BLM442E Homework

Yasemin Maya Kara

1521221114

6/04/2020

Introduction

in this homework there are some implementation of some famous algorithms in computer security field like RSA that used to generate public private pairs and digital signature also it consist a comparison between AES and DES encryption algorithms with different key sizes

in each question I research the theory behind the algorithm and the most popular way to implement it and then fix the codes so it meets our requirements

codes are written in simple pure python using Crypto package in some questions.

Q1) Generate an RSA public-private key pair. K A+ and K A

The public key , as its name implies, is public and open to anyone in the system. The public key is used to encrypt data.

The private key however is private. It is only ever stored on user's device. The private key is used to decrypt data.

Public key is used to convert the message to an unreadable form. Private key is used to convert the received message back to the original message. Both these keys help to ensure the security of the exchanged data. A message encrypted with the public key cannot be decrypted without using the corresponding private key.

RSA (Rivest–Shamir–Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. Algorithm flow is shown below.

1- choose two large prime numbers p, q.

2-compute n = pq, z = (p-1)(q-1)

3-choose e (with e<n) that has no common factors

with z (e, z are "relatively prime").

4-choose d such that ed-1 is exactly divisible by z.

(in other words: ed mod z = 1).

5-public key is (n,e). private key is (n,d)

```
# in real word primes must be very very big
primes =list(filter(lambda x: is_prime(x),list(range(100, 199))))
p = random.choice(primes)
#to prevent seleccting same prime we delete it from the list
primes.remove(p)
q=random.choice(primes)
public, private = generate_keypair(p, q)
print("p =",p)
print("q =",q)
print ("Your public key is ", public ," and your private key is ", private)
```

```
e = 29427
d = 28327
p = 179
q = 167
Your public key is  (29427, 29893)  and your private key is  (28327, 29893)
```

2) Generate two symmetric keys: 128 bit K 1 and 256 bit K 2 . Print values of the keys on the

screen. Encypt them with K A+ , print the results, and then decrypt them with K A- . Again print the results. Provide a screenshot showing your results.

In the image below symmetric keys are generated in tow way the first one using rand range and the next one using get random bytes function both ways generating keys with required limits

```
k1=random.randrange(2**127, 2**128)
k2=random.randrange(2**255, 2**256)
print("k1 (",k1.bit_length()," bit) = ",k1)
print("k2 (",k2.bit_length()," bit) = ",k2)
```

```
k1 ( 128  bit) =  331549013627491670979645017840482185021
k2 ( 256  bit) =  967807731514915264247105546222291474450636441885596140197897049768257242801 30
```

```
from Crypto.Random import get_random_bytes
k1=get_random_bytes(16)
k2=get_random_bytes(32)
print("k1  = ",k1)
print("k2  = ",k2)
```

```
k1  =  b'\xe6\xa1\x85\xecz\xf7m\xffsP,\x19\xf2\x97D\xda'
k2  =  b'\x9f\r\x8c\x1d\xd6\xa8\xd8b\xf3\x02\xb0\x94=\xc9Y\xc8\xd9+\xae\xcc\xa9~\x11I\x07\x80c\xca\x16\x12\xc8i'
```

Encrypting k1 with public key and decrypting it using private key

```
encrypted_msg = encrypt(public, str(k1))
print ("Your encrypted k1 is: ")
print (''.join(map(lambda x: str(x), encrypted_msg)))
print ("Decrypting k1 with private key ", private )
print ("Your k1 is:")
print (decrypt( private, encrypted_msg))
```

```
Your encrypted k1 is:
14178216815053259901330315889505325990129746078505325990877783750532599013303186441907150532599097419064212145053259909974974269801447113715505325990607824247505325990974198225053259902424719064974650532599091451297421681
Decrypting k1 with private key  (28327, 29893)
Your k1 is:
b'\xe6\xa1\x85\xecz\xf7m\xffsP,\x19\xf2\x97D\xda'
```

Encrypting k2 with public key and decrypting it

```
encrypted_msg = encrypt(public, str(k2))
print ("Your encrypted k2 is: ")
print (''.join(map(lambda x: str(x), encrypted_msg)))
print ("Decrypting k2 with private key ", private )

print ("Your k2 is:")
print (decrypt(private , encrypted_msg))
```

```
Your encrypted k2 is:
14178216815053259902424797450531604450532599087718644505325990607891455053259909145158895053259901297487750532599091458771417850532599097416180505325990244541982250532599014178244545053259902424711430169555053259901864424247180265053259901864487750532599091452424712900505325990129741330350532599018644186445053259901297424247200875053259906078607823649505325990244541906450532599087724454186445053259901864412974505325990607815889505325990607819822505325990186448775845843215 81
Decrypting k2 with private key  (28327, 29893)
Your k2 is:
b'\x9f\r\x8c\x1d\xd6\xa8\xd8b\xf3\x02\xb0\x94=\xc9Y\xc8\xd9+\xae\xcc\xa9~\x11I\x07\x80c\xca\x16\x12\xc8i'
```

3) Consider a long text m. Apply SHA256 Hash algorithm (Obtain the message digest, H(m)). Then encrypt it with K A- . (Thus generate a digital signature.) Then verify the digital signature. (Decrypt it with K A+ , apply Hash algorithm to the message, compare). Print m, H(m) and digital signature on the screen. Provide a screenshot. (Or you may print in a file and provide the file).

Digital signature used for verifying authentication and message integrity.

for creating digital signature we need hashed message and then to encrypt it with private key

digital signature= (message, K A- (h(message)))

```
from hashlib import sha256
def hashFunction(message):
    hashed = sha256(message.encode("UTF-8")).hexdigest()
    return hashed
message="this secret  message is writen by maya kara is here"
hMsg= hashFunction(message)
encryptedHMsg=encrypt(private, hMsg)
print ("- your message is: ")
print(message)
print ("- Your encrypted H(message) is : ")
print (''.join(map(lambda x: str(x), encryptedHMsg)))
```

```
- your message is:
this secret  message is writen by maya kara is here
- Your encrypted H(message) is :
579520702986210502365412664298612664255361882221050103145795207302073025536163531744017440579532462986579545382070207023654579518822298657952070126641244117440163531244129861266465833246207057951744012664255362553620702070658312664658316353658365831244118822126641882210314207017440210501244 1
```

for Verifying the digital signature we need to

1-Decrypt the encrypted H(m) in signature with K A+ to obtain H(m)

2-get H(m) of the pure message in signature

3- compare bothe H(m) values if they are equal the signature is valid

```
print ("First  H(m) value: ",decrypt(public , encryptedHMsg))
print ("Second H(m) value: ",hashFunction(message))
```

```
First  H(m) value:  fb97481491193dffb3fb4620c30fe2f2c0dfb07b63511e88d37f8000e2dcea52
Second H(m) value:  fb97481491193dffb3fb4620c30fe2f2c0dfb07b63511e88d37f8000e2dcea52
```

4) Generate or find any file of size 1MB. Now consider following three algorithms:

i) AES (128 bit key) in CBC mode.

ii) AES (256 bit key) in CBC mode.

iii) DES in CBC mode (you need to generate a 56 bit key for this).

For each of the above algorithms, do the following:

a) Encrypt the file of size 1MB. Store the result (and submit it with the homework) (Note: IV should be randomly generated, Key = K 1 or K 2).

b) Decrypt the file and store the result. Show that it is the same as the original file.

c) Measure the time elapsed for encryption. Write it in your report. Comment on the result.

d) For the first algorithm, change Initialization Vector (IV) and show that the corresponding ciphertext chages for the same plaintext (Give the result for both).

You need to do this homework in your own. No groups are allowed.

AES algorithm flow likewise:

- Add round key
- Substitute bytes
- Shift rows
- Mix columns
- Add round key

in code below algorithm details is implemented in Crypto package

encrypt file function take file name and encrypt contents and store it in file itself

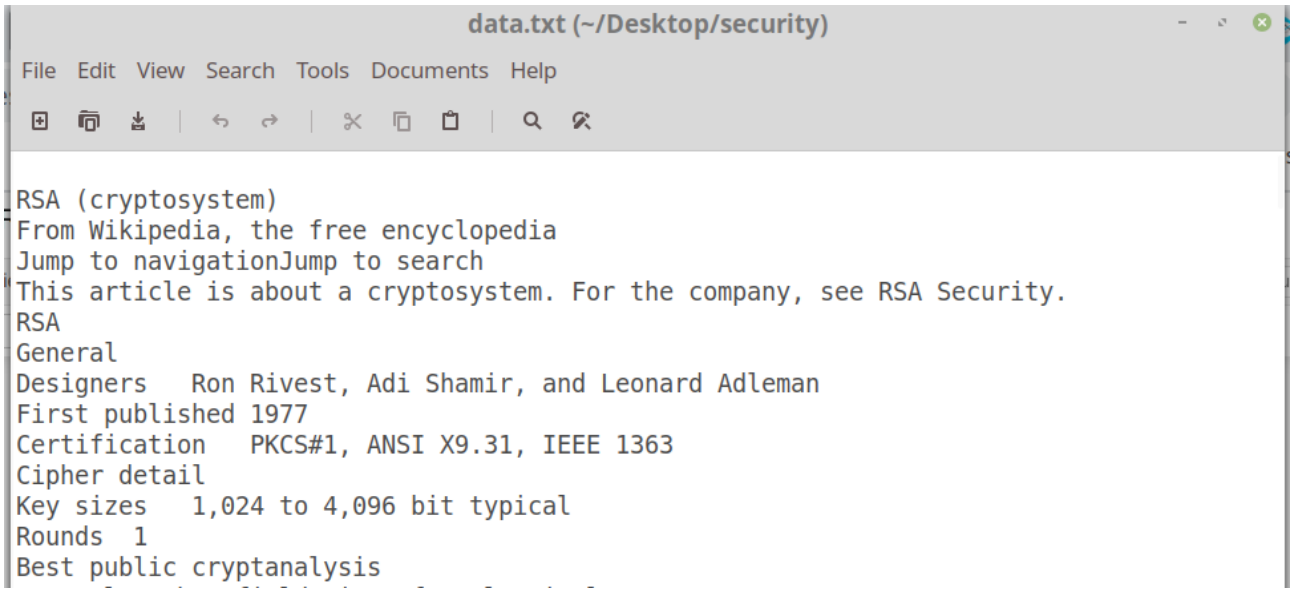
decrypt file function take file name and decrypt content and store it back to same file

```
from Crypto import Random
from Crypto.Cipher import AES
import os
import os.path
from os import listdir
from os.path import isfile, join
import time


class Encryptor:
    def __init__(self, key):
        self.key = key
    # for soving not completed block in file end problem
    def pad(self, s):
        return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

    def encrypt(self, message, key, key_size=128):
        message = self.pad(message)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(key, AES.MODE_CBC, iv)
        return iv + cipher.encrypt(message)

    def encrypt_file(self, file_name):
        with open(file_name, 'rb') as fo:
            plaintext = fo.read()
        enc = self.encrypt(plaintext, self.key)
        with open(file_name + ".enc", 'wb') as fo:
            fo.write(enc)
        os.remove(file_name)
        return enc
```

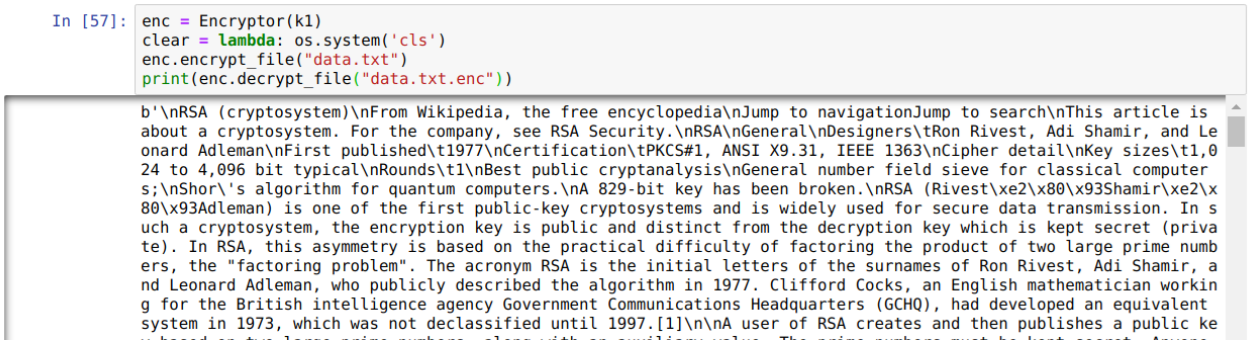
```
def decrypt(self, ciphertext, key):
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext[AES.block_size:])
    return plaintext.rstrip(b"\0")

def decrypt_file(self, file_name):
    with open(file_name, 'rb') as fo:
        ciphertext = fo.read()
    dec = self.decrypt(ciphertext, self.key)
    with open(file_name[:-4], 'wb') as fo:
        fo.write(dec)
    os.remove(file_name)
    return dec
```

for testing algorithm data.txt file is added to project package

data.txt starts with some information about RSA from Wikipedia and ends with some random values as below.



in code below data.txt is encrypted using AES algorithm with k1 and then decrypted and printed. By comparing original file and text in the image below we can find that file is successfully decrypted. Same result is taken with encryption with k2



in this step time is majored for both k1 and k2 results shows that encryption and decryption using k1 is faster that k2 although in this example results are almost same with bigger files difference between results will be bigger.

another point is that in theory it is known that in AES algorithm encryption is faster than decryption however in this example the opposite is obtained so probably encryption is not paralleled in Crypto implementation

```
start_time = time.time()
enc = Encryptor(k1)
clear = lambda: os.system('cls')
enc.encrypt_file("data.txt")
print("encryption time with k1 ( 128  bit) =  %s seconds" % (time.time() - start_time))

start_time = time.time()
enc.decrypt_file("data.txt.enc")
print("decryption time with k1 ( 128  bit) =  %s seconds" % (time.time() - start_time))
```

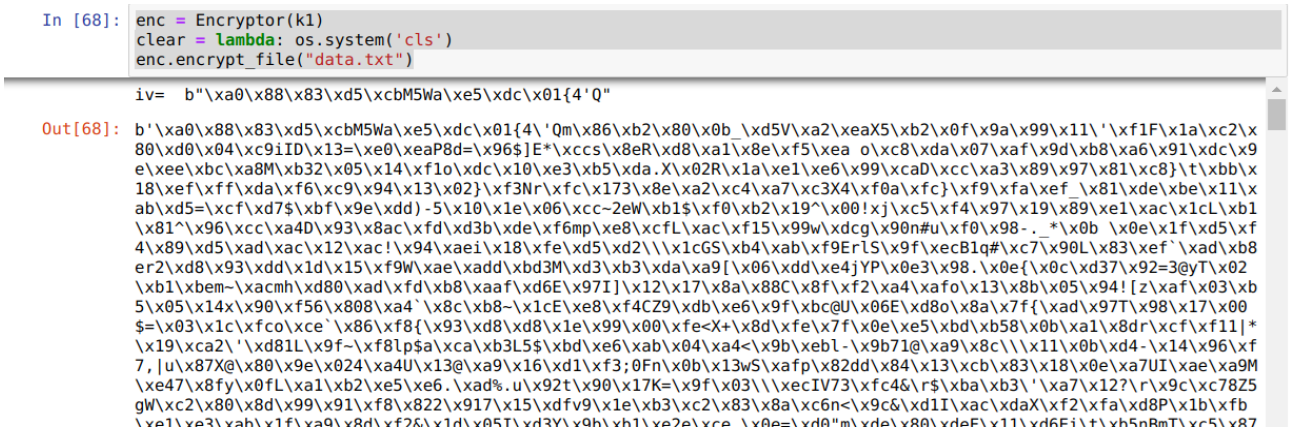
```
encryption time with k1 ( 128  bit) =  0.013556480407714844 seconds
decryption time with k1 ( 128  bit) =  0.00768733024597168 seconds
```

```
start_time = time.time()
enc = Encryptor(k2)
clear = lambda: os.system('cls')
enc.encrypt_file("data.txt")
print("encryption time with k2 ( 256  bit) =  %s seconds" % (time.time() - start_time))

start_time = time.time()
enc.decrypt_file("data.txt.enc")
print("decryption time with k2 ( 256  bit) =  %s seconds" % (time.time() - start_time))
```

```
encryption time with k2 ( 256  bit) =  0.01474905014038086 seconds
decryption time with k2 ( 256  bit) =  0.009457826614379883 seconds
```

when ever we run encryption code iv will be initialized random and generate different cipher text

images below show different generated iv and cipher text when we run code for tow times.



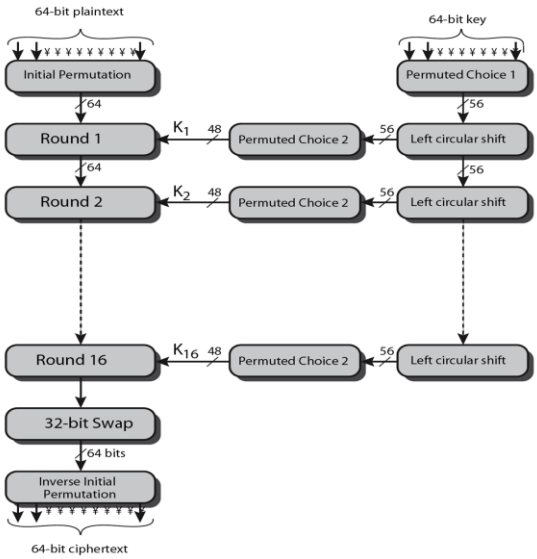
```
In [68]: enc = Encryptor(k1)
         clear = lambda: os.system('cls')
         enc.encrypt_file("data.txt")

         iv=  b"\xa0\x88\x83\xd5\xcbM5Wa\xe5\xdc\x01{4'Q"

Out[68]: b'\xa0\x88\x83\xd5\xcbM5Wa\xe5\xdc\x01{4\'Qm\x86\xb2\x80\x0b_\xd5V\xa2\xeaX5\xb2\x0f\x9a\x99\x11\'\xf1F\x1a\xc2\x
         80\xd0\x04\xc9iID\x13=\xe0\xeaP8d=\x96$]E*\xccs\x8eR\xd8\xa1\x8e\xf5\xea o\xc8\xda\x07\xaf\x9d\xb8\xa6\x91\xdc\x9
         e\xee\xbc\xa8M\xb32\x05\x14\xf1o\xdc\x10\xe3\xb5\xda.X\x02R\x1a\xe1\xe6\x99\xcaD\xcc\xa3\x89\x97\x81\xc8}\t\xbb\x
         18\xef\xff\xda\xf6\xc9\x94\x13\x02}\xf3Nr\xfc\x173\x8e\xa2\xc4\xa7\xc3X4\xf0a\xfc}\xf9\xfa\xef_\x81\xde\xbe\x11\x
         ab\xd5=\xcf\xd7$\xbf\x9e\xdd)-5\x10\x1e\x06\xcc~2eW\xb1$\xf0\xb2\x19^\x00!xj\xc5\xf4\x97\x19\x89\xe1\xac\x1cL\xb1
         \x81^\x96\xcc\xa4D\x93\x8ac\xfd\xd3b\xde\xf6mp\xe8\xcfL\xac\xf15\x99w\xdcg\x90n#u\xf0\x98-._*\x0b \x0e\x1f\xd5\xf
         4\x89\xd5\xad\xac\x12\xac!\x94\xaei\x18\xfe\xd5\xd2\\\x1cGS\xb4\xab\xf9ErlS\x9f\xecB1q#\xc7\x90L\x83\xef`\xad\xb8
         er2\xd8\x93\xdd\x1d\x15\xf9W\xae\xadd\xbd3M\xd3\xb3\xda\xa9[\x06\xdd\xe4jYP\x0e3\x98.\x0e{\x0c\xd37\x92=3@yT\x02
         \xb1\xbem~\xacmh\xd80\xad\xfd\xb8\xaaf\xd6E\x97I]\x12\x17\x8a\x88C\x8f\xf2\xa4\xafo\x13\x8b\x05\x94![z\xaf\x03\xb
         5\x05\x14x\x90\xf56\x808\xa4`\x8c\xb8~\x1cE\xe8\xf4CZ9\xdb\xe6\x9f\xbc@U\x06E\xd8o\x8a\x7f{\xad\x97T\x98\x17\x00
         $=\x03\x1c\xfco\xce`\x86\xf8{\x93\xd8\xd8\x1e\x99\x00\xfe<X+\x8d\xfe\x7f\x0e\xe5\xbd\xb58\x0b\xa1\x8dr\xcf\xf11|*
         \x19\xca2\'\xd81L\x9f~\xf8lp$a\xca\xb3L5$\xbd\xe6\xab\x04\xa4<\x9b\xebl-\x9b71@\xa9\x8c\\\x11\x0b\xd4-\x14\x96\xf
         7,|u\x87X@\x80\x9e\x024\xa4U\x13@\xa9\x16\xd1\xf3;0Fn\x0b\x13wS\xafp\x82dd\x84\x13\xcb\x83\x18\x0e\xa7UI\xae\xa9M
         \xe47\x8fy\x0fL\xa1\xb2\xe5\xe6.\xad%.u\x92t\x90\x17K=\x9f\x03\\\xecIV73\xfc4&\r$\xba\xb3\'\xa7\x12?\r\x9c\xc78Z5
         gW\xc2\x80\x8d\x99\x91\xf8\x822\x917\x15\xdfv9\x1e\xb3\xc2\x83\x8a\xc6n<\x9c&\xd1I\xac\xdaX\xf2\xfa\xd8P\x1b\xfb
```

```
In [70]: enc = Encryptor(k1)
         clear = lambda: os.system('cls')
         enc.encrypt_file("data.txt")
```

```
iv=  b'e\x89RnA\x98jz8\xb0\x8e\xa7C\xf1)\x91'

Out[70]: b'e\x89RnA\x98jz8\xb0\x8e\xa7C\xf1)\x91\xee\x8bP\xde_\xa1`T+\x81\x8b^\xac\x81\x96\xa9\x0f\xb6\x06\x1e\x91o\xa7\x1
         8\x12\x8e\xec\\1Dn\x02\x0b\xbb\xf5\xd5\xf1I\xfb\xee9\xf4\x18\x8a\x02aq\xefE\xcbk\xb6|\xa1\x9c\xf5i`P\x9e\xc5\x81
         \xb2Q\xf0\xf8o\x1d\x98\xfc\xfa\x1dh\x12\xd9\x06DI\xf9\x9b\xfa\x98\x98juy\x80\xca\xaa""\x92\xab\x9b\x11@\xd7\x94r!
         \x06N\x94]\xd3\xd1.b\xfd\x805\xa2\x84v\nx\xf2\xf9,\xc9\x12k0\x10\x0c%\x00\x85\x86\x1e$\xf6\x96\x10XW\x01\xc1\xb3
         $\x15l\xad\xc5Ri\xc2\x97\x9f\t&\xdf\xf6He\x16\xe4>\xba\x8b\xb2B\xd9\xce(\x95t\xe8Z\x89\x1f|H\x8d0\r\xfa\xbes\x08
         \xec\x1b\xab\x1er\x95FG\r~\xdb\x99<#R\x14Vg\xae\x08\r\x0c\xd4\x9d6\x99\x98\'\xd6p\x12]\xd4mC\xbc-}\x92\xc5\xebM
         $\xbbg\x01\x1a\xfe^\xb1;\xd2_8\x172|e\xf7C9\n\x0b,_\x91\x90c\xac\xabu\x06~u\xaa\x97\xf0j\xfa\x0bCR\xaf\x14\xc4\x1
         8\xc8\x92w2\xa4G\xe6l\r\x0b\x07\xf2\xfc\x16;\xe3t\xd0\x0b+\xe3\x1bp\tXEG\x13\x88\x9d\xf9\xa9\xfe\xdf:\xae\xf7<\x9
         2\x90\xafY\xd9\x82s{\xe3\x04`RR\x14\xd7s!$\x83=cH\xf4\xa8\x8d\xa0z\xa4\xacj?\xc9\xc0\x02\xa0w*\x04\xd6\x80\xfd\xf
         3\x9a\x81\xea\xf9\xfc\xf4=\xd1\xf0l\x18q\xa0:\x9f\xd5\x9e\x90\x85\tx\x8bi\x92`SK\xf7Y}\xa3\x91\x07V\xf6T\xaf\xd3
         \xfb\xc9\xb2\xf8\x1fI\x89\x00i\xdf\xcd\xa98\xca>\x17\xc6\xe5\xa9W\xb6\xbb\xb3\x9c\xf8\xf0h\xf7kNr\xa3\x9a=Y\xfb\x
         b0Tkm\xf5\x1dqZ1\x89\x1a\xb9:\x9d}\xa8IXr-\x88\xd9\x04\x05\xb765\x14\x99$:,\x9cf\xcf\xaa\x05zD\x03iC\xe1\xfb\xd5
         \xc4\xbfje\x1a\xfc\xb1\xbfLm\xf1\x19\xe9\xf1c\x91\x91\xb3\xfe\xaa\x9c\x84\xdb9\xaa\xe2V\x97}S\xb3\xc7\x87\\{Mw=\x
         9fx\xf3\xe7\n\x92\x81=6b\xcc\x92b\xc6\x82\xf2\xcbp\xfeN\x96\xe8\x92\xc7\xf7W\nb\x8e\xf8\xd3\xf9td\xcb\xb6}-Eb\xea
         \x1d\x13\x8c!s\x91\xe5\x9cG\xf9\x10\xc6\xf2\xaai[\xf8\xaeb\x92\xd6\xf01i\xe9\xe5G\x9fAi\xd7D\x9dG\xe2\x7f\x16g\x8
         c\xf3\x95\x9b\x83\xeb\xa8F\x7f\x9f\xac\xa1F\x7fQ\x1e\xc9\x0bV\x1bAN\xde9%\x86,H\x1b0`\x80#*\x7fp\xc2)\x8d\xb5\x9f
         \x04\xdb\x83\xdf\xc9\xd4\xd1\xff\xb5dP\x0b\xfe\x8ay\x8a\xcb\x8e\xe1a#\xb2&\xe0\x85D\xc9E\xd6JP\xa1\xf6\x1e\xf08\x
```

DES algorthim

flow of DES algorithm is shown in image below the most useful propriety of DES that encryption and decryption are done using same function.



```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

class DESencryptor:
    def __init__(self, key):
        self.key = key
        self.des =   DES.new(self.key, DES.MODE_ECB)


    def encrypt(self, message):
        message = pad(message,8)

        return self.des.encrypt(message)

    def encrypt_file(self, file_name):
        with open(file_name, 'rb') as fo:
            plaintext = fo.read()
        enc = self.encrypt(plaintext)
        with open(file_name + ".enc", 'wb') as fo:
            fo.write(enc)
        os.remove(file_name)
        return enc
    def decrypt(self, ciphertext):
        return unpad(self.des.decrypt(ciphertext),8)

    def decrypt_file(self, file_name):
        with open(file_name, 'rb') as fo:
            ciphertext = fo.read()
        dec = self.decrypt(ciphertext)
        with open(file_name[:-4], 'wb') as fo:
            fo.write(dec)
        os.remove(file_name)
        return dec
```

algorithm also tested with same text file and 8 byte random generated key

```
In [61]: key = get_random_bytes(8)
desEnc = DESencryptor(key)
clear = lambda: os.system('cls')
desEnc.encrypt_file("data.txt")
print(desEnc.decrypt_file("data.txt.enc"))
```

```
b'\nRSA (cryptosystem)\nFrom Wikipedia, the free encyclopedia\nJump to navigationJump to search\nThis article is
about a cryptosystem. For the company, see RSA Security.\nRSA\nGeneral\nDesigners\tRon Rivest, Adi Shamir, and Le
onard Adleman\nFirst published\t1977\nCertification\tPKCS#1, ANSI X9.31, IEEE 1363\nCipher detail\nKey sizes\t1,0
24 to 4,096 bit typical\nRounds\t1\nBest public cryptanalysis\nGeneral number field sieve for classical computer
s;\nShor\'s algorithm for quantum computers.\nA 829-bit key has been broken.\nRSA (Rivest\xe2\x80\x93Shamir\xe2\x
80\x93Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. In s
uch a cryptosystem, the encryption key is public and distinct from the decryption key which is kept secret (priva
te). In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numb
ers, the "factoring problem". The acronym RSA is the initial letters of the surnames of Ron Rivest, Adi Shamir, a
nd Leonard Adleman, who publicly described the algorithm in 1977. Clifford Cocks, an English mathematician workin
g for the British intelligence agency Government Communications Headquarters (GCHQ), had developed an equivalent
system in 1973, which was not declassified until 1997.[1]\n\nA user of RSA creates and then publishes a public ke
y based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone
```

Results obtained by majoring time passed to encrypt and decrypt same file in DES is shown below. Because DES use same function for both operation it is very normal to have encryption time that almost equal to decryption time.

```
key = get_random_bytes(8)

start_time = time.time()
desEnc = DESencryptor(key)
clear = lambda: os.system('cls')
desEnc.encrypt_file("data2.txt")
print("encryption time = %s seconds ---" % (time.time() - start_time))
start_time = time.time()
desEnc.decrypt_file("data2.txt.enc")
print("decryption time = %s seconds ---" % (time.time() - start_time))
```

```
encryption time = 0.0027666091918945312 seconds ---
decryption time = 0.003014802932739258 seconds ---
```