ENHANCEMENT OF AES , DES AND RSA USING MERGING ALGORITHM

(Group 3)

INTRODUCTION:

All messages have different level of sensitivity . Some are more sensitive while some are less. Hence the User has to change the algo repeatedly according to the sensitivity of message he/she wants to send.. Even cryptography api can be insecure apis which gives third party access by creating backdoor entries so that the attacker can get access. To fix this we can provide user with options to switch within a single program that has multiple types of ways to encrypt according to the needs of user. Some will use a completely different technique from others. Keeping in mind the efficiency without compromisation of confidentiality, security and integrity. Every cryptographic algorithm is vulnerable , the best algorithm is that which is very difficult to crack. Enabling double or triple factor authentication and encryption will make it less vulnerable to attacker.

PROBLEM STATEMENT:

Current algorithm of encrypting and decrypting i.e. conventional and public key cryptography – Few techniques use patterns to encode which is at some point of time decoded key cryptography has an issue of storage and even an attacker can send the victim with a fake key. Some widely used techniques whether it is symmetric algorithms like AES, or asymmetric algorithms like RSA or even hashing like MD5,SHA256 requires some or other types of keys Managing encryption keys adds to the workload. Accessing the encrypted data is difficult. Also Performance might be hampered by key management systems.

SOLUTION:

So to enhance the security towards encryption and decryption, merging algorithm takes place a major role, here, we use algorithms to encode a key by using different pattern for each characters. And another merging code also been shown which creates multiple keys for a single plaintext, that, will be entered in the same order to decrypt again. With the help of standard cryptographic algorithms like AES, DES and RSA. We were able to use modified algorithms to improve the time complexity with lesser rounds.

LITERATURE SURVEY:

1. A hardware implementation of a modified DES algorithm by T. Kropf, W. Beller, T. Giesler, 2003:
   This work describes a hardware implementation of an upgraded version of the Data Encryption Standard encryption algorithm. Because of the increased key length of 768 bits, this adjustment results in a significantly more secure improvement. Upward compatibility is assured, allowing the chip to be used for regular DES encryption. The implementation supports all four standard encryption modes (ECB, CBC, CFB, and OFB). A common interface enables flexible use in a variety of contexts. The chip's encryption kernel achieves a ciphering rate of more than 40 Mbit/s, which is only limited to around 10 Mbit/s by the selected, completely asynchronous handshake protocol. The chip has been manufactured and tested satisfactorily.

2. An implementation of DES and AES, secure against attacks by Mehdi-Laurent Akkar, Christophe Giraud, 2001:
   Since Paul Kocher's introduction of Power Analysis on smart cards, several countermeasures have been developed to safeguard cryptographic method implementations. We offer a novel protection concept in this paper: the altered masking approach. We use this strategy to secure two of the most widely used block cyphers: DES and AES Rijndael. To that aim, we provide several altered S-boxes for DES as well as a new masking approach with applicability to the non-linear section of Rijndael.

3. An analysis of NewDES: a modified version of DES by Charles Connell, 1990:
   This study investigates Robert Scott's encryption technique, which is a modified version of the Data Encryption Standard. Scott's purpose in modifying DES is to overcome two perceived weaknesses: the length of the key and the "Secret-ness" of the design of the S-boxes. His secondary objective is to develop an algorithm that is simple to implement in software on a microcomputer. Scott's algorithm is straightforward to implement. One of the key reasons behind this is that it only performs operations on full bytes, therefore no individual bit manipulation occurs.

Modifications to AES Algorithm:

- This proposed project can be used to encrypt the file and other relevant information to provide additional security to the owner of the file.
- The enhanced cryptographic algorithm has successfully shared and distributed files among users and has successfully encrypted the file in the server using the merging algorithm.
- This project was proposed to increase the time complexity using modified AES and easy-access with more security.

Multi Modular Chaotic Logistic Map for Image Encryption:

This paper helps in enhancing authentication using image encryption with help of fuzzy logistic map. This paper shows a new behavior towards bifurcation diagram to show better performance using random tests. Since the proposed paper showed excellent results, it can be used as an application in image encryption. Also, proved increase in security and complexity. The proposed chaotic system used one modular chaotic logistic map as a master map to inject the main parameters for the parallel multi-modular chaotic logistic maps. Also, the fuzzy set theory is used as a fuzzy logic selector to improve chaotic performance.

Digital signature:

The digital signature of the original image is added to the encoded version of the original image. The encoding of the image is done using an appropriate error control code, such as a Bose-Chaudhuri Hochquenghem (BCH) code. At the receiver end, after the decryption of the image, the digital signature can be used to verify the authenticity of the image. This paper demonstrated a new technique that uses digital signatures to encrypt the image. This technique works well with images of all sizes. This encryption technique provides three layers of security. In the first step, an error control code is used which is determined in real-time, based on the size of the input image. Without the knowledge of the specific error control code, it is very difficult to obtain the original image and tamper with it. The dimension of the image also changes due to the added redundancy. This poses an additional difficulty to decrypt the image.

- For DES , we have increased the number of rounds
- GUI makes this proposed project much more efficient and user-friendly

Programming language used : Python

Algorithms : Modified AES,RSA and DES

ENTIRE PROJECT IN ALGORITHM:

ALGORITHMS:

MODIFIED AES:

- Even though modification takes place into the original AES algorithm, the security remains same.It has been found that increasing the number of rounds provides more security.
- Using Bit permutation- Not a complex mathematics but only shifting of positions of bits of every state.
- Bit Permutation- Encryption
- Inverse Bit Permutation- Decryption
- Adding new key: Using AES algorithm and Modified key(additional key), which will be XOR'ed with plain text and then encrypted by public key. This process is known as InitialAddRoundKey.
- Reconfiguration in subBytes: Inserting a new operation in existing subByte operation, namely, Modified Transport.

Data Encryption Standard (DES) :

- The key length is 56 bits. In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.

- The initial permutation is performed on plain text. Next, the initial permutation (IP) produces two halves of the permuted block; says Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process. In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block. The result of this process produces 64-bit ciphertext.

- DES has a 56-bit key which raises the possibility of 2^{56} possible keys which make brute force impossible.
- The 8 S-boxes used in each round were not made public and even it impossible for any to discover the design of the s-boxes which makes the attack more impossible.
- The number of rounds in DES increases the complexity of the algorithm.
- However, the cryptanalysis attack is easier than the brute force attack on DES.

MODIFIED RSA:

- This algorithm uses a mod operator for computational purposes.
- The objective of this algorithm presents the implementation of successive subtraction operation instead of using division operator.
- A pair of positive integers and forms a new pair that consists of the smaller number and the difference between the larger and smaller numbers. The process repeats until the numbers are equal; then that value is the greatest common divisor of the original pair.
- The division form of Euclid's algorithm starts with a pair of positive integers and forms a new pair that consists of the smaller number and the remainder obtained by dividing the larger number by the smaller number. The process repeats until one number is zero.
- The other number then is the greatest common divisor of the original pair. This idea to reduce the mathematical steps to solve that expression. So we conclude that easily computation can be performed and complexity was reduced. Complexity time also decreased.

PATTERN ALGORITHM:

This algorithm is to create a pattern for each character and to encrypt it using that specific pattern.

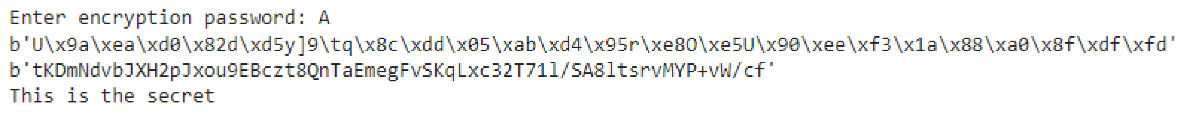
MODIFIED AES:

CODE:

```
# AES 256 encryption/decryption using pycrypto library
#!pip install pycrypto
import base64
import hashlib
import Crypto
from Crypto.Cipher import AES
from Crypto import Random

BLOCK_SIZE = 16
pad = lambda s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * chr(BLOCK_SIZE
- len(s) % BLOCK_SIZE)
unpad = lambda s: s[:-ord(s[len(s) - 1:])]


password = input("Enter encryption password: ")


def encrypt(raw, password):
    private_key = hashlib.sha256(password.encode("utf-8")).digest()
    print(private_key)
    raw = pad(raw)
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
    return base64.b64encode(iv + cipher.encrypt(raw))


def decrypt(enc, password):
#   private_key=input()
    private_key = hashlib.sha256(password.encode("utf-8")).digest()
    enc = base64.b64decode(enc)
    iv = enc[:16]
    cipher = AES.new(private_key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(enc[16:]))
```

```python
# First let us encrypt secret message
encrypted = encrypt("This is the secret ", password)
print(encrypted)

# Let us decrypt using our original password
decrypted = decrypt(encrypted, password)
print(bytes.decode(decrypted))
```

OUTPUT:



```
Enter encryption password: A
b'U\x9a\xea\xd0\x82d\xd5y]9\tq\x8c\xdd\x05\xab\xd4\x95r\xe8O\xe5U\x90\xee\xf3\x1a\x88\xa0\x8f\xdf\xfd'
b'tKDmNdvbJXH2pJxou9EBczt8QnTaEmegFvSKqLxc32T71l/SA8ltsrvMYP+vW/cf'
This is the secret
```

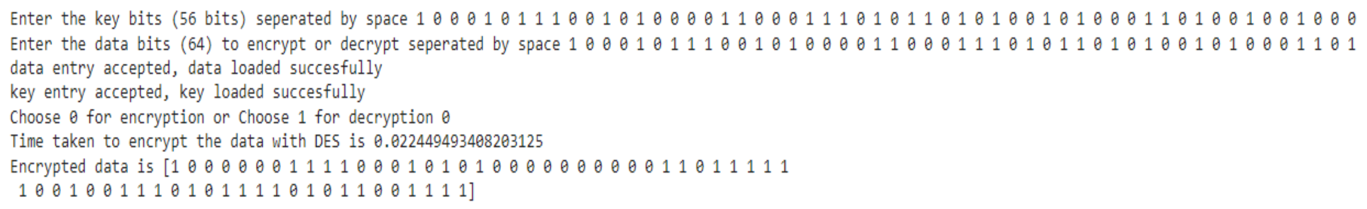
Data Encryption Standard (DES) :

CODE:

```python
import numpy as np
import time

IP = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]

FP = [40, 8, 48, 16, 56, 24, 64, 32,
        39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30,
        37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28,
        35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25]

EBox = [32,1,2,3,4,5,
            4,5,6,7,8,9,
            8,9,10,11,12,13,
            12,13,14,15,16,17,
            16,17,18,19,20,21,
            20,21,22,23,24,25,
            24,25,26,27,28,29,
            28,29,30,31,32,1]

SBox =[
    # S1
    [14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
     0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
     4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
     15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],

    # S2
    [15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
     3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
     0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
     13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],

    # S3
    [10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
     13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
     13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
     1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],

    # S4
    [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
     13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
     10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
     3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],

    # S5
    [2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
     14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
     4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
     11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],

    # S6
    [12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
     10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
     9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
         4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],

        # S7
        [4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
         13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
         1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
         6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],

        # S8
        [13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
         1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
         7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
         2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],
  ]

F_PBox = [16, 7, 20, 21, 29, 12, 28, 17,
              1, 15, 23, 26, 5, 18, 31, 10,
              2, 8, 24, 14, 32, 27, 3, 9,
              19, 13, 30, 6, 22, 11, 4, 25 ]

key_PBox = [14,    17,   11,    24,     1,    5,
                 3,    28,   15,     6,    21,   10,
                23,    19,   12,     4,    26,    8,
                16,     7,   27,    20,    13,    2,
                41,    52,   31,    37,    47,   55,
                30,    40,   51,    45,    33,   48,
                44,    49,   39,    56,    34,  53,
                46,    42,   50,    36,    29,   32]


def xor(left,xorstream):
    xorresult = np.logical_xor(left,xorstream)

    xorresult  = xorresult.astype(int)

    return xorresult

def E_box(right):
    expanded = np.empty(48)
    j = 0
    for i in EBox:
        expanded[j] = right[i - 1]
        j += 1
    expanded = list(map(int,expanded))
    expanded = np.array(expanded)
    return expanded

#clean this code please (sboxlookup)
def sboxloopup(sinput,x):
    tableno = x - 1
    row = int((np.array2string(sinput[0]) +
np.array2string(sinput[5])),2)

    # make this part of the code better
    column = sinput[1:5]
    column = np.array2string(column)
    column = column[1:8].replace(" ", "")
    column = int(column,2)
    # print(column,"column")

    elementno = (16 * row) + column
    soutput = SBox[tableno][elementno]
    soutput = list(np.binary_repr(soutput, width=4))
    #converting to list twice seems redundant but seems to be the only
simple way as map always returns map object
    soutput= np.array(list(map(int, soutput)))
    return soutput

def sbox(sboxin):
#takes 48 bit input and return 32 bit
    sboxin1 = sboxin[0:6]
    sboxout1 = sboxloopup(sboxin1,1)
    sboxin2 = sboxin[6:12]
    sboxout2 = sboxloopup(sboxin2,2)
    sboxin3 = sboxin[12:18]
    sboxout3 = sboxloopup(sboxin3, 3)
    sboxin4 = sboxin[18:24]
    sboxout4 = sboxloopup(sboxin4, 4)
    sboxin5 = sboxin[24:30]
    sboxout5 = sboxloopup(sboxin5, 5)
    sboxin6 = sboxin[30:36]
    sboxout6 = sboxloopup(sboxin6, 6)
    sboxin7 = sboxin[36:42]
    sboxout7 = sboxloopup(sboxin7, 7)
    sboxin8 = sboxin[42:48]
    sboxout8 = sboxloopup(sboxin8, 8)
    sboxout =
np.concatenate([sboxout1,sboxout2,sboxout3,sboxout4,sboxout5,sboxout6,s
boxout7,sboxout8])
    return sboxout

def f_permute(topermute):
    permuted= np.empty(32)
    j = 0
    for i in F_PBox:
        permuted[j] = topermute[i - 1]
        j += 1
    return permuted

def f_function(right,rkey):
    expanded = E_box(right)
    xored = xor(expanded,rkey)
    sboxed = sbox(xored)
    xorstream = f_permute(sboxed)
    return xorstream

def round(data,rkey):
    l0 = data[0:32]
    r0 = data[32:64]
    xorstream = f_function(r0,rkey)
    r1 = xor(l0,xorstream)
    l1 = r0
    returndata = np.empty_like(data)
    returndata[0:32] = l1
    returndata[32:64] = r1
    return(returndata)

def permutation(data,x):
    #intial and final permutation conditional based on other passed
value
    permute1 = np.empty_like(IP)
    if x == 0:
        j = 0
        for i in IP:
            permute1[j] = data[i-1]
            j += 1
        return(permute1)
    else:
        permute2 = np.empty_like(FP)
        k = 0
        for l in FP:
            permute2[k] = data[l-1]
            k += 1
        return(permute2)

def userinput():
    keyinp = input("Enter the key bits (56 bits) seperated by space "
"").strip().split()
    datainp = input("Enter the data bits (64) to encrypt or decrypt
seperated by space " "").strip().split()
    #change to 56 later
    lenofkey = 56
    #change to 64 later
    lenofdata = 64
    if len(datainp) == lenofdata and len(keyinp) == lenofkey:
        print("data entry accepted, data loaded succesfully")
        print("key entry accepted, key loaded succesfully")
    else:
        while len(datainp) != lenofdata:
            print("length of data entered ",len(datainp))
            datainp = input("Error in entered data. Enter the data (64
bits) to encrypt or decrypt seperated by space " "").strip().split()

        print("data entry accepted, data loaded succesfully")
        while len(keyinp) != lenofkey:
            print("length of key entered ", len(keyinp))
            keyinp = input("Error in entered key. Enter the key (56
bits) to encrypt or decrypt seperated by space " "").strip().split()
        print("key entry accepted, key loaded succesfully")
#also add functionality to accept 64 bit keys instead of 54
    return keyinp,datainp


def keyshift(toshift,n):
    if (n == 1) or (n == 2) or (n == 9) or (n == 16):
        toshift= np.roll(toshift,-1)
        return toshift
    else:
        toshift = np.roll(toshift, -2)
        return toshift

def keypermute(key16):
    keypermuted = np.empty([16,48])
    l = 0
    for k in key16:
        j = 0
        for i in key_PBox:
            keypermuted[l][j] = k[i - 1]
            j += 1
        l += 1
    return keypermuted

#
def keyschedule(key):
    left = key[0:28]
    right = key[28:56]
    shifted = np.zeros(56)
    key16 = np.zeros([16,56])
    for i in range(1,17):
        shifted[0:28] = keyshift(left,i)
        shifted[28:56] = keyshift(right,i)
        left = shifted[0:28]
        right = shifted[28:56]
#add shifted to key16 and return key16
        key16[i - 1] = shifted
#key16 is the final shifted 16 key pair now to permute
    key16 = keypermute(key16)
    key16 = [list(map(int, x)) for x in key16]
    key16 = np.array(key16)
    return key16


def main():
    key, data = userinput()
    # key = ['1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1',
'1', '0', '0', '0', '1', '1', '1', '0', '0', '1', '1', '1', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0',
'0', '0', '0']
    # data = ['1', '1', '1', '1', '1', '1', '1', '0', '0', '1', '1',
'1', '0', '0', '0', '1', '1', '1', '0', '0', '1', '1', '1', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '0', '0',
'0', '1', '1', '1', '0', '0', '0', '1', '1', '1', '0', '0', '0', '0',
'0', '0', '0', '0', '1', '0', '1', '1', '0', '1', '0']
    # # print(data,key)
```

```
#taking input for decryption and encryption
    operate = int(input("Choose 0 for encryption or Choose 1 for 
decryption "))
    starttime = time.time()
    key16 = keyschedule(key)

    if operate == 0:
        data = permutation(data,0)
# testing round function now
        for i in range(16):
            data = round(data,key16[i])


#making left side right and right side left
        data = np.roll(data,32)
        data = (permutation(data, 1))
        print("Time taken to encrypt the data with DES is", time.time() 
- starttime)
        print("Encrypted data is", data)

    if operate == 1:
        data = permutation(data, 0)
        # testing round function now
        for i in range(16):
            data = round(data, key16[16 - (i + 1)])

        data = np.roll(data, 32)
        data = (permutation(data, 1))
        print("Time taken to decrypt the data with DES is", time.time() 
- starttime)
        print("Decrypted data is", data)



#real main stops




#
# #testing with madeup data
#     data1 = np.array([0,1,0,1,0,0,0,1])
#
# #testing IP and FP
#     print(permutation(data, 0))
#     print(permutation(data, 1))
#     rkey1 = np.array(range(0,64))
    # # data1 = round(data,rkey1)
    # data2 = np.array([1,1,1,1,0,1,1,1])
    # print(xor(data1,data2))

#ebox testing
    # data1 = np.array(range(0,32))
    # print(len(E_box(data1)), E_box(data1))

    # sboxtest =
np.array([0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,1,0,0,0,1,1,0,0,0,1,0,0,0
,0,1,1,1,0,0,1,0,1,0,0,0,1,1,0,0,1])
    # print(len(sboxtest))
    # sboxoutput = sbox(sboxtest)
    # print(sboxoutput)



    # key =
np.array([0,1,1,1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,1,0,0,0,1,1,0,0,0,1,0,0,0
,0,1,1,1,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,1,1,0])
    # key16 = keyschedule(key)
    # print(key16)
#calling key schedule
    # for i in range(0,16):
    # roundkey = key16[i]
    # print(roundkey)
    #     roundresult = round(data,roundkey)
    #     data = roundresult



    # totestshift =
np.array([0,1,1,1,1,1,1,1,0,0,0,1,1,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1])
    # print("test data", totestshift)
    # print(keyshift(totestshift,1))
    # print(keyshift(totestshift,3))
    #
    # permutekeygen = list(range(1,57))
    # key16permutetest = np.empty([16,56])
    # for i in range(16):
    #     key16permutetest[i] = permutekeygen
    #
    # keypermute(key16permutetest)
    #



main()
```

OUTPUT:



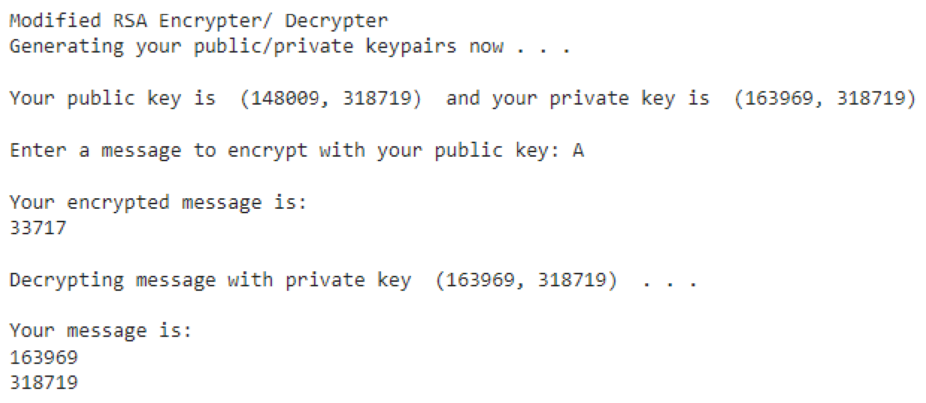
```
Enter the key bits (56 bits) seperated by space 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0
Enter the data bits (64) to encrypt or decrypt seperated by space 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1
data entry accepted, data loaded succesfully
key entry accepted, key loaded succesfully
Choose 0 for encryption or Choose 1 for decryption 0
Time taken to encrypt the data with DES is 0.022449493408203125
Encrypted data is [1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1
 1 0 0 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1]
```

MODIFIED RSA:

CODE:

```
import random
import pandas as pd


'''
Euclid's algorithm for determining the greatest common divisor
Use iteration to make it faster for larger integers
'''
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


'''
Tests to see if a number is prime.
'''
def is_prime(num):
    if num == 2:
        return True
    if num < 2 or num % 2 == 0:
        return False
    for n in range(3, int(num**0.5)+2, 2):
        if num % n == 0:
            return False
    return True


'''
Euclid's extended algorithm for finding the multiplicative inverse of
two numbers
'''
def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi

    while e > 0:
        temp1 = (int)(temp_phi/e)
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2

        x = x2- temp1* x1
        y = d - temp1 * y1

        x2 = x1
        x1 = x
        d = y1
        y1 = y

    if temp_phi == 1:
        return d + phi


def generate_keypair(p, q, r):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')
    #n = pqr
    n = p * q * r
    #Phi is the totient of n
    phi = (p-1) * (q-1) * (r-1)

    #Choose an integer e such that e and phi(n) are coprime
    e = random.randrange(1, phi)

    #Use Euclid's Algorithm to verify that e and phi(n) are comprime
    g = gcd(e, phi)

    while g != 1:
        e = random.randrange(1, phi)
        g = gcd(e, phi)

    #Use Extended Euclid's Algorithm to generate the private key
    d = multiplicative_inverse(e, phi)

    #Return public and private keypair
    #Public key is (e, n) and private key is (d, n)
    return ((e, n), (d, n), phi)

def encrypt(plaintext):
    data1 = pd.read_csv("data1.csv")
    data2 = pd.read_csv("data2.csv")

    key = int(data2['0'][1]) # e
    n = int(data1['0'][3])


    #Convert each letter in the plaintext to numbers based on the
character using a^b mod m
    cipher = [(ord(char) ** int(key)) % n for char in plaintext]
    #Return the array of bytes
    return cipher

def decrypt(ciphertext):
    data1 = pd.read_csv("data1.csv")
    data2 = pd.read_csv("data2.csv")

    key = int(data2['0'][2]) # d
    n = int(data1['0'][3])
    print(key)
    print(n)
    #Generate the plaintext based on the ciphertext and key using a^b
mod m
    plain = [chr((char ** key) % n) for char in ciphertext]
    #Return the array of bytes as a string
    return ''.join(plain)


if __name__ == '__main__':

    print("\n\nModified RSA Encrypter/ Decrypter")

    primes=[]
    total_no_primes = 0
    with open('primes.txt') as pfile:
        for line in pfile:
            primes.append(int(line)) # = [int(i) for i in line.split()]
            total_no_primes += 1
    # 3 prime nos
    p = primes[random.randint(1, total_no_primes-1)]
    q = primes[random.randint(1, total_no_primes-1)]
    r = primes[random.randint(1, total_no_primes-1)]


    print("Generating your public/private keypairs now . . .")
    public, private, phi = generate_keypair(p, q, r)
    print("\nYour public key is ", public ," and your private key is ",
private)

    data1 = [p, q, phi, public[1]]  #  p, q, phi, n
    df = pd.DataFrame(data1)
    df.to_csv('data1.csv') # offline storage of p, q, phi, n in table 1

    data2 = [r, public[0], private[0]] #  r, e, d
    df = pd.DataFrame(data2)
    df.to_csv('data2.csv') #offline storage of r, e, d in table 2

    message = input("\nEnter a message to encrypt with your public key:
")
    encrypted_msg = encrypt(message)
    print("\nYour encrypted message is: ")
    print(''.join([str(x) for x in encrypted_msg]))
    print("\nDecrypting message with private key ", private ," . . .")
    print("\nYour message is:")
    print(decrypt(encrypted_msg))
```

OUTPUT:



```
Modified RSA Encrypter/ Decrypter
Generating your public/private keypairs now . . .

Your public key is  (148009, 318719)  and your private key is  (163969, 318719)

Enter a message to encrypt with your public key: A

Your encrypted message is:
33717

Decrypting message with private key  (163969, 318719)  . . .

Your message is:
163969
318719
```

MODIFIED MERGING ALGORITHM:

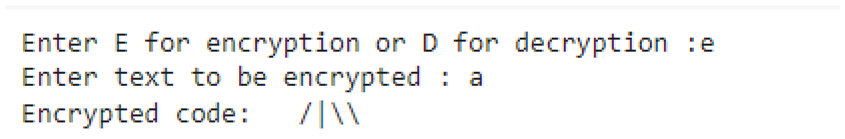
PATTERN ALGORITHM:

CODE:

```
PED_DICT = { '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':'||//\\',
    '1':'/||||\\', '2':'//|||\\', '3':'///||\\',
    '4':'////|\\', '5':'/////\\', '6':'|////\\',
    '7':'||///\\\\', '8':'|||//\\', '9':'||||/\\',
    '0':'////\\\\', ', ':'||//||\\', '.':'/|/|/|',
    '?':'//||//', '/':'|//|/', '-':'|////|',
    '(':'|/||/\\\\', ')':'|/||/\\|',
}

def encrypt(message):
    cipher = ''
    for letter in message:
        if letter != ' ':
            cipher +=PED_DICT[letter] + ' '
        else:
            cipher += ' '

    return cipher


def decrypt(message):
    message += ' '
    decipher = ''
    citext = ''
    for letter in message:
        if (letter != ' '):
            i = 0
            citext += letter
        else:
            i += 1
            if i == 2 :
                decipher += ' '
            else:
                decipher += list(PED_DICT.keys())[list(PED_DICT
                .values()).index(citext)]
                citext = ''

    return decipher
df = pd.DataFrame(list(PED_DICT.items()),columns =
['Characters','codes'])


def main():
   ED=input("Enter E for encryption or D for decryption :")
   if ED=="E" or 'e':
     message = input("Enter text to be encrypted : ")
     output = encrypt(message.upper())
     print ("Encrypted code:  ",output)
   elif ED=="D" or 'd':
    message = input("Enter message to be decrypted ")
    output = decrypt(message)
    print ("Decrypted text:  ",output)
   #elif ED=="L":
   # print(df)
   else:
    print("Enter only E or D")
```

```
# Executes the main function
if __name__ == '__main__':
  main()
```

OUTPUT:



```
Enter E for encryption or D for decryption :e
Enter text to be encrypted : a
Encrypted code:   /|\\
```

ASCII ALGORITHM:

CODE:

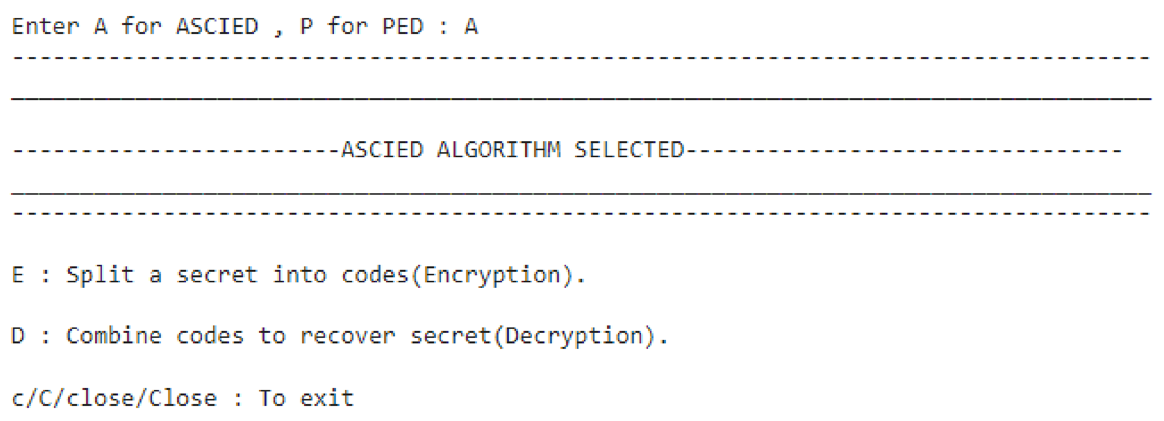
```
try:
    import os
    import getpass
    import secrets
    import sys
except ImportError:
    print('Critical Error: Required Modules Not found!\n')
    x = input('Press any key to continue...')
    sys.exit(1)

A = ('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', '0', '1',
'2', '3', '4', '5', '6', '7', '8', '9')


# converts Alphanumeric characters to numbers of base 36
def f(x):
  store = []
  for s in x:
    count = 0
    for i in range(36):
        if A[i].lower() == s.lower():
          store.append(i)
          count = 1
          break
    if count == 0:
      store.append(' ')
  return tuple(store)


# converts base 36 numbers to alphanumeric charactors.
def rf(x):
    store = []
    q = ''
    for s in x:
        try:
            store.append(A[s])
        except(IndexError, TypeError):
            store.append(' ')
    q = ''.join(store)
    return q

# generates a key without keyfile.
def ikey(x):
    seed = list(range(36))
    masterkey = []
    for i in range(len(x)):
        masterkey.append(secrets.choice(seed))
    return tuple(masterkey)


# encrypts a given string and returns ciphertxt and key as a tuple. (no
file generated!)
def en(msg):
    ciphertxt = []
    x = f(msg)
    y = ikey(msg)
    for i in range(len(x)):
            if type(x[i]) is int:
                ciphertxt.append(((x[i]+y[i]) % 36))
            else:
                ciphertxt.append(' ')
    ctxt = rf(tuple(ciphertxt))
    shk = rf(y)
    return (ctxt, shk)
```

```
# decrypts a given encrypted string and returns a plaintxt as output.
def de(c, k):
    ciphertxt = []
    x = f(c)
    y = f(k)
    if len(x) <= len(y):
        for i in range(len(x)):
            if type(x[i]) is int and type(y[i]) is int:
                ciphertxt.append(((x[i]-y[i]) % 36))
            else:
                ciphertxt.append(' ')
    else:
        x = input('Incorrect Input!!!\nPress any key to continue...')
        sys.exit(1)
    return rf(tuple(ciphertxt))


# function for secret splitting interface.
def sprocess():
    table = []

print('''\n-----------------------------------------------------------
               |            Secret splitting                |

-----------------------------------------------------------''')
    while 1:
        try:
            x = int(input('\nEnter the number of shares(atmost
10,atleast 2):'))
            if 1 < x < 11:
                break
        except ValueError:
            print('\nPlease enter a valid integer greater than 1 but
less than or equal to 10!\n')
    msg = getpass.getpass('Enter the secret:')
    table += list(en(msg))
    for i in range(2, x):
        tmp = table[-1]
        table.pop()
        table += list(en(tmp))
    for i in range(len(table)):
        print('SHARE', i+1, ':', table[i])
```

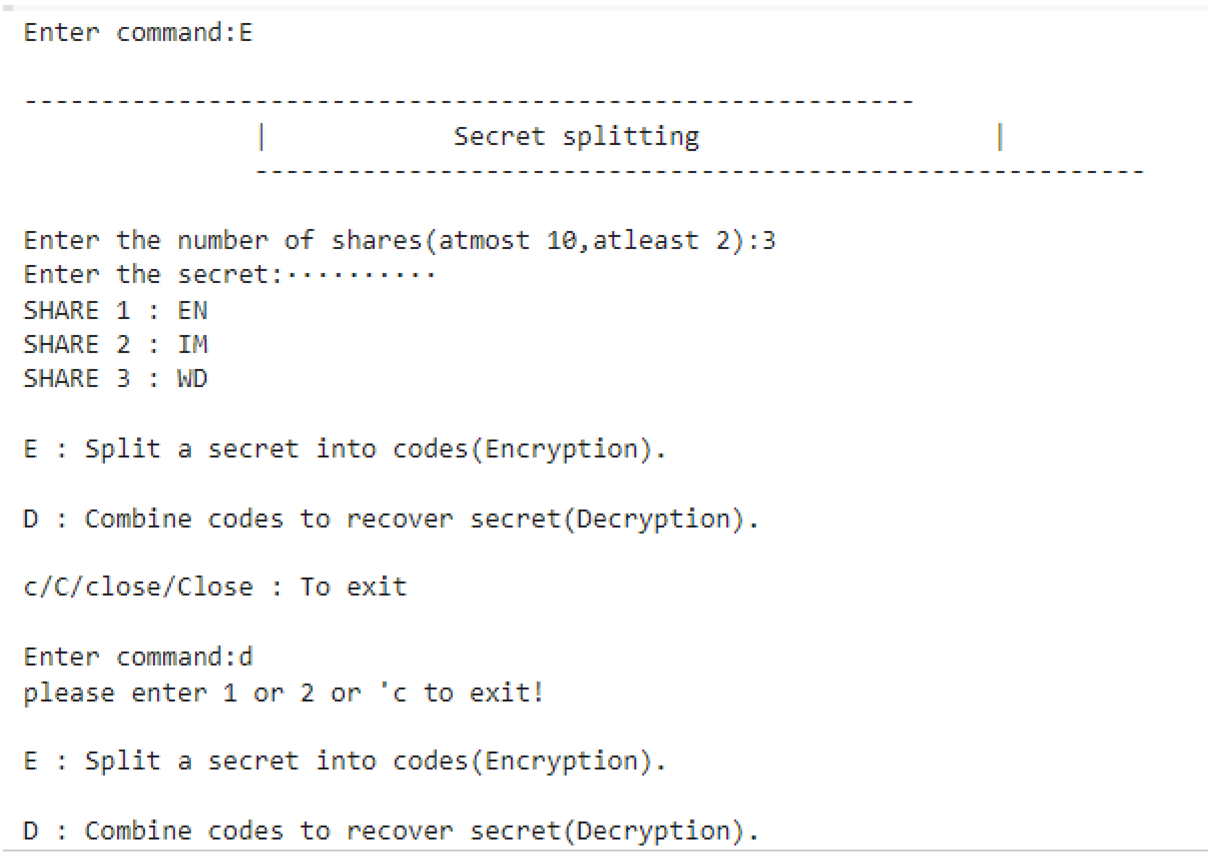
```
# function for secret combining interface.
def cprocess():
    table = []
    print('''\n
---------------------------------------------------------
                |          Secret Combine                     |

-----------------------------------------------------------''')
    while 1:
        try:
            x = int(input('\nEnter no. of shares to combine(atmost
10,atleast 2):'))
            if 1 < x < 11:
                break
        except ValueError:
                print('\nPlease enter a valid integer greater than 1
but less than or equal to 10!\n')
    for i in range(x):
            table.append(getpass.getpass(str('Enter Share
'+str(i+1)+':')))
    for i in range(x-1):
            hook = []
            a, b = table[-2], table[-1]
            table.pop()
            table.pop()
            hook.append(de(a, b))
            table += hook
    print()
    print(''.join(table))


# function for main interface.
def mm():
   print('\nE : Split a secret into codes(Encryption).')
   print('\nD : Combine codes to recover secret(Decryption).')
   print('\nc/C/close/Close : To exit')
   cmd = input('\nEnter command:')
   if cmd == 'E':
       sprocess()
       mm()
   elif cmd == 'D':
       cprocess()
```

```
        mm()
    elif cmd.lower() == 'c' or cmd.lower() == 'close':
        sys.exit()
    else:
        print('please enter 1 or 2 or \'c to exit!')
        mm()
    sys.exit()

#Merging both

select_=input('Enter A for ASCIED , P for PED : ')
if select_ == 'A':

print('----------------------------------------------------------------
-------------------')

print('________________________________________________________________
___________________')
  print("\n------------------------ASCIED ALGORITHM
SELECTED--------------------------------")

print('________________________________________________________________
___________________')

print('----------------------------------------------------------------
-------------------')
  #mer=input("Enter E for encryption or D for decryption: ")
  mm()
elif select_ == 'P':
##  if __name__ == '__main__':
   main()
```

OUTPUT:



```
Enter A for ASCIED , P for PED : A
---------------------------------------------------------------------------
___________________________________________________________________________

----------------------ASCIED ALGORITHM SELECTED-----------------------------
___________________________________________________________________________
---------------------------------------------------------------------------

E : Split a secret into codes(Encryption).

D : Combine codes to recover secret(Decryption).

c/C/close/Close : To exit
```



```
Enter command:E

----------------------------------------------------------
               |             Secret splitting             |
               ----------------------------------------------------------

Enter the number of shares(atmost 10,atleast 2):3
Enter the secret:··········
SHARE 1 : EN
SHARE 2 : IM
SHARE 3 : WD

E : Split a secret into codes(Encryption).

D : Combine codes to recover secret(Decryption).

c/C/close/Close : To exit

Enter command:d
please enter 1 or 2 or 'c to exit!

E : Split a secret into codes(Encryption).

D : Combine codes to recover secret(Decryption).
```

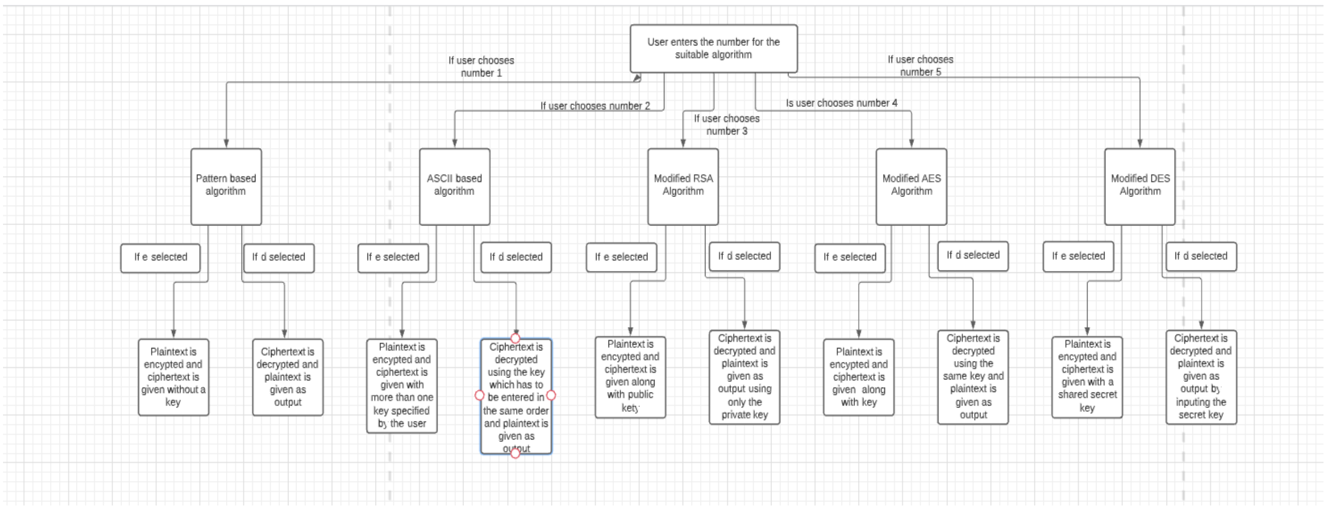
GUI (Tkinter)

```
from tkinter import *

# import other necessary modules
import random
import time
import datetime

# creating root object
root = Tk()

# defining size of window
root.geometry("1200x6000")

# setting up the title of window
root.title("Message Encryption and Decryption")

Tops = Frame(root, width = 1600, relief = SUNKEN)
Tops.pack(side = TOP)

f1 = Frame(root, width = 800, height = 700,
                        relief = SUNKEN)
f1.pack(side = LEFT)

# ==============================================
#               TIME
# ==============================================
localtime = time.asctime(time.localtime(time.time()))

lblInfo = Label(Tops, font = ('helvetica', 50, 'bold'),
        text = "SECRET MESSAGING ",
                fg = "Black", bd = 10, anchor='w')

lblInfo.grid(row = 0, column = 0)

lblInfo = Label(Tops, font=('arial', 20, 'bold'),
          text = localtime, fg = "Steel Blue",
                        bd = 10, anchor = 'w')

lblInfo.grid(row = 1, column = 0)

rand = StringVar()
Msg = StringVar()
key = StringVar()
```

```
mode = StringVar()
Result = StringVar()
Algo=StringVar()

# exit function
def qExit():
    root.destroy()

# Function to reset the window
def Reset():
    rand.set("")
    Msg.set("")
    key.set("")
    mode.set("")
    Algo.set("")
    Result.set("")



# reference
'''
lblReference = Label(f1, font = ('arial', 16, 'bold'),
                text = "Name:", bd = 16, anchor = "w")

lblReference.grid(row = 0, column = 0)
'''
txtReference = Entry(f1, font = ('arial', 16, 'bold'),
               textvariable = rand, bd = 10, insertwidth = 4,
                        bg = "powder blue", justify = 'right')

txtReference.grid(row = 0, column = 1)

# labels
lblMsg = Label(f1, font = ('arial', 16, 'bold'),
         text = "MESSAGE", bd = 16, anchor = "w")

lblMsg.grid(row = 1, column = 0)

txtMsg = Entry(f1, font = ('arial', 16, 'bold'),
         textvariable = Msg, bd = 10, insertwidth = 4,
                bg = "powder blue", justify = 'right')

txtMsg.grid(row = 1, column = 1)

lblkey = Label(f1, font = ('arial', 16, 'bold'),
            text = "KEY", bd = 16, anchor = "w")
```

```
lblkey.grid(row = 2, column = 0)

txtkey = Entry(f1, font = ('arial', 16, 'bold'),
        textvariable = key, bd = 10, insertwidth = 4,
              bg = "powder blue", justify = 'right')

txtkey.grid(row = 2, column = 1)

#Mode
lblmode = Label(f1, font = ('arial', 16, 'bold'),
         text = "Type e for encryption or d for decryption)",
                        bd = 16, anchor = "w")

lblmode.grid(row = 3, column = 0)



txtmode = Entry(f1, font = ('arial', 16, 'bold'),
         textvariable = mode, bd = 10, insertwidth = 4,
                bg = "powder blue", justify = 'right')

txtmode.grid(row = 3, column = 1)

lblService = Label(f1, font = ('arial', 16, 'bold'),
           text = "The Result-", bd = 16, anchor = "w")

lblService.grid(row = 2, column = 2)

txtService = Entry(f1, font = ('arial', 16, 'bold'),
           textvariable = Result, bd = 10, insertwidth = 4,
                   bg = "powder blue", justify = 'right')

txtService.grid(row = 2, column = 3)



#algochoose
lblalgo = Label(f1, font = ('arial', 16, 'bold'),
         text = "Enter A for ASCIED , P for PED",
                        bd = 16, anchor = "w")
lblalgo.grid(row = 0, column = 0)
# Vigenère cipher
import base64

# Function to encode
def encode(key, clear):
    enc = []
    for i in range(len(clear)):
        key_c = key[i % len(key)]
        enc_c = chr((ord(clear[i]) +
                    ord(key_c)) % 256)

        enc.append(enc_c)

    return base64.urlsafe_b64encode("".join(enc).encode()).decode()

# Function to decode
def decode(key, enc):
    dec = []

    enc = base64.urlsafe_b64decode(enc).decode()
    for i in range(len(enc)):
        key_c = key[i % len(key)]
        dec_c = chr((256 + ord(enc[i]) -
                       ord(key_c)) % 256)

        dec.append(dec_c)
    return "".join(dec)

try:
    import os
    import getpass
    import secrets
    import sys
except ImportError:
    print('Critical Error: Required Modules Not found!\n')
    x = input('Press any key to continue...')
    sys.exit(1)

A = ('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', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9')


# converts Alphanumeric characters to numbers of base 36
def f(x):
  store = []
  for s in x:
    count = 0
    for i in range(36):
        if A[i].lower() == s.lower():
          store.append(i)
          count = 1
          break
    if count == 0:
      store.append(' ')
  return tuple(store)


# converts base 36 numbers to alphanumeric charactors.
def rf(x):
    store = []
    q = ''
    for s in x:
        try:
            store.append(A[s])
        except(IndexError, TypeError):
            store.append(' ')
    q = ''.join(store)
    return q

# generates a key without keyfile.
def ikey(x):
    seed = list(range(36))
    masterkey = []
    for i in range(len(x)):
        masterkey.append(secrets.choice(seed))
    return tuple(masterkey)


# encrypts a given string and returns ciphertxt and key as a tuple. (no file generated!)
def en(msg):
    ciphertxt = []
    x = f(msg)
    y = ikey(msg)
    for i in range(len(x)):
            if type(x[i]) is int:
                ciphertxt.append(((x[i]+y[i]) % 36))
            else:
                ciphertxt.append(' ')
    ctxt = rf(tuple(ciphertxt))
    shk = rf(y)
    return (ctxt, shk)


# decrypts a given encrypted string and returns a plaintxt as output.
def de(c, k):
    ciphertxt = []
    x = f(c)
    y = f(k)
    if len(x) <= len(y):
        for i in range(len(x)):
            if type(x[i]) is int and type(y[i]) is int:
                ciphertxt.append(((x[i]-y[i]) % 36))
            else:
                ciphertxt.append(' ')
    else:
        x = input('Incorrect Input!!!\nPress any key to continue...')
        sys.exit(1)
    return rf(tuple(ciphertxt))


# function for secret splitting interface.
def sprocess():
    table = []
    print('''\n---------------------------------------------------------
          |          Secret splitting                 |
          ---------------------------------------------------------''')
    while 1:
        try:
            x = int(input('\nEnter the number of shares(atmost 10,atleast 2):'))
            if 1 < x < 11:
                break
        except ValueError:
            print('\nPlease enter a valid integer greater than 1 but less than or equal to 10!\n')
    msg = getpass.getpass('Enter the secret:')
    table += list(en(msg))
    for i in range(2, x):
        tmp = table[-1]
        table.pop()
        table += list(en(tmp))
    for i in range(len(table)):
        print('SHARE', i+1, ':', table[i])


# function for secret combining interface.
def cprocess():
    table = []
    print('''\n         ---------------------------------------------------------
          |         Secret Combine                  |
        ---------------------------------------------------------''')
    while 1:
        try:
            x = int(input('\nEnter no. of shares to combine(atmost 10,atleast 2):'))
            if 1 < x < 11:
                break
        except ValueError:
                print('\nPlease enter a valid integer greater than 1 but less than or equal to 10!\n')
    for i in range(x):
            table.append(getpass.getpass(str('Enter Share '+str(i+1)+':')))
    for i in range(x-1):
            hook = []
            a, b = table[-2], table[-1]
            table.pop()
            table.pop()
            hook.append(de(a, b))
            table += hook
    print()
    print(''.join(table))

def Ref():
    print("Message= ", (Msg.get()))

    clear = Msg.get()
    algo=Algo.get
    k = key.get()
    m = mode.get()

    if (m == 'E' & algo=='A'):
        Result.set(sprocess(k, clear))
    else:
        Result.set(decode(k, clear))

# Show message button
btnTotal = Button(f1, padx = 16, pady = 8, bd = 16, fg = "black",
                        font = ('arial', 16, 'bold'), width = 10,
                       text = "Show Message", bg = "violet",
                         command = Ref).grid(row = 7, column = 1)

# Reset button
btnReset = Button(f1, padx = 16, pady = 8, bd = 16,
                  fg = "black", font = ('arial', 16, 'bold'),
                    width = 10, text = "Reset", bg = "green",
                   command = Reset).grid(row = 7, column = 2)

# Exit button
btnExit = Button(f1, padx = 16, pady = 8, bd = 16,
                 fg = "black", font = ('arial', 16, 'bold'),
                      width = 10, text = "Exit", bg = "red",
                  command = qExit).grid(row = 7, column = 3)
# keeps window alive
root.mainloop()
```

Flowchart



User enters the number for the suitable algorithm
If user chooses number 1
If user chooses number 2
If user chooses number 3
Is user chooses number 4
If user chooses number 5
Pattern based algorithm
ASCII based algorithm
Modified RSA Algorithm
Modified AES Algorithm
Modified DES Algorithm
If e selected
If d selected
Plaintext is encypted and ciphertext is given without a key
Ciphertext is decrypted and plaintext is given as output
Plaintext is encypted and ciphertext is given with more than one key specified by the user
Ciphertext is decrypted using the key which has to be entered in the same order and plaintext is given as output
Plaintext is encypted and ciphertext is given along with public key
Ciphertext is decrypted and plaintext is given as output using only the private key
Plaintext is encypted and ciphertext is given along with key
Ciphertext is decrypted using the same key and plaintext is given as output
Plaintext is encypted and ciphertext is given with a shared secret key
Ciphertext is decrypted and plaintext is given as output by inputing the secret key

REFERENCE:

1. Heidilyn V. Gamido, Marlon V. Gamido, Ariel M. Sison, Developing a secured image file management system using modified AES,2019.
2. V Gautham, MS Yashas, Image Encryption for secure internet Transfer, 2021.
3. Haidelyn, Ariel, Modified AES for text and Image encryption, 2018.
4. Priyanka, Rohan, Modification to AES algorithm using complex encryption, 2020.
5. Aloka Sinha*, Kehar Singh, A technique for image encryption using digital signature, 2003
6. Mahmoud Gad , Esam Hagras, Hasan Soliman , and Noha Hikal,A New Parallel Fuzzy Multi Modular Chaotic Logistic Map for Image Encryption,2021
7. Nivetha A,Preethy Mary S,Santosh kumar J,Modified RSA Encryption Algorithm using Four Keys,2015.
8. A hardware implementation of a modified DES algorithm by T. Kropf, W. Beller, T. Giesler, 2003
9. An implementation of DES and AES, secure against attacks by Mehdi-Laurent Akkar, Christophe Giraud, 2001
10. An analysis of NewDES: a modified version of DES by Charles Connell, 1990

ALL CODES LINK:
https://colab.research.google.com/drive/1VDO7g9zJj0n_h0D3-qh67yxL_7a7yAQS?usp=sharing

CONTRIBUTIONS

Name	Registration Number	Contributions
Deepa S	**20BCI0116**	**Modified DES, GUI, Literature Survey**
Evangeline S	**20BCI0274**	**Modified AES, GUI, Literature Survey**
Akshat Pattiwar	**20BCI0258**	**Modified RSA, flowchart, Pattern and ASCII based Algorithm**