Practical Lab File

Name:	Shitab Tanzim
Enrol No.:	A91005218061
Dept:	B.Tech CSE
Batch:	2018-22
Semester:	7
Section:	A
Subject:	Applied Cryptography
Sub.Code:	CSE442
Notebook:	CryptographyFile.ipynb [link]

INDEX

Q. Implement Caesar Cipher encryption-decryption.

Code:

```
def encrypt(P,k):
   answer=""
   for p in P:
      if (p.isupper()):
         answer += chr( (ord(p)+k-65)%26 + 65 )
      else:
         answer += chr( (ord(p)+k-97)%26 + 97 )
   return answer

def decrypt(P,k):
   answer=""
   for p in P:
      if (p.isupper()):
         answer += chr( (ord(p)-k-65)%26 + 65 )
      else:
         answer += chr( (ord(p)-k-97)%26 + 97 )
   return answer

P = input("Enter Text: ")
k = int(input("Enter Key: "))
encrypted = encrypt(P,k)
decrypted = decrypt(encrypted,k)
print("Encoded Message: ", encrypted)
print("Decoded Message: ", decrypted)
```

Output:

```
Enter Text: EtTuBrute
Enter Key: 6
Encoded Message:  KzZaHxazk
Decoded Message:  EtTuBrute
```

Q. Multiplicative Cipher Encryption.

Code:

```
def encrypt(P,k):
   answer=""
   for p in P:
      if (p.isupper()):
         answer += chr( (ord(p)*k-65)%26 + 65 )
      else:
         answer += chr( (ord(p)*k-97)%26 + 97 )
   return answer

P = input("Enter Text: ")
n = int(input("Enter Key: "))
encrypted = encrypt(P,k)
print("Encoded Message: ", encrypted)
```

Output:

```
Enter Text: hdsvkk
Enter Key: 4
Encoded Message:  hvoxqq
```

Q. Euclidean Algorithm (Basic).

Code:
```
def gcd(a, b):
   if a == 0 :
      return b
   return gcd(b%a, a)

a, b = input("Enter a and b to demonstrate Euclidean Algo: ").split()
a, b = int(a), int(b)
print("GCD of {} and {} is {}".format(a,b,gcd(a,b)))
```

Output:

```
Enter a and b to demonstrate Euclidean Algo: 36 56
gcd of 36 and 56 is 4
```

Q. Euclidean Algorithm (Extended).

Code:
```
def gcdExtended(a, b):
   if a == 0 :
      return b, 0, 1
   gcd, x1, y1 = gcdExtended(b%a, a)
   x = y1 - (b//a) * x1
   y = x1
   return gcd, x, y

a, b = input("Enter a and b to demonstrate EXTENDED Euclidean Algo:
").split()
a, b = int(a), int(b)
g, x, y = gcdExtended(a, b)
print("GCD of {} and {} is {}".format(a,b,g))
```

Output:

```
Enter a and b to demonstrate EXTENDED Euclidean Algo: 5 35
GCD of 5 and 35 is 5
```

Q. Multiplicative inverse using extended Euclidean.

Code:

```
def multiplicative_inverse(A, M):

    gcd, x, y = gcdExtended(A, M)
    if x < 0:
        x += M
      return x

def gcdExtended(a, b):
     if a == 0 :
        return b, 0, 1

    gcd, x1, y1 = gcdExtended(b%a, a)

    x = y1 - (b//a) * x1
    y = x1

    return gcd, x, y

a, b = input("Enter a and b to demonstrate Multiplicative Inverse Using
Euclidean Algo: ").split()
a, b = int(a), int(b)

print("Multiplicative Inverse is {}".format(multiplicative_inverse(a, b)))
```

Output:

```
Enter a and b to demonstrate Multiplicative Inverse Using Euclidean Algo: 5 11
Multiplicative Inverse is 9
```

Q. Encrypt the text "The house is being sold tonight" Using Vignere Cipher with Key= “Dollar”.

Code:

```
def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) - len(key)):
            key.append(key[i % len(key)])
    return("".join(key))

def cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) + ord(key[i])) % 26
        x += ord('A')
        cipher_text.append(chr(x))
    return("".join(cipher_text))

def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("".join(orig_text))

string = "THE HOUSE IS BEING SOLD TONIGHT"
keyword = "DOLLAR"
key = generateKey(string, keyword)
cipher_text = cipherText(string,key)
print("Ciphertext :", cipher_text)
print("Original/Decrypted Text :", originalText(cipher_text, key))
```

Output:

```
Ciphertext : WVPEHFXGPEIJWPPTNXWGZWDKWCYTGYW
Original/Decrypted Text : THETHOUSETISTBEINGTSOLDTTONIGHT
```

Q. Encrypt the text "The house is being sold tonight" Using Affine Cipher key=(15,20)

<u>Code</u>:

```
def gcdExtended(a, b):
    if a == 0 :
        return b, 0, 1
    gcd, x1, y1 = gcdExtended(b%a, a)
    x = y1 - (b//a) * x1
    y = x1
    return gcd, x, y

def modinv(a, m):
    gcd, x, y = gcdExtended(a, m)
    if gcd != 1:
        return None
    else:
        return x % m

def affine_encrypt(text, key):
    return ''.join([chr(((key[0]*(ord(t)-ord('A'))+key[1])%26)+ord('A')) for t in
text.upper().replace(' ', '') ])

def affine_decrypt(cipher, key):
    return ''.join([chr(((modinv(key[0],26)*(ord(c)-ord('A')-key[1]))%
26)+ord('A')) for c in cipher ])


text = 'THE HOUSE IS BEING SOLD TONIGHT'
key = [15, 20]
affine_encrypted_text = affine_encrypt(text, key)
print('Encrypted Text: {}'.format( affine_encrypted_text ))
print('Decrypted Text: {}'.format( affine_decrypt(affine_encrypted_text,
key) ))
```

<u>Output</u>:

```
Encrypted Text: TVCVWIECKEJCKHGEWDNTWHKGVT
Decrypted Text: THEHOUSEISBEINGSOLDTONIGHT
```

Q. Encrypt text using PlayFair Cipher.

Code:

```
def convertPlainTextToDiagraphs (plainText):
    for s in range(0,len(plainText)+1,2):
        if s<len(plainText)-1:
            if plainText[s]==plainText[s+1]:
                plainText=plainText[:s+1]+'X'+plainText[s+1:]
    if len(plainText)%2 != 0:
        plainText = plainText[:]+'X'
    return plainText

def generateKeyMatrix (key):
    matrix_5x5 = [[0 for i in range (5)] for j in range(5)]
    simpleKeyArr = []
    for c in key:
        if c not in simpleKeyArr:
            if c == 'J':
                simpleKeyArr.append('I')
            else:
                simpleKeyArr.append(c)
    is_I_exist = "I" in simpleKeyArr
    for i in range(65,91):
        if chr(i) not in simpleKeyArr:
            if i==73 and not is_I_exist:
                simpleKeyArr.append("I")
                is_I_exist = True
            elif i==73 or i==74 and is_I_exist:
                pass
            else:
                simpleKeyArr.append(chr(i))
    index = 0
    for i in range(0,5):
        for j in range(0,5):
            matrix_5x5[i][j] = simpleKeyArr[index]
            index+=1
    return matrix_5x5
```

```
def indexLocator (char,cipherKeyMatrix):
    indexOfChar = []
    if char=="J":
        char = "I"
    for i,j in enumerate(cipherKeyMatrix):
        for k,l in enumerate(j):
            if char == l:
                indexOfChar.append(i)
                indexOfChar.append(k)
                return indexOfChar

def encryption (plainText,key):
    cipherText = []
    keyMatrix = generateKeyMatrix(key)
    i = 0
    while i < len(plainText):
        n1 = indexLocator(plainText[i],keyMatrix)
        n2 = indexLocator(plainText[i+1],keyMatrix)
        if n1[1] == n2[1]:
            i1 = (n1[0] + 1) % 5
            j1 = n1[1]
            i2 = (n2[0] + 1) % 5
            j2 = n2[1]
            cipherText.append(keyMatrix[i1][j1])
            cipherText.append(keyMatrix[i2][j2])
            cipherText.append(", ")
        elif n1[0]==n2[0]:
            i1= n1[0]
            j1= (n1[1] + 1) % 5
            i2= n2[0]
            j2= (n2[1] + 1) % 5
            cipherText.append(keyMatrix[i1][j1])
            cipherText.append(keyMatrix[i2][j2])
            cipherText.append(", ")
        else:
            i1 = n1[0]
            j1 = n1[1]
```

```
            i2 = n2[0]
            j2 = n2[1]
            cipherText.append(keyMatrix[i1][j2])
            cipherText.append(keyMatrix[i2][j1])
            cipherText.append(", ")
        i += 2
    return cipherText

key = input("Enter key: ").replace(" ","").upper()
plainText =input("Plain Text: ").replace(" ","").upper()
convertedPlainText = convertPlainTextToDiagraphs(plainText)
cipherText = " ".join(encryption(convertedPlainText,key))
print(cipherText)
```

<u>Output</u>:

```
Enter key: 5
Plain Text: Encrypt This
H K ,  B S ,  U T ,  S Y ,  S I ,  H T ,
```

Q. Encrypt text using Hill Cipher.

Code:

```
keyMatrix = [[0] * 3 for i in range(3)]
messageVector = [[0] for i in range(3)]
cipherMatrix = [[0] for i in range(3)]

def getKeyMatrix(key):
    k = 0
    for i in range(3):
        for j in range(3):
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1

def encrypt(messageVector):
    for i in range(3):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(3):
                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):
    getKeyMatrix(key)
    for i in range(3): messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector)
    CipherText = []
    for i in range(3):
        CipherText.append(chr(cipherMatrix[i][0] + 65))
    return "".join(CipherText)

message = input("Enter Message: ")
key = input("Enter key: ")
print("CipherText: ",HillCipher(message, key))
```

Output:

```
Enter Message: HELLOO
Enter key: HITHERSUP
CipherText:  ESH
```

Q. Diffie-Helman Exchange Algorithm.

Code:

```
from random import randint
 P = 23
G = 9

print('The Value of P is ',P)
print('The Value of G is ',G)

private_a = int(input("A, Choose private key: "))
print('The Private Key a for A is ',private_a)
gen_a = int(pow(G,private_a,P))

private_b = int(input("B, Choose private key: "))
print('The Private Key b for B is ',b)
gen_b = int(pow(G,private_b,P))

secret_a = int(pow(gen_b,private_a,P))
secret_b = int(pow(gen_a,private_b,P))

print('Secret key for the A is : ',secret_a)
print('Secret Key for the B is : ',secret_b)
```

Output:

```
The Value of P is  23
The Value of G is  9
A, Choose private key: 7
The Private Key a for A is  7
B, Choose private key: 9
The Private Key b for B is  9
Secret key for the A is :  13
Secret Key for the B is :  13
```

Q. Write a program to implement encryption and decryption of the message using DES.

Code:

```
from Crypto.Cipher import DES

def pad(text):
    n = len(text) % 8
    return text + (b' ' * n)


key = b'eight888'
text1 = b'This is the message I am encrypting!'

des = DES.new(key, DES.MODE_ECB)

padded_text = pad(text1)
encrypted_text = des.encrypt(padded_text)

print(encrypted_text)
print(des.decrypt(encrypted_text))
```

Output:

```
b'\xa8\xc8\xee\xd5T6b?\xb49\x90h^r\x8a\xd1Q\x97\xa7"\xa8\x1a\xc1B+\xef\
b'This is the message I am encrypting!    '
```

Q. AES encryption and decryption implementation.

Code:

```
from Crypto.Cipher import AES

plaintext= b'This is the message to be encrypted'
key = b'Sixteen byte key'
cipher = AES.new(key, AES.MODE_EAX)
nonce = cipher.nonce
ciphertext, tag = cipher.encrypt_and_digest(plaintext)

print(ciphertext)

key = b'Sixteen byte key'
cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
plaintext = cipher.decrypt(ciphertext)
try:
   cipher.verify(tag)
   print("--authentic--")
   print("DecryptedText: ", plaintext)
except ValueError:
   print("Key incorrect or message corrupted”)
```

Output:

```
b'\x88\x05\xb6c\xac\xb8\xb6\x9fD=*\x8d\x02\xb3\r\x9c9\x10\xcf)\xdd\x94\x7f\
--authentic--
DecryptedText:  b'This is the message to be encrypted'
```

Q. Write a program to implement encryption of message using RSA. Inputs: Plain text: 6, p=7, q=17 ,Output Ciphertext=41.

Code:

```
import math
message = int(input("Enter the message to be encrypted: "))

p,q = int(input("Enter p: ")), int(input("Enter q: "))
n = p*q
print("n is : ",n)
PHI= (p-1)*(q-1)
print("Phi is : ",PHI)
d,e= None,None

for i in range(2,PHI):
    if math.gcd(i,PHI)==1:
        e=i
        break
print("e is : ",e)

for i in range(0,10):
    x= 1+ i*PHI
    if x%e == 0:
        d= int(x/e)
        break

print("d is : ",d)
c= int(math.pow(message,e))%n
print("Encrypted Message is: ", c)
message_back= (c**d)%n
print("Decrypted Message is: ",message_back)
```

Output:

```
Enter the message to be encrypted: 6
Enter p: 7
Enter q: 17
n is :  119
Phi is :  96
e is :  5
d is :  77
Encrypted Message is:  41
Decrypted Message is:  6
```

Q. Generate a message digest using SHA-224, SHA-256, SHA-384, SHA-512.

Code:

```
import hashlib
msg = b"Encrypt This"
print("Message: ", msg)

sha224Object = hashlib.sha224()
sha224Object.update(msg)
digest_sha224 = sha224Object.hexdigest()
print("Sha-224 hash:")
print(digest_sha224)

sha256Object = hashlib.sha256();
sha256Object.update(msg);
digest_sha256 = sha256Object.hexdigest();
print("Sha-256 hash:");
print(digest_sha256);

sha384Object = hashlib.sha384();
sha384Object.update(msg);
digest_sha384 = sha384Object.hexdigest();
print("Sha-384 hash:");
print(digest_sha384);

sha512Object = hashlib.sha512()
sha512Object.update(msg)
digest_sha512 = sha512Object.hexdigest()
print("Sha-512 hash:")
print(digest_sha512)
```

Output:

```
Message:  b'Encrypt This'
Sha-224 hash:
722307e92d8e85aa8875704e5eb9c1b9dee3f75a21544d955e0a80ae
Sha-256 hash:
e93c9a3c7683a1b6dcdadfe055f13f000be274d89a2f45cc3d43147571858ebe
Sha-384 hash:
4dca6e4e3fe013cfaa7e53a127b30b364f658b78e23cd09a0fa200d2954982191c7cdc121af
Sha-512 hash:
7b1cdfa99b91ef68358f29d1e1370b7e43d6de81c5c15d7a5401f1ac73cb80aab5381111d48
05a5560c85aa5
```

Q. Create digital signature using Digital Signature Algorithm(DSA).

Code:

```
from Crypto.PublicKey import DSA
from Crypto.Hash import SHA256
from Crypto.Signature import DSS

KEYSIZE= 1024
message= input("Enter Message: ").encode()
key= DSA.generate(KEYSIZE)

publickey= key.publickey()
print(publickey.exportKey())

message_hash= SHA256.new(message)
signer= DSS.new(key, 'fips-186-3')
signature= signer.sign(message_hash)
print(int.from_bytes(signature, "big", signed=False))

verifier= DSS.new(key, 'fips-186-3')

try:
  verifier.verify(message_hash, signature)
  print("Verification Successful")
except ValueError:
  print("Verification Failed”)
```

Output:

```
Enter Message: This Message
b'-----BEGIN PUBLIC KEY-----\nMIIBtzCCASsGByqGSM44BAEwggEeA
Z8/0EVRxQyGt+yCyenO9RKL166dUdcwKV9h03Seb0IiFR\n7o9dQGk1gvj7
yWpJeIvg/hAhUA/v8/FuE7AqfsHbuM5kDQqJnAe6ECgYAq9dMELYW8kxK5Q
3kQ4YxSoVT/HD\neCKqIetoP229ZKDp8GRgulqEaeFUpa8LgN3hfMQf+seO
EA24Wp67r5rFOiBmoMON1Fq9uekKcP\n2kHb4aJLSNTk6cnkhRmNbxUYc2v
9yNE5wiDMDYLlvFqlfOfDSaYERknIBlJJ4S9MvsWnggi938\nFTvSaMLD3P
185314610968439737262308560771599137923042908189436718393880
Verification Successful
```