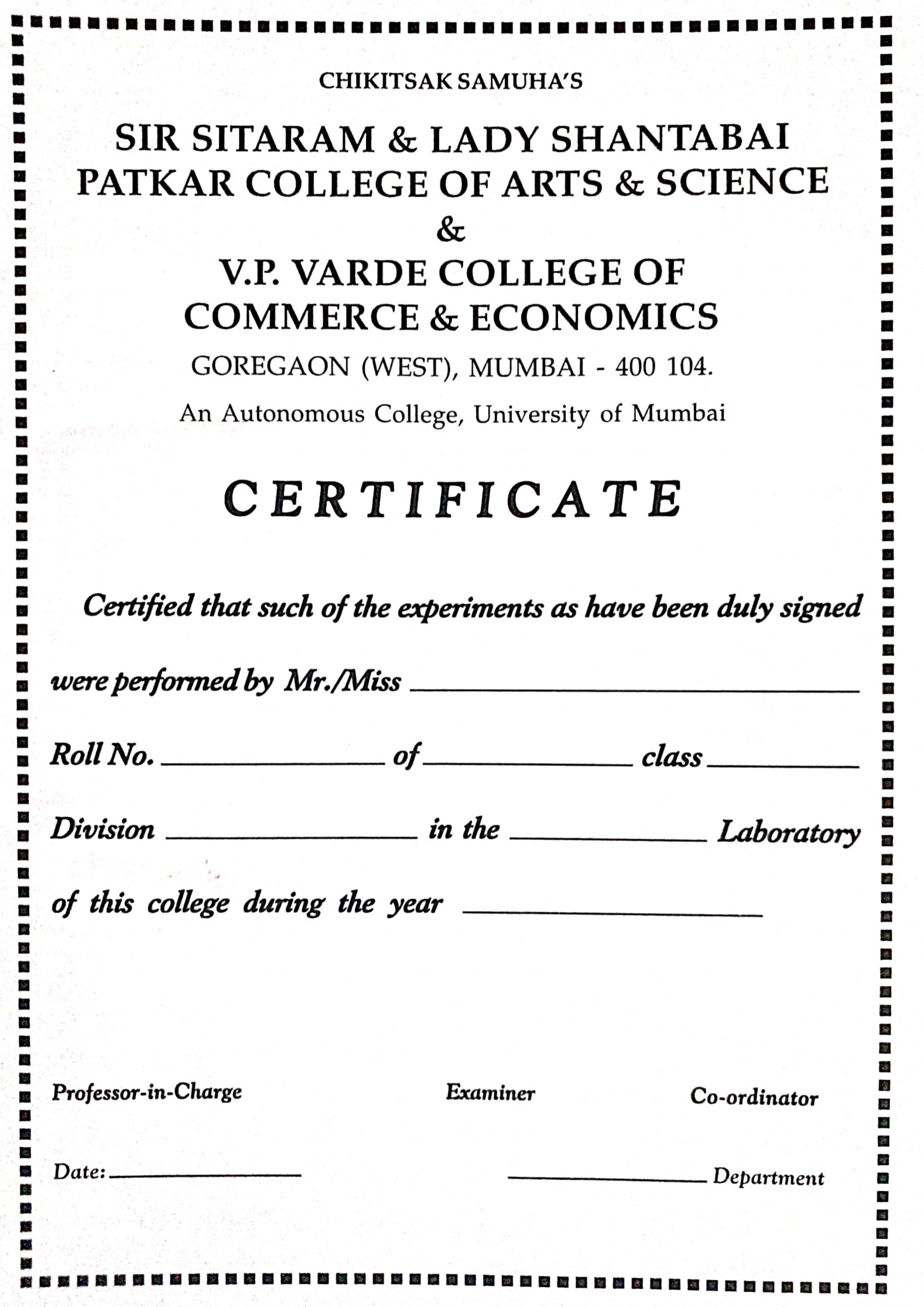
**AND**

**V. P. VARDE COLLEGE OF COMMERCE & ECONOMICS**

**An Autonomous college,**

**Affiliated to University of Mumbai**

**GOREGAON (W). MUMBAI -400062**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Practical No** | **Practical Aim** | **Date** | **Sign** |
| 1 | Program to implement password salting and hashing to create secure passwords. | 29.07.2022 |  |
| 2 | Program to implement various classical ciphers:   1. Substitution Cipher, 2. Vigenère Cipher, and 3. Affine cipher. | 08.08.2022 |  |
| 3 | Program to demonstrate cryptanalysis breaking Caesar Cipher. | 12.08.2022 |  |
| 4 | Program to implement AES algorithm for file encryption and decryption. | 29.08.2022 |  |
| 5 | Program to implement Steganography for hiding messages inside the image file. | 16.09.2022 |  |
| 6 | Program to implement HMAC for signing messages. | 16.09.2022 |  |
| 7 | Program to implement RSA encryption/decryption. | 19.09.2022 |  |
| 8 | Program to implement:   1. El-Gamal Cryptosystem 2. Elliptic Curve Cryptography | 23.09.2022 |  |

**Practical No: 1**

**Aim:** **Program to implement password salting and hashing to create secure passwords.**

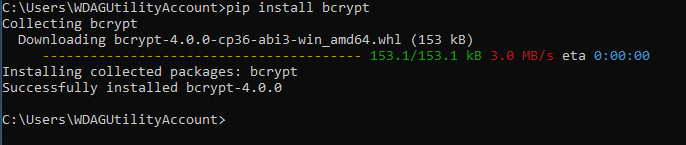
**Theory:**

**Hashing** is a cryptographic technique that transforms any form of data into a special text string. For any given input, there is a deterministic output. When you put a plaintext into a hashing algorithm in simpler terms, you get the same outcome. Suppose you change anything about the input or the plaintext to the hashing algorithm. The hashing output also becomes different.

Hashing is mainly used for authentication purposes. Salting makes password hashing more secure. Salting is an extra action during hashing. If two clients have the same password, they will also have the same password hashes. A salt, which is a random series of characters, is an extra input to the password before hashing. This makes an alternate hash result for the two passwords. Salting makes it difficult to use lookup tables and rainbow tables to crack a hash.

**Prerequisite:**

* pip install bcrypt

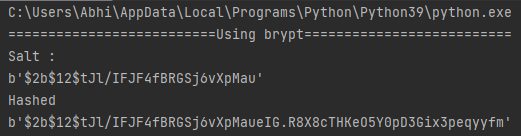
****

**Program:**

**Using Bcrypt**

import bcrypt  
# Declaring our password  
password = b'14-Abhijeet N'  
# Adding the salt to password  
salt = bcrypt.gensalt()  
# Hashing the password  
hashed = bcrypt.hashpw(password, salt)  
# printing the salt  
print("Salt :")  
print(salt)  
# printing the hashed  
print("Hashed")  
print(hashed)

**Output:**

****

**Using Hashlib:**

import hashlib

# Declaring Password

password = '14-Abhijeet N'

# adding salt as password

salt = "Big\_hash"

# Adding salt at the last of the password

dataBase\_password = password+salt

# Encoding the password

hashed = hashlib.md5(dataBase\_password.encode())

# Printing the Hash

print(hashed.hexdigest())

**Output:**



**Aim:** **Program to implement various classical ciphers:**

**a. Substitution Cipher,**

**b. Vigenère Cipher,**

**c. Affine cipher.**

**Theory:**

* **Substitution Cipher:**

In cryptography, a substitution cipher is a method of encryption by which units of plaintext are replaced with ciphertext according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth.

* **Vigenère cipher:**

The Vigenère cipher is an algorithm that is used to encrypting and decrypting the text. The Vigenère cipher is an algorithm of encrypting an alphabetic text that uses a series of interwoven Caesar ciphers. It is based on a keyword's letters. It is an example of a polyalphabetic substitution cipher.

* **Affine Cipher:**

The Affine cipher is a type of monoalphabetic substitution cipher, wherein each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which.

The whole process relies on working modulo m (the length of the alphabet used). In the affine cipher, the letters of an alphabet of size m are first mapped to the integers in the range 0 … m-1.

**Programs:**

* **Substitution Cipher:**

def encSubtitution(msg, key):

enc\_txt = ""

for i in msg:

enc\_num = ((ord(i) - 65) + key) % 26

enc\_txt += chr(enc\_num + 65)

return enc\_txt

def decSubstitution(encMsg, key):

dec\_txt = ""

for i in encMsg:

dec\_num = ((ord(i) - 65) - key) % 26

dec\_txt += chr(dec\_num + 65)

return dec\_txt

msg = " AbhijeetNaikwadi"

toUpper = msg.upper()

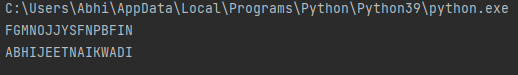
key = 5

enc = encSubtitution(toUpper, key)

print(enc)

print(decSubstitution(enc, key))

**Output:**

****

* **Vigenère Cipher:**

def encVigenere(msg, key):

msg = msg.upper()

key = key.upper()

enc\_txt = ""

if len(msg) == len(key):

for i in range(len(msg)):

enc\_num = ((ord(msg[i]) + ord(key[i])) % 26)

enc\_txt += chr(enc\_num + 65)

return enc\_txt

def decVigenere(encMsg, key):

encMsg=encMsg.upper()

key=key.upper()

if len(encMsg) == len(key):

dec\_txt = ""

for i in range(len(encMsg)):

dec\_num = ((ord(encMsg[i]) - ord(key[i]) + 26) % 26)

dec\_txt += chr(dec\_num + 65)

return dec\_txt

msg = " AbhijeetNaikwadi"

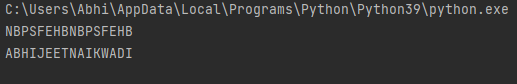
key = " NaikwadiAbhijeet"

enc\_msg=encVigenere(msg,key)

print(enc\_msg)

print(decVigenere(enc\_msg,key))

**Output:**

****

* **Affine cipher:**

def encAffine(msg, key\_a, key\_b):

enc\_str = ""

for i in toupper:

enc\_num = ((key\_a \* (ord(i) - 65) + key\_b) % 26)

enc\_str += chr(enc\_num + 65)

return enc\_str

def decAffine(encMsg, key\_a, ey\_b):

inv\_kA = pow(key\_a, -1, 26)

dec = ""

for i in encMsg:

dec\_txt = (inv\_kA \* ((ord(i) - 65) - key\_b) % 26)

dec += chr(dec\_txt + 65)

return dec

msg = "AbhijeetNaikwadi"

toupper = msg.upper()

key\_a = 3

key\_b = 1

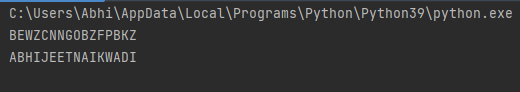
enc\_affine = encAffine(toupper, key\_a, key\_b)

print(enc\_affine)

dec\_affine = decAffine(enc\_affine, key\_a, key\_b)

print(dec\_affine)

**Output:**

****

**Practical No: 3**

**Aim:** **Program to demonstrate cryptanalysis breaking Caesar Cipher.**

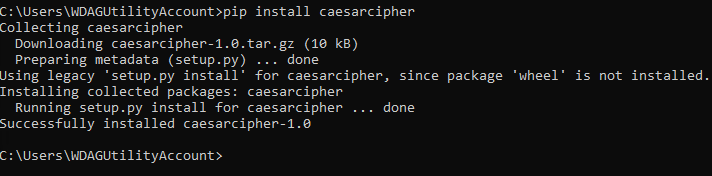
**Theory:**

**What is cryptanalysis?**

Cryptanalysis is the study of ciphertext, ciphers and cryptosystems with the aim of understanding how they work and finding and improving techniques for defeating or weakening them. For example, cryptanalysts seek to decrypt ciphertexts without knowledge of the plaintext source, encryption key or the algorithm used to encrypt it; cryptanalysts also target secure hashing, digital signatures and other cryptographic algorithms.

**Prerequisite:**

* pip install caesarcipher

****

**Program:**

from caesarcipher import CaesarCipher

cipher = CaesarCipher('Abhijeet N',offset=7)

print(cipher.encoded)

message = cipher.encoded.upper() #encrypted message

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

for key in range(len(LETTERS)):

translated = ''

for symbol in message:

if symbol in LETTERS:

num = LETTERS.find(symbol)

num = num - key

if num < 0:

num = num + len(LETTERS)

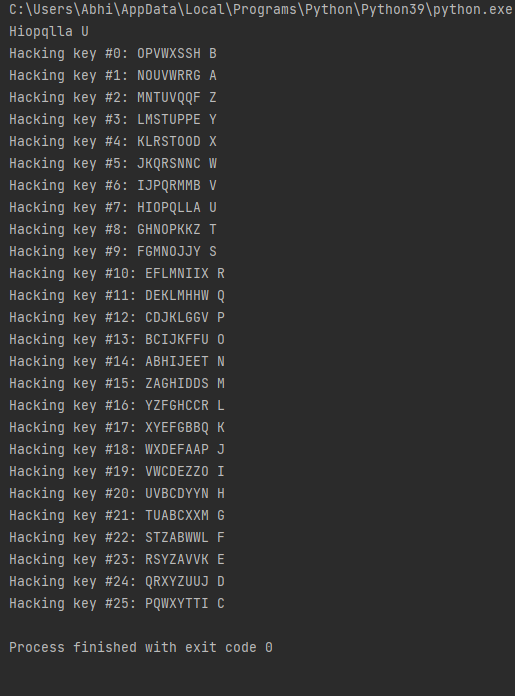
translated = translated + LETTERS[num]

else:

translated = translated + symbol

print('Hacking key #%s: %s' % (key, translated))

**Output:**

****

**Practical No: 4**

**Aim:** **Program to implement AES algorithm for file encryption and decryption.**

**Theory:**

**Advanced Encryption Standard (AES)** is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.

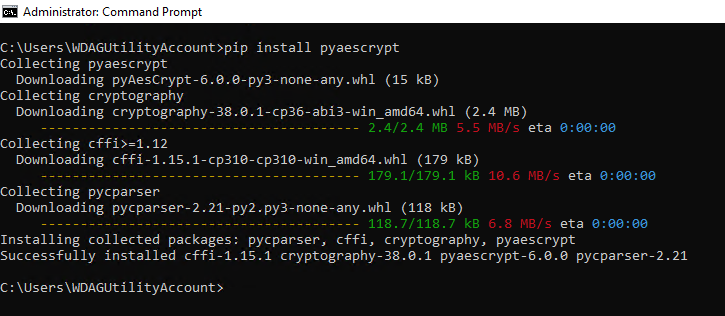
Points to remember

* AES is a block cipher.
* The key size can be 128/192/256 bits.
* Encrypts data in blocks of 128 bits each.

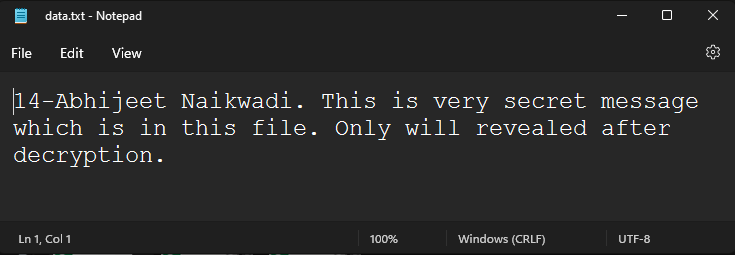
That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

**Prerequisite:**

* pip install pyAesCrypt

****

**Input file:**

****

**Program:**

import pyAesCrypt

# custom encryption/decryption buffer size (default is 64KB)

bufferSize = 128 \* 1024

password = "Pass@123"

# encrypt

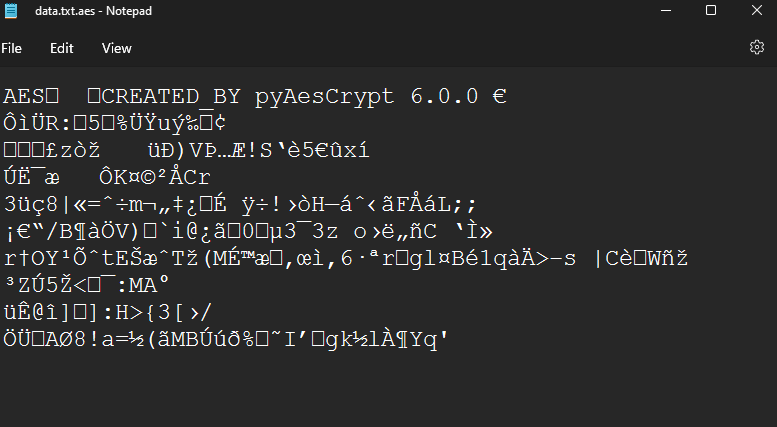
pyAesCrypt.encryptFile(".\Files\data.txt", "data.txt.aes", password, bufferSize)

# decrypt

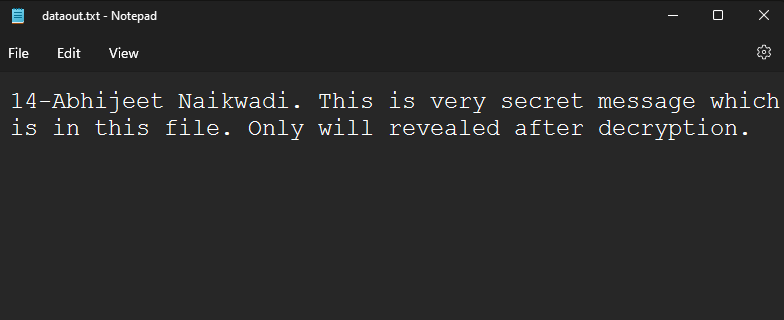
pyAesCrypt.decryptFile("data.txt.aes", "dataout.txt", password, bufferSize)

**Output:**

Data.txt.aes



Dataout.txt



**Practical No: 5**

**Aim:** **Program to implement Steganography for hiding messages inside the image file.**

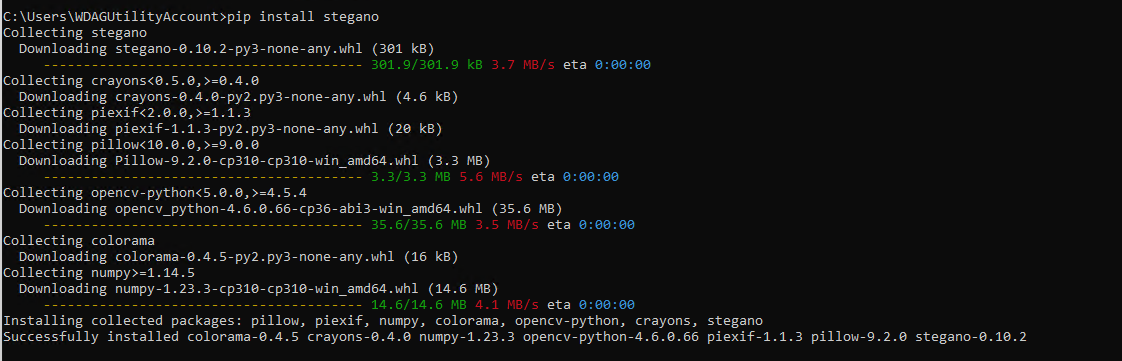
**Theory:**

**Steganography** is the technique of hiding secret data within an ordinary, non-secret, file or message in order to avoid detection; the secret data is then extracted at its destination. The use of steganography can be combined with encryption as an extra step for hiding or protecting data. The word steganography is derived from the Greek words steganos (meaning hidden or covered) and the Greek root graph (meaning to write).

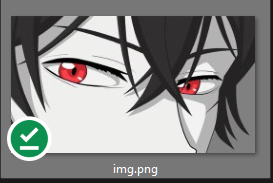
Steganography can be used to conceal almost any type of digital content, including text, image, video or audio content; the data to be hidden can be hidden inside almost any other type of digital content. The content to be concealed through steganography -- called hidden text -- is often encrypted before being incorporated into the innocuous-seeming cover text file or data stream. If not encrypted, the hidden text is commonly processed in some way in order to increase the difficulty of detecting the secret content.

**Prerequisite:**

* pip install stegano



**Img.png**



**Program:**

from stegano import lsb

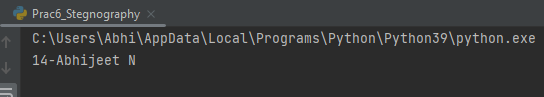
secret = lsb.hide("./Files/img.png", "luci is best")

secret.save("./Files/steg\_op.png")

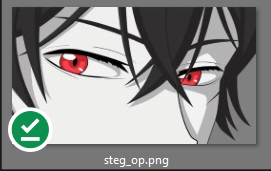
clear\_message = lsb.reveal("./Files/steg\_op.png")

print(clear\_message)

**Output:**



* **steg\_op.png**



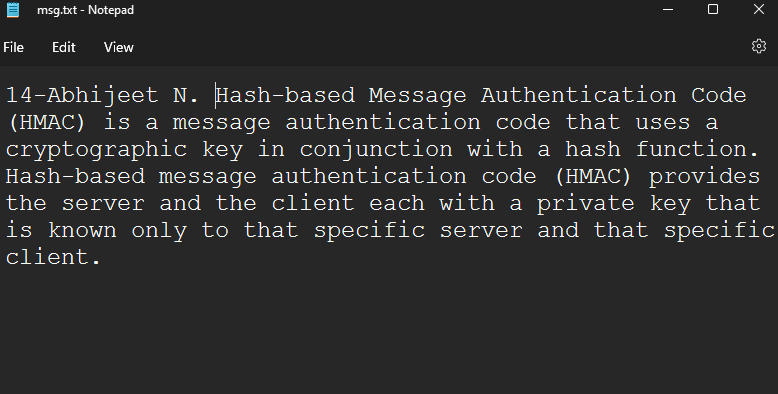
**Practical No: 6**

**Aim:** **Program to implement HMAC for signing messages.**

**Theory:**

**Hash-based Message Authentication Code (HMAC)** is a message authentication code that uses a cryptographic key in conjunction with a hash function. Hash-based message authentication code (HMAC) provides the server and the client each with a private key that is known only to that specific server and that specific client.

* **msg.txt:**



**Program:**

import hashlib

import hmac

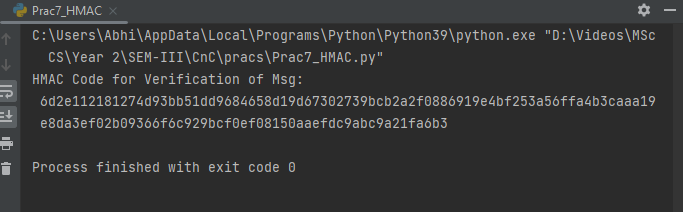
key = "Secret Key"

message = "14-Abhijeet N"

h = hmac.new(key.encode(), message.encode(), hashlib.sha512).hexdigest()

print("HMAC Code for Verification of Msg:",h)

**Output:**

****

**B)**

import hmac,hashlib

f = open('./Files/msg.txt', 'rb')

try:

while True:

block = f.read(1024)

if not block:

break

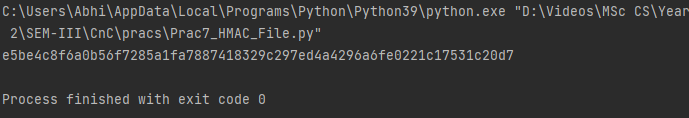
digest = hmac.new(b'14-Abhijeet', block, hashlib.sha256).hexdigest()

finally:

f.close()

print(digest)

**Output:**



**Practical No: 7**

**Aim:** **Program to implement RSA encryption/decryption.**

**Theory:**

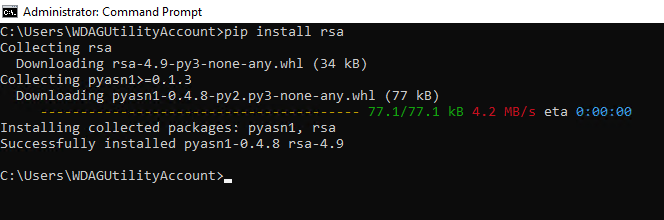
**RSA(Rivest, Shamir, Adleman)** algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client’s public key and sends the encrypted data.
3. Client receives this data and decrypts it.

**Prerequisite:**

* pip install rsa



**Programs:**

import rsa

publicKey, privateKey = rsa.newkeys(512)

message = "luci is best"

encMessage = rsa.encrypt(message.encode(),publicKey)

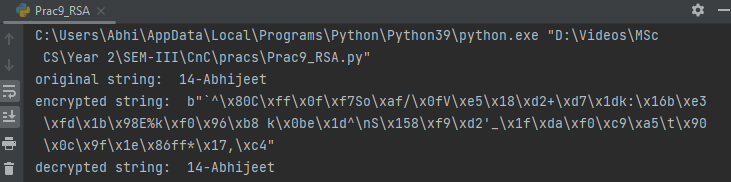
print("original string: ", message)

print("encrypted string: ", encMessage)

decMessage = rsa.decrypt(encMessage, privateKey).decode()

print("decrypted string: ", decMessage)

**output:**

****

**Practical No: 8**

**Aim:** **Program to implement:**

1. **El-Gamal Cryptosystem.**
2. **Elliptic Curve Cryptography.**

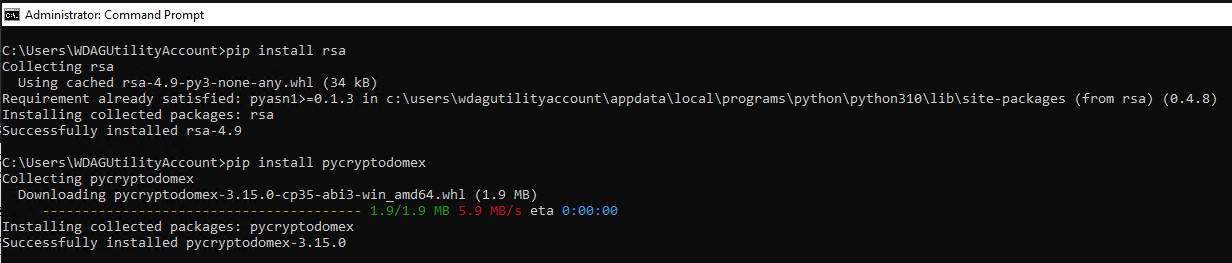
**Theory:**

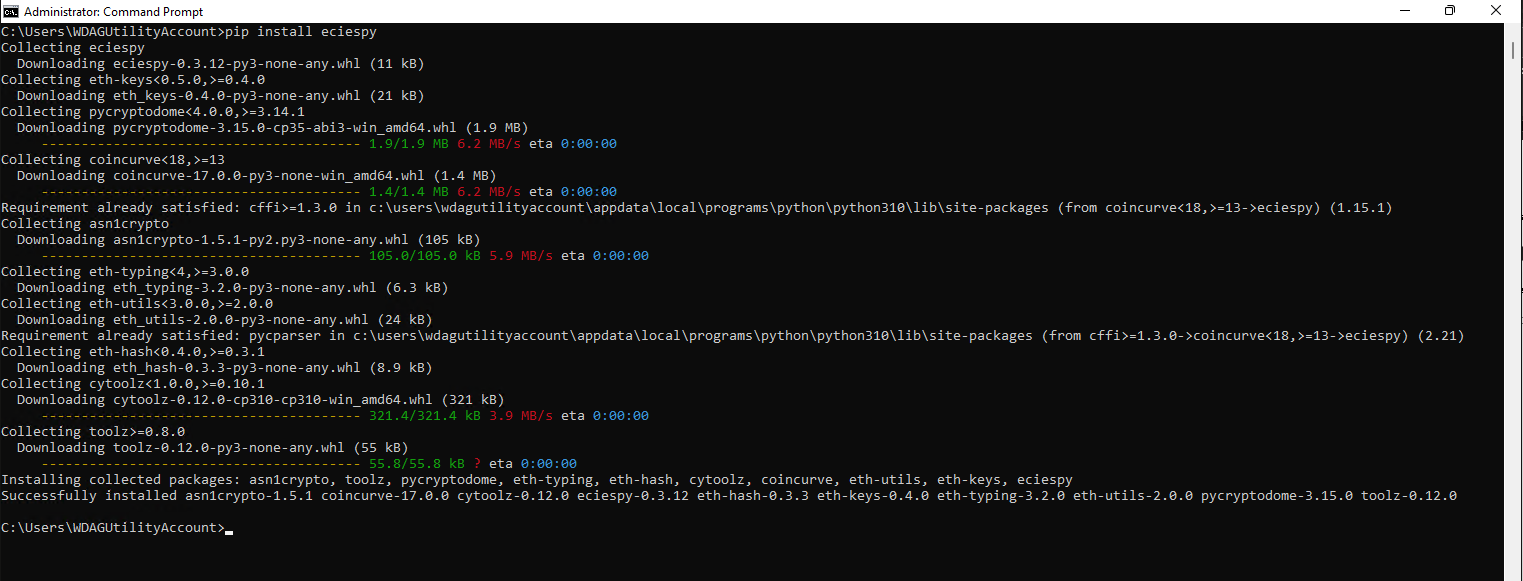
**ElGamal** is a public key encryption algorithm that was described by an Egyptian cryptographer Taher Elgamal in 1985. This encryption algorithm is used in many places. Many of us may have also used this encryption algorithm in GNU Privacy Guard or GPG.

**Elliptic curve cryptography (ECC)** is a modern type of public-key cryptography wherein the encryption key is made public, whereas the decryption key is kept private. This particular strategy uses the nature of elliptic curves to provide security for all manner of encrypted products.

**Prerequisite:**

* pip install elgamal
* pip install pycryptodomex
* pip install eciespy





**Program:**

from elgamal.elgamal import Elgamal

msg=b'14-Abhijeet N'

pb\_k,pv\_k=Elgamal.newkeys(16)

print(pb\_k)

print(pv\_k)

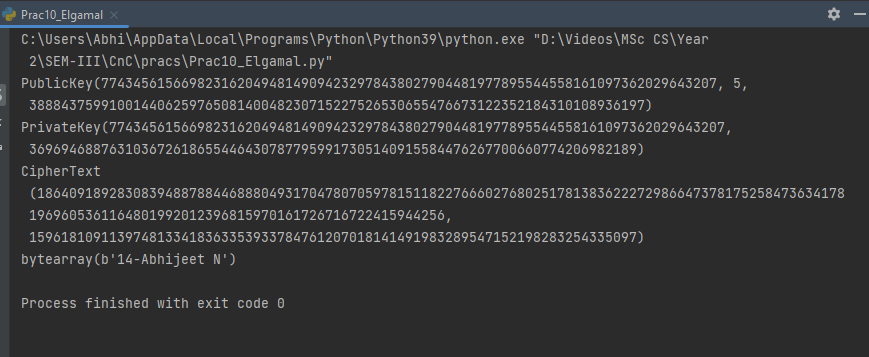
Enc\_msg=Elgamal.encrypt(msg,pb\_k)

print(Enc\_msg)

Dec\_msg=Elgamal.decrypt(Enc\_msg,pv\_k)

print(Dec\_msg)

**Output:**

****

**B) Elliptic Curve Cryptography**

from ecies.utils import generate\_eth\_key

from ecies import encrypt, decrypt

import binascii

privKey = generate\_eth\_key()

privKeyHex = privKey.to\_hex()

pubKeyHex = privKey.public\_key.to\_hex()

print("Encryption public key:", pubKeyHex)

print("Decryption private key:", privKeyHex)

plaintext = b'14-Abhijeet N'

print("Plaintext:", plaintext)

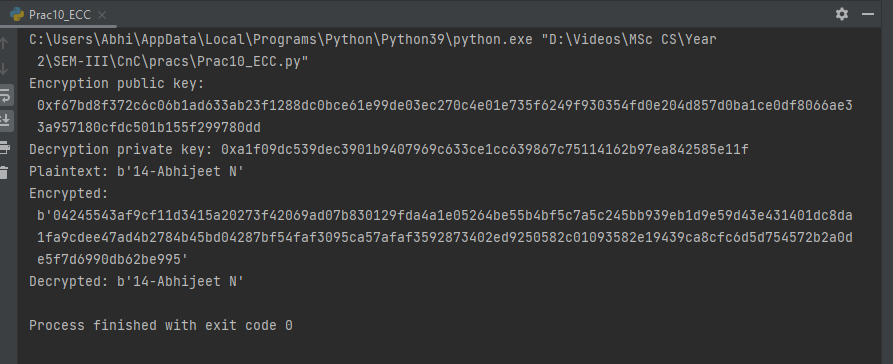
encrypted = encrypt(pubKeyHex, plaintext)

print("Encrypted:", binascii.hexlify(encrypted))

decrypted = decrypt(privKeyHex, encrypted)

print("Decrypted:", decrypted)

**Output:**

****