COMP9337: Securing Wireless Networks

GROUP SWN19-B

Name: Wenxun Peng ZID: z5195349

Name: Tianyi Jiang ZID: z5159471

Code:

Version: Python 3.7

3(DES):

```
from Crypto.Cipher import DES

from Crypto import Random

import sys

import time

try:

    iv_input = sys.argv[1]

    if len(iv_input) != 16:

        raise ValueError

    iv = bytes.fromhex(iv_input)                    # str -> bytes

    cbc_key_input = sys.argv[2]

    if len(cbc_key_input) != 16:

        raise ValueError

    cbc_key = bytes.fromhex(cbc_key_input)

except ValueError:

    print("The value of iv and key must be hexadecimal digits and the length must be 8 bytes!")

    sys.exit()

try:

    file_name = sys.argv[3]

    #file_name = 'test.txt'

    f = open(file_name,'rb')

    plain_text = f.read()

except:

    print("Wrong file name!")
    sys.exit()
plain_text_array = bytearray(plain_text)
del plain_text_array[-1]
plain_text = bytes(plain_text_array)
des1 = DES.new(cbc_key, DES.MODE_CBC, iv)
des2 = DES.new(cbc_key, DES.MODE_CBC, iv)
pad = 8 - len(plain_text) % 8
if pad != 8:
    for i in range(pad):
        plain_text = plain_text.__add__(b'\x00')
time_encrypt_start = time.time()
cipher_text = des1.encrypt(plain_text)
time_encrypt_end = time.time()
time_encrypt = time_encrypt_end - time_encrypt_start
print(f'encrypt time is {time_encrypt}s.')
file_output = sys.argv[4]
with open(file_output, 'wb') as f:
    f.write(cipher_text)
time_decrypt_start = time.time()
msg=des2.decrypt(cipher_text)
time_decrypt_end = time.time()
time_decrypt = time_decrypt_end - time_decrypt_start
print(f'decrypt time is {time_decrypt}s.')
```

4.c(AES):

```
from Crypto.Cipher import AES

from Crypto import Random

import sys

import time

def pad(text):

    while len(text) % 16!=0:

        text += ' '

    return text

cbc_key = Random.get_random_bytes(16)

print ('key', [x for x in cbc_key])

iv = Random.get_random_bytes(16)

aes1 = AES.new(cbc_key, AES.MODE_CBC, iv)

aes2 = AES.new(cbc_key, AES.MODE_CBC, iv)

#plain_text = 'hello world 1234' # <- 16 bytes

f=open(sys.argv[1])

plain_text=f.read()

plain_text=pad(plain_text)

print(plain_text)

start_encrypt=time.time()

cipher_text = aes1.encrypt(plain_text)

end_encrypt=time.time()

time_encrypt = end_encrypt-start_encrypt

print(cipher_text)

start_decrypt = time.time()

msg=aes2.decrypt(cipher_text)
```

```
msg=bytes.decode(msg)

end_decrypt=time.time()

time_decrypt = end_decrypt-start_decrypt

msg=msg.rstrip(' ')

msg=bytes(msg,encoding='utf-8')

print(msg)
```

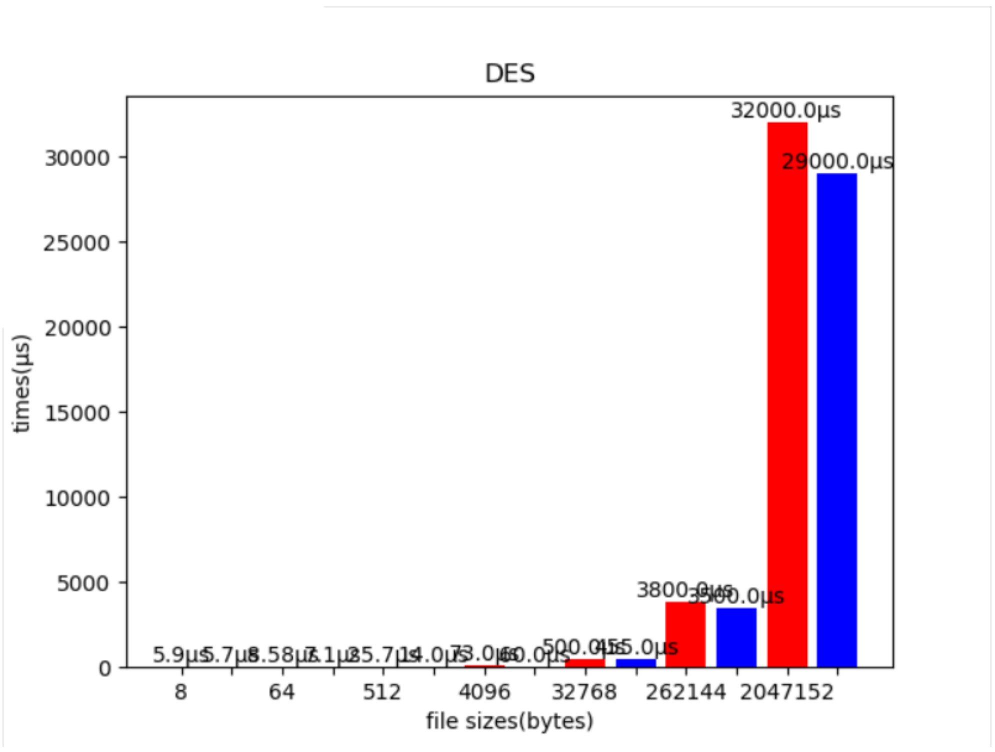
4.d(RSA):

```
import Crypto
from Crypto.PublicKey import RSA
from Crypto import Random
import ast
import sys
import time

def pad(text):
    while len(text) % 32!=0:
        text += ' '
    return text
random_generator = Random.new().read
key = RSA.generate(1024, random_generator) #generate pub and priv key

publickey = key.publickey() # pub key export for exchange
f=open(sys.argv[1])
plain_text=f.read()
plain_text=pad(plain_text)
plain_text=bytes(plain_text,encoding='utf-8')
start_encrypt=time.time()
encrypted = publickey.encrypt(plain_text, 32)
end_encrypt=time.time()
encrypt_time = end_encrypt - start_encrypt
print(f'encrypt time is {encrypt_time}s')
#message to encrypt is in the above line 'encrypt this message'
print ('encrypted message:', encrypted) #ciphertext

#decrypted code below
start_decrypt=time.time()
decrypted = key.decrypt(ast.literal_eval(str(encrypted)))
```

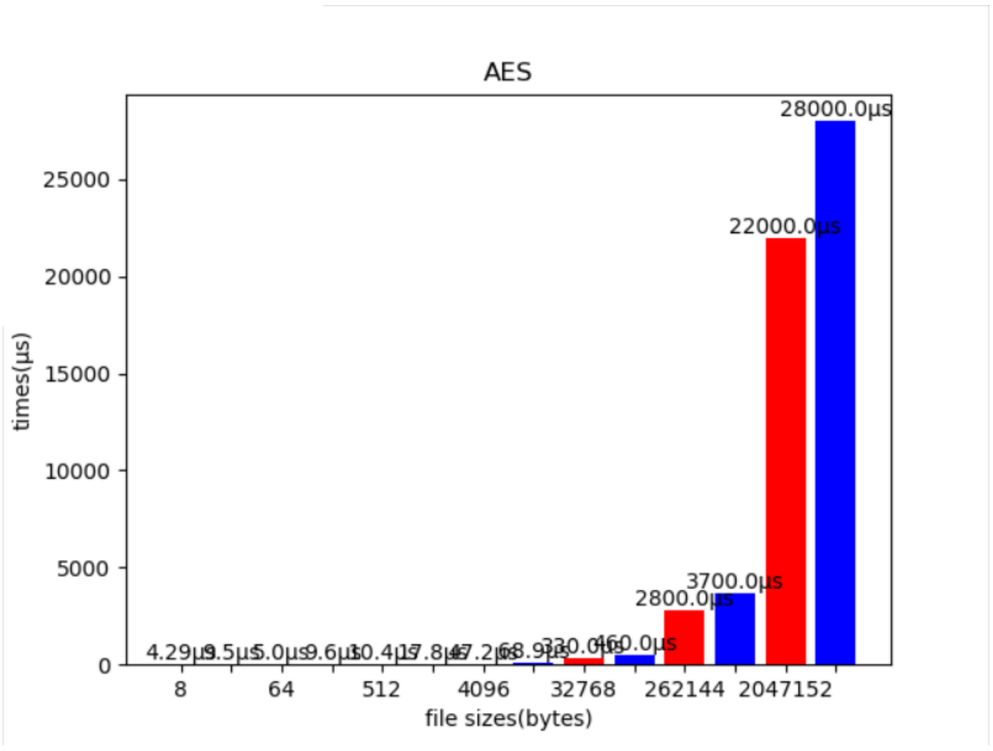
```
decrypted=bytes.decode(decrypted)
end_decrypt=time.time()
derypt_time = end_decrypt - start_decrypt
print(f'decrypt time is {decrypt_time}s')
decrypted=decrypted.rstrip(' ')
decrypted=bytes(decrypted,encoding='utf-8')
print ('decrypted', decrypted)
```

Graphs:
In DES, AES and RSA Graphs, red means encryption and blue means decryption.

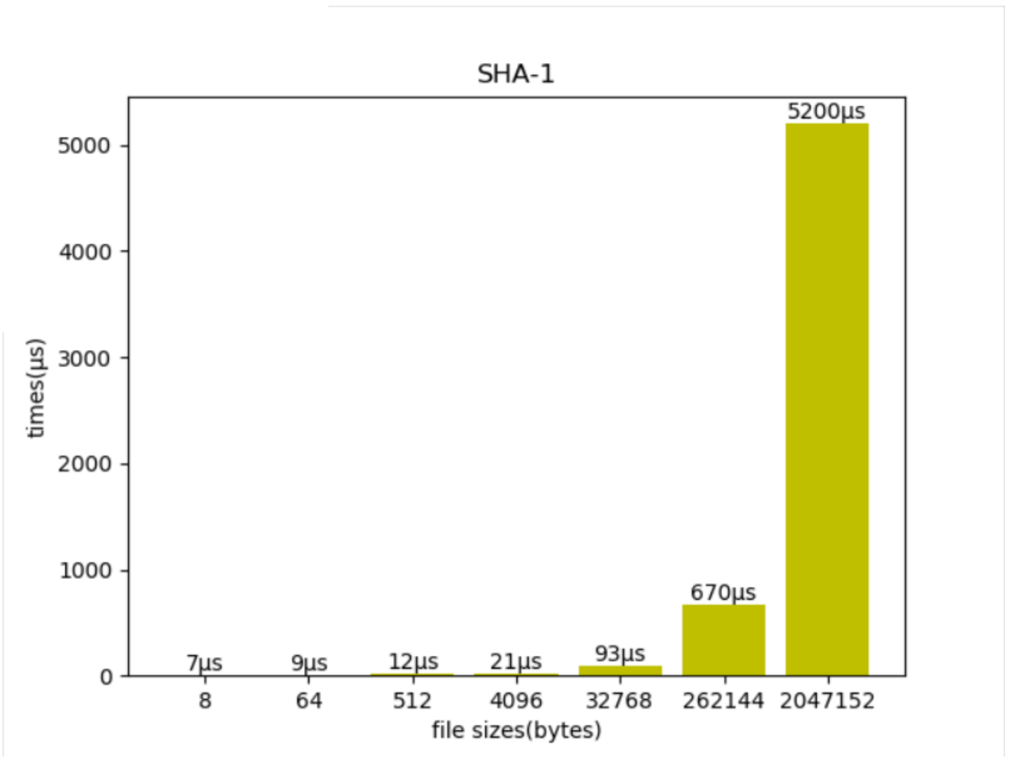
DES:



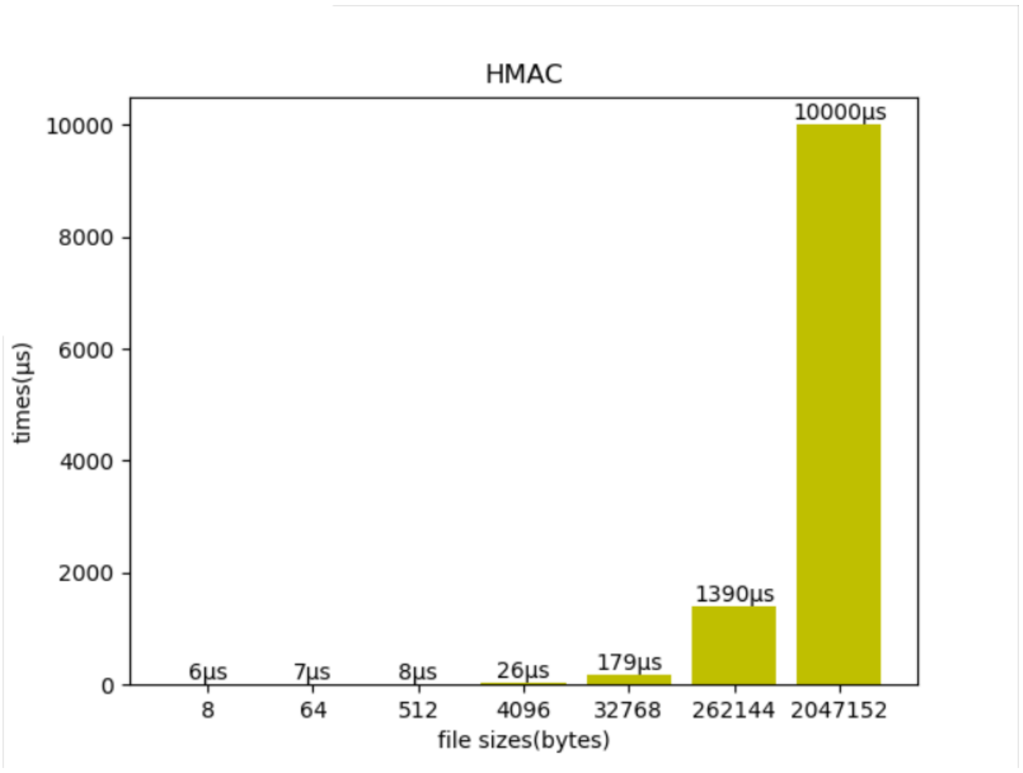
AES:



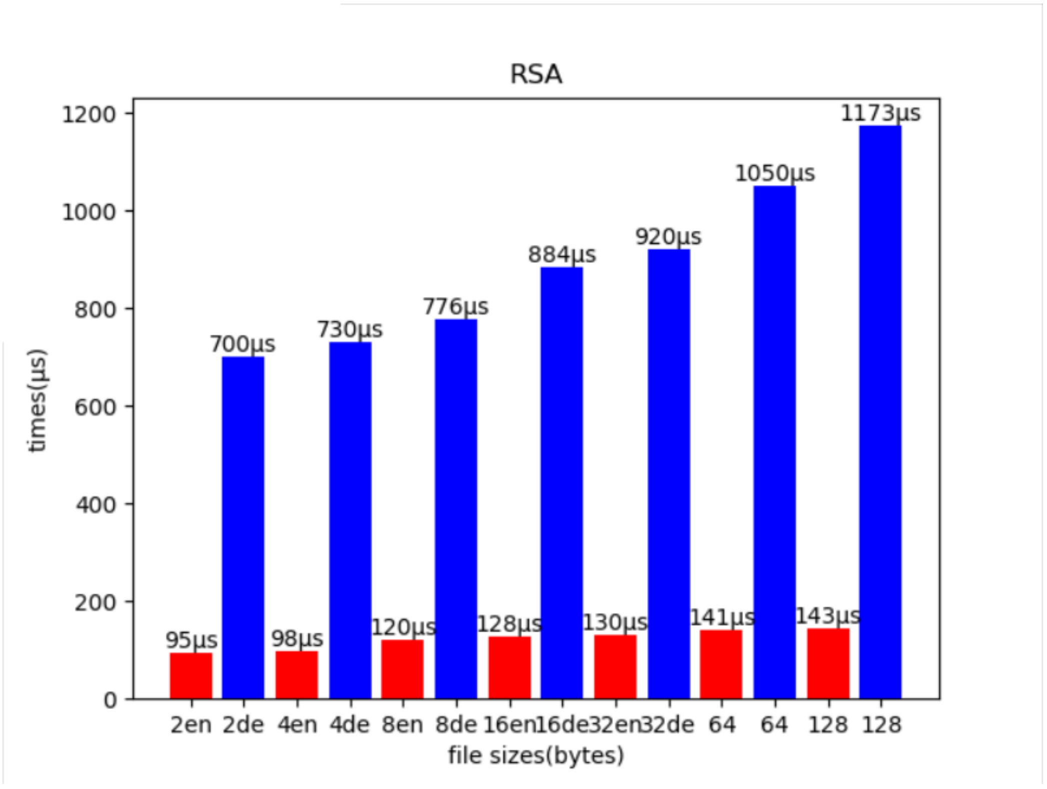
SHA-1:


SHA-1
times(µs)
5000
4000
3000
2000
1000
0
5200µs
670µs
7µs
9µs
12µs
21µs
93µs
8
64
512
4096
32768
262144
2047152
file sizes(bytes)

HMAC:


HMAC
times(µs)
10000
8000
6000
4000
2000
0
10000µs
1390µs
6µs
7µs
8µs
26µs
179µs
8
64
512
4096
32768
262144
2047152
file sizes(bytes)

RSA:



Question:

§ Compare DES encryption and AES encryption. Explain your observations.

As you can see from the figure, AES is more efficient than DES, especially when the file size is large. This is probably because DES was originally designed to be implemented quickly with hardware, and then people designed AES that has efficiency on both software and hardware, and AES uses Rijndael algorithm which is more advanced. In addition, AES has a longer key length than DES, so AES performs better than DES in terms of efficiency and security.

§ Compare DES encryption and RSA encryption. Explain your observations.

As can be seen from the graph, DES is more efficient than RSA in encryption efficiency. Because of the long key of RSA algorithm, the computation of encryption is very large, which results in the slow speed of encryption. However, its security is better than DES algorithm. In addition, DES is a Symmetric key cryptography algorithm, which means its encryption and decryption use the same key, but RSA is a Asymmetric key cryptography algorithm, which means that encryption and decryption use different keys.

§ Compare DES encryption and SHA-1 digest generation. Explain your observations.
SHA-1 and DES encryption have the similar efficiency. SHA-1 is slightly more efficient than DES, but SHA-1 is not an encryption algorithm, it is an algorithm used to confirm data integrity.

§ Compare HMAC signature generations and SHA-1 digest generation. Explain your observations.
SHA-1 and HMAC belong to hash algorithm, and SHA-1 has higher efficiency than HMAC. In fact, the built-in function used by HMAC in the experiment is MD5, so it is actually a comparison of efficiency between MD5 and SHA-1. Because the data digest length of SHA-1 algorithm is longer, MD5 is easier to collide than SHA-1. Finally, HMAC can also use hashing algorithms such as SHA-1, SHA-256 and so on.

§ Compare RSA encryption and decryption times. Can you explain your observations?

As can be seen from the figure, RSA encryption is faster than decryption. Because public key (e) can be chosen to encrypt manually, the encrypted ciphertext is a relatively small number, and to obtain plaintext from the ciphertext, the private key (d) is a larger number than the public key (e), so when calculating plaintext, a very large number will be obtained, which greatly reduces the efficiency of decryption.