T.C.

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

COMPUTER SECURITY

HOMEWORK 1

Beyza Koşer	1521221046

1.Generate an RSA public-private key pair. K_A^+ and K_A^-.

Öncelikle hazır kütüphaneler kullanarak 1024 bitlik public ve private key generate ettim.

```
1]: #1)public private key
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

keyPair = RSA.generate(1024)
pubKey = keyPair.publickey()
privKeyPEM = keyPair.exportKey()

n = hex(pubKey.n)
e=hex(pubKey.e)
d=hex(keyPair.d)

print("Public key:  (n=",n, ",e=",e,")")
print("Private key:  (n=",n, ",d=",d,")")
```

```
Public key:  (n= 0xa0bcbf4c1315147b99ff92403c563897817c627b17b90410e8c0134519a5a29922898a17ebc5f271905c3443f1466fa16289edc1ad25
f91f788fb06e6b36799023fcdddcf227ee25f0c0ec640728dfb83cebb05a0d9458c14dbb552efe910c15e41fb323553064e1aac042bca1f0bdaae00e6a74ce3
2f636f066e5943b4eee37 ,e= 0x10001 )
Private key:  (n= 0xa0bcbf4c1315147b99ff92403c563897817c627b17b90410e8c0134519a5a29922898a17ebc5f271905c3443f1466fa16289edc1ad2
5f91f788fb06e6b36799023fcdddcf227ee25f0c0ec640728dfb83cebb05a0d9458c14dbb552efe910c15e41fb323553064e1aac042bca1f0bdaae00e6a74ce
32f636f066e5943b4eee37 ,d= 0x4f1a141a6019b3a6d03691c6c2eabb96f3c63ec0ccffe41c5e2e884d2d3df232e2684569584265f9b54d6afb34b5b121fe
d85a05422cb9d8defff7979f738386bf45084276d540469f1970dbabda49b727e1bd7213db01b7d22be7e2b954566e310d455ac44309e85114356def9ca4885
08aa3cc2e6a4f691fed76a451c81d41 )
```

2. Generate two symmetric keys: 128 bit K1 and 256 bit K2. 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.

Birisi 128 bit diğeri 256 bitlik olacak şekilde simetrik key oluşturdum.

```
In [2]: #2)k1=128 bit
# k2=256 bit
import random
def generate_Ks(keySize):
    ks= random.randrange(2**(keySize-1),2**keySize)
    return ks
k1 = generate_Ks(128)
print(k1.bit_length()," bit = k1 =",k1)
k2 = generate_Ks(256)
print(k2.bit_length()," bit = k2 =",k2)
```

```
128  bit = k1 = 311671775134014134082216306687778740634
256  bit = k2 = 609087494493830297701615717335114537995303999123033938561423459554601853062 66
```

Yine hazır kütüphaneler kullanarak encryption() ve decryption() metodlarını oluşturdum**.** Decrypt edildiğinde aynı sonucu verdiğini göstermek için ekrana yazdırdım.

```
In [3]: def encryption(pubKey,msg):
    encryptor = PKCS1_OAEP.new(pubKey)
    encrypted = encryptor.encrypt(msg)
    return encrypted

def decryption(private,encrypted):
    decryptor = PKCS1_OAEP.new(private)
    decrypted = decryptor.decrypt(encrypted)
    return decrypted
```

```
In [4]: print("k1 =",k1)
k1_encryption=encryption(pubKey,str.encode(str(k1)))
print("k1_encryption= ",k1_encryption)
print("k1_decryption= ",decryption(keyPair,k1_encryption))
```

```
k1 = 311671775134014134082216306687778740634
k1_encryption=  b'\x82K\xadP\xb4\xe9\xb8\x0c>\xe9\xbf\xb34\x80\x93\x16k4\xf0\xf4\x9d*\xa3\xf6\xca\rF\x1bA\xff\xc4\x90\x01\xe
7:\xb4#|\xf4\xe5\xdaa\xb6\x91\x06\xb9\xea\x1c\xd4\xbc\xa9\xe7_\x06\x0c\xe4\xb3NmoF\xa8\xd8`\xe6%B\x1f%D\x00\x87b\x84@\xacH.\x08
0\x01l\x1dD\x0b1\x7f\x03\xaaVm\x8b,\xa9@\xa7\xb0\x15\xf6X\xd6\xb5e\x99sT\xeb JT\xd5\xce\xef\xca\xc0g\xd0\x9f\xa9\x8d6E\x12\xc8
\x1e\x1b\x19\xb1'
k1_decryption=  b'311671775134014134082216306687778740634'
```

```
In [5]: print("k2 =",k2)
        k2_encryption=encryption(pubKey,str.encode(str(k2)))
        print("k2_encryption= ",k2_encryption)
        print("k2_decryption= ",decryption(keyPair,k2_encryption))
```

```
k2 = 609087494493830297701615717335114537995303999123033938561423459554601853062 66
k2_encryption=  b'@\x85\x1d\x8a7sD\xcb;\xed9\xed\xff\x19\x8b\x8d\xf5\xeff\xf2\x7f\xd7m\x91\x9e\x7fm\xf5\x83\x01{\xd6O(Ww\x88\xc
5\xd4\xfc\xeb\xaeq\xd5\x95\x97\xa5\xbd\xfc\xf9\n\x80\xed\xc3]Hi\xea;\xee\xf8\xd4\x96U\xabl\xeb\xd0Q\x8d}\xb8Y6\x9f\x0e\xa1\x17z
\xf9,\x88\xdd\xeal9\xe6M\x12\xe8\x0c\xc7\xb2U\xa5\x92\xb7A\xe6m\xb8\x8eF7\x00\x10\xa9U\xcf\x9c\x85lA\xe1\x1b\x15$\xf0yQ|\xd2\xc
2\r \xd8\xa8\x90'
k2_decryption=  b'609087494493830297701615717335114537995303999123033938561423459554601853062 66'
```

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 digitalsignature on the screen. Provide a screenshot. (Or you may print in a file and provide the file).

```
In [35]: import hashlib, binascii

         def hashAlgorithm(msg):
             sha256hash = hashlib.sha256(msg).digest()
             return sha256hash
```

```
In [45]: message=b"Consider a long text m. Apply SHA256 Hash algorithm"
         hash_message=hashAlgorithm(message)
         print("message = ",message)
         print("hash_message = ",hash_message)
         encrypted_hash=encryption(keyPair,hash_message)
         print("decrypted    = ",decryption(pubKey,encrypted_hash))
```

```
message =  b'Consider a long text m. Apply SHA256 Hash algorithm'
hash_message =  b"k\x1aO\xc7\x81\x8a}\xce\n\xc7\xa7#\xbe\xab\xe4s'\xe6\x9f\xa5x\xadb\xf7\xe5]\xd9!h\x01\x84\xb7"
decrypted    =  b"k\x1aO\xc7\x81\x8a}\xce\n\xc7\xa7#\xbe\xab\xe4s'\xe6\x9f\xa5x\xadb\xf7\xe5]\xd9!h\x01\x84\xb7"
```

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 = K1 or K2).

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).

1 Mb lık file.txt adında dosya oluşturdum. IV' u random generate ettim. İlk olarak 128 bitlik key size , sonrada 256 bitlik key size vererek metodları çağırdım. Hızlarını ekrana yazdırdım. Beklendiği gibi 128 bitlik key size daha hızlı oldu.

In [20]:
```
#4) AES
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

def pad(s):
    return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

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

def encrypt_file_AES(key,file_name,output_filename):
    with open(file_name, 'rb') as fo:
        plaintext = fo.read()
    enc = encrypt_AES(plaintext, key)
    with open(output_filename, 'wb') as fo:
        fo.write(enc)
    return enc

def decrypt_AES(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_AES(key,file_name):
    with open(file_name, 'rb') as fo:
        ciphertext = fo.read()
    dec =decrypt_AES(ciphertext,key)
    with open(file_name[:-4], 'wb') as fo:
        fo.write(dec)
    return dec
```

In [21]:
```
from Crypto.Random import get_random_bytes
k1=get_random_bytes(16)
print("k1 =" ,k1)
start_time = time.time()
encrypt_file_AES(k1,"file.txt","file_AES_128.txt")
finish_time=time.time()
seconds_AES_128_encryption=finish_time-start_time
print("(encryption) k1 = 128 bit= %s seconds" ,seconds_AES_128_encryption)

start_time = time.time()
decrypt_file_AES(k1,"file_AES_128.txt")
finish_time=time.time()
seconds_AES_128_decryption=finish_time-start_time
print("(decryption) k1 = 128 bit= %s seconds" , seconds_AES_128_decryption)

k2=get_random_bytes(32)
print("k2 =" ,k2)
start_time = time.time()
encrypt_file_AES(k2,"file.txt","file_AES_256.txt")
finish_time=time.time()
seconds_AES_256_encryption=finish_time-start_time
print("(encryption) k2 = 256 bit= %s seconds" ,seconds_AES_256_encryption)

start_time = time.time()
decrypt_file_AES(k2,"file_AES_256.txt")
finish_time=time.time()
seconds_AES_256_decryption=finish_time-start_time
print("(decryption) k2 = 256 bit= %s seconds" , seconds_AES_256_decryption)
```

```
k1 = b'\xa4\xd1a\xfdU\rl\tA\xe6\xf4v\xfdw\xe9N'
iv=  b'\x1c\x83\xfft\x96\xfd\x1c\x82\xe6\xebA\x96E1\x0f\xff'
(encryption) k1 = 128 bit= %s seconds 0.013008832931518555
(decryption) k1 = 128 bit= %s seconds 0.014916658401489258
k2 = b'l1\x12\xf8\x15@\xfas_\xb8,\xe7\xa93\x15\xaahwL\x88D\xdd\x98\x8b\xcf\xafGh\x92\x9bf\xe8'
iv=  b'\xc3\x91b\xa36\xbd.\x7f\x01r\xdd\xd7\x0e\x14~\x82'
(encryption) k2 = 256 bit= %s seconds 0.01396036148071289
(decryption) k2 = 256 bit= %s seconds 0.01595616340637207
```

DES algoritmasında 56 bitlik olması istenmişti fakat 8 byte dan aşağısını kabul etmediğinden 7 vermek zorunda kaldım. Yine beklediğim gibi key size ı daha küçük olmasına rağmen en yavaş çalışan algoritma bu oldu.

```
In [23]: k3 = get_random_bytes(8)#7 yi kabul etmedi 8 verdim
         start_time = time.time()

         encrypt_file_DES(k3,"file.txt","file_des.txt")
         finish_time=time.time()
         seconds_DES_encryption=finish_time-start_time

         print("(encryption) k3 = 56 bit= %s seconds" ,seconds_DES_encryption)

         start_time = time.time()
         decrypt_file_DES(k3,"file_des.txt")
         finish_time=time.time()
         seconds_DES_decryption=finish_time-start_time
         print("(decryption) k3 = 56 bit= %s seconds" , seconds_DES_decryption)

         (encryption) k3 = 56 bit= %s seconds 0.01894974708557129
         (decryption) k3 = 56 bit= %s seconds 0.0219423770904541
```

Encrypted ettiğim file.txt nin boyutu değişmedi.

Kaynakça

https://www.youtube.com/watch?v=UB2VX4vNUa0

https://cryptobook.nakov.com/symmetric-key-ciphers/aes-encrypt-decrypt-examples