COMP 9337 Securing Wireless Networks

T1, 2019

Lab 1

Group: SWN19 AI

Zhou JIANG (z5146092), Wanze LIU (z5137189)

School of Computer Science and Engineering

UNSW Sydney

README

This code should be compatible for **Python3.6**, which has been tested on CSE machine. Please typing:

python3 <python file > <arguments…>

to run the codes.

HOW TO TEST

To test tempdes.py file ONLY, please refer to the lab description for command-line-arguments.

INPLEMENTATION

Python code implementation in parts 3, 4.c and 4.d

In part 3.

1. We encrypt file by using the key and IV that provided by the lab1 materials (README)
2. After encryption finished, we write the encrypted message to the file named ‘mytest.des’ .
3. Checking whether the encryption file is matching the ‘test.des’ file provided.
4. Using checksum method to see if it matchs by typing command ‘ md5sum file name ‘ in the shell interface.

In part 4.c and 4.d, a separate code “run.py” calls other functions, following such steps:

1. It uses python built-in random method to generate random characters in original test files in different sizes.
2. Then it calls each encryption method (in different python files) to encrypt and/or decrypt such test files.
3. As each method returns encryption and/or decryption time, it records the time consumption for each method in using test files of different sizes.
4. Use matplotlib module to visualize the relation of time consumption and file size, in different methods.

We use such method to read test files in different sizes.

1. Input file as binary format
2. Calculating the file length, if the length of file is not multiple of 8 ,then we append extra byte in the end of the file by using ‘\x00’ string.
3. If the length of file is multiple of 8 , then we implement the algorithm to encrypt it directly.

Part 3 code tempdes.py is shown below:

```
from Crypto.Cipher import DES
import binascii
import time
import sys


def read_file(name):
    patch = '\x00'
    blocksize = 8
```

```
    with open(name) as f:
        a =    f.read()
        b = a

    if a:
        for i in range(0, len(a), blocksize):
            if i + blocksize < len(a):
                pass
            else:
                patch_num = 8 - len(a[i:])
                b = a + patch * (patch_num)

    return b

def DES_CBC_enc(cbc_key,iv,plain_text):
    des1 = DES.new(cbc_key, DES.MODE_CBC, iv)
    cipher_text = des1.encrypt(plain_text)
    return cipher_text



def    DES_CBC_dec(cbc_key,iv,cipher_text):
    des2 = DES.new(cbc_key, DES.MODE_CBC, iv)
    msg_text = des2.decrypt(cipher_text)
    return msg_text

def run(iv,key,inputfile, outputfile):
    test_file = read_file(inputfile)
    cbc_key = binascii.unhexlify(key)
    iv = binascii.unhexlify(iv)

    start1 = time.time()
    cipher = DES_CBC_enc(cbc_key, iv, test_file)
    cipher_time = time.time() - start1

    with open(outputfile, mode='wb') as file1:
        file1.write(cipher)

    start2 = time.time()
    decipher = DES_CBC_dec(cbc_key, iv, cipher)
    decipher_time = time.time() - start2

    with open('deciper_file.txt', mode='wb') as file:
        file.write(decipher)
```

```
    return cipher_time , decipher_time



if __name__ == '__main__':
    iv,key,inputfile,outputfile = sys.argv[1],sys.argv[2],sys.argv[3],sys.argv[4]
    a,b = run(iv,key,inputfile,outputfile)
    print('encryption time : ',a)
    print('decryption time : ',b)
```

Part 4.c code tempaes.py is shown below:

```
from Crypto.Cipher import AES
import time
from Crypto import Random
import sys



def read_file(name):
    patch = '\x00'
    blocksize = 16
    with open(name) as f:
        a = f.read()
        b = a

    if a:
        for i in range(0, len(a), blocksize):
            if i + blocksize < len(a):
                pass
            else:
                patch_num = 16 - len(a[i:])
                b = a + patch * (patch_num)

    return b



def AES_CBC_enc(cbc_key, iv, plain_text):
    aes1 = AES.new(cbc_key, AES.MODE_CBC, iv)
    cipher_text = aes1.encrypt(plain_text)
    return cipher_text



def AES_CBC_dec(cbc_key, iv, cipher_text):
```

```
    aes2 = AES.new(cbc_key, AES.MODE_CBC, iv)
    msg_text = aes2.decrypt(cipher_text)
    return msg_text


def run(inputfile, outputfile):
    iv = Random.get_random_bytes(16)
    key = Random.get_random_bytes(16)

    test_file = read_file(inputfile)

    start1 = time.time()
    cipher = AES_CBC_enc(key, iv, test_file)
    cipher_time = time.time() - start1


    with open(outputfile, mode='wb') as file1:
        file1.write(cipher)


    start2 = time.time()
    decipher = AES_CBC_dec(key, iv, cipher)
    decipher_time =time.time() -  start2


    with open('deciper_file.txt', mode='wb') as file:
        file.write(decipher)

    return cipher_time,decipher_time


if __name__ == '__main__':

    inputfile,outputfile = sys.argv[1],sys.argv[2]

    a,b = run(inputfile, outputfile)
    print('encryption time : ',a)
    print('decryption time : ',b)
```

Part 4.d code temprsa.py is shown below:

```
import Crypto
from Crypto.PublicKey import RSA
from Crypto import Random
```

```
import ast

import time

import sys



def read_file(name):

    patch = '\x00'

    blocksize = 8

    with open(name) as f:

        a = f.read()

        b = a


    if a:

        for i in range(0, len(a), blocksize):

            if i + blocksize < len(a):

                pass

            else:

                patch_num = 8 - len(a[i:])

                b = a + patch * (patch_num)


    return b



def run(name):

    file = read_file(name)

    random_generator = Random.new().read

    key = RSA.generate(1024, random_generator) #generate pub and priv key


    publickey = key.publickey() # pub key export for exchange


    start1 = time.time()

    try:    # for Python 2.7 and Python 3.6, different positions of arguments

        encrypted = publickey.encrypt(file,32)

    except TypeError:

        encrypted = publickey.encrypt(32,file)


    cipher_time = time.time() - start1


    #message to encrypt is in the above line 'encrypt this message'

    print('encrypted message:', encrypted) #ciphertext


    #decrypted code below

    start2 = time.time()

    decrypted = key.decrypt(ast.literal_eval(str(encrypted)))
```

```
    decipher_time = time.time() - start2
    print('decrypted', decrypted)

    return cipher_time , decipher_time



if __name__ == "__main__":

    inputfile = sys.argv[1]
    a,b = run(inputfile)
    print('encryption time : ',a)
    print('decryption time : ',b)
```

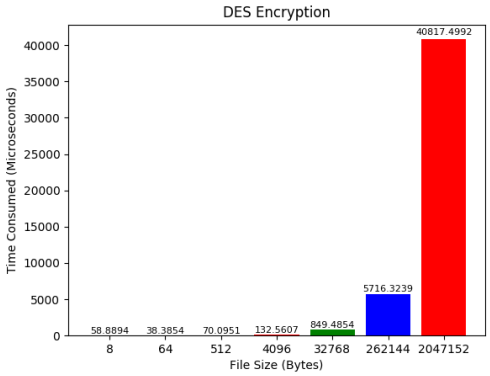
Part 4 separate code file run.py is shown below:

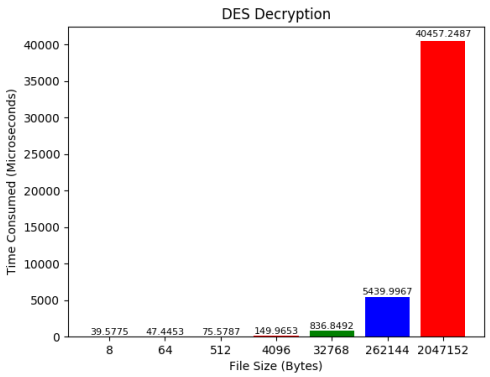
```
import time
import string
import random
import matplotlib.pyplot as plot

from tempaes import run as aesrun
from tempdes import run as desrun
from tempHMAC import run as HMACrun
from tempshal import run as shalrun
from temprsa import run as rsarun



KEY = '40fedf386da13d57'
IV = 'fedcba9876543210'

'''
    Structure Definition of self.type_size

    {Algorithm Name: [[File Size, Encryption Time, Decryption Time], ...], ...}
'''

class Testrun:
    def __init__(self):
        self.type_size = {'DES': [[8, 0, 0], [64, 0, 0], [512, 0, 0], [4096, 0, 0], [32768, 0, 0], [262144, 0, 0], [2047152, 0, 0]],
                          'AES': [[8, 0, 0], [64, 0, 0], [512, 0, 0], [4096, 0, 0], [32768, 0, 0], [262144, 0, 0], [2047152, 0, 0]],
                          'HMAC': [[8, 0, 0], [64, 0, 0], [512, 0, 0], [4096, 0, 0], [32768, 0, 0], [262144, 0, 0], [2047152, 0, 0]],
                          'SHA-1': [[8, 0, 0], [64, 0, 0], [512, 0, 0], [4096, 0, 0], [32768, 0, 0], [262144, 0, 0], [2047152, 0, 0]],
                          'RSA': [[2, 0, 0], [4, 0, 0], [8, 0, 0], [16, 0, 0], [32, 0, 0], [64, 0, 0], [128, 0, 0]]}
```

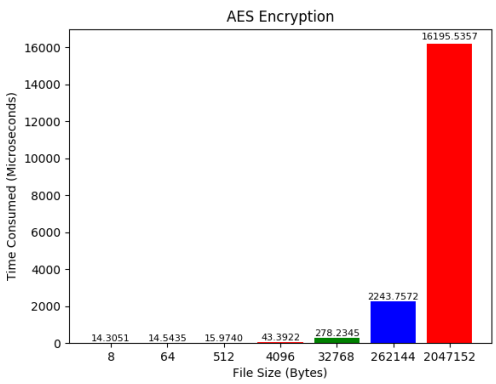
```
        self.time = None


    def generator(self):
        for type in self.type_size.keys():
            for size in self.type_size[type]:
                self.file_generator(type, size[0])



    def file_generator(self, type, size):
        file_name = type + '_' + str(size) + '.txt'
        print('Generating test file ' + file_name)
        with open(file_name, 'w') as file:
            for _ in range(0, size):
                file.write(random.choice(string.printable))


    def function_call(self):
        for des in self.type_size['DES']:
            des[1], des[2] = desrun(IV, KEY, 'DES' + '_' + str(des[0]) + '.txt', 'DES_OUT' + '_' + str(des[0]) + '.des')
        for aes in self.type_size['AES']:
            aes[1], aes[2] = aesrun('AES' + '_' + str(aes[0]) + '.txt', 'AES_OUT' + '_' + str(aes[0]) + '.aes')
        for HMAC in self.type_size['HMAC']:
            HMAC[1] = HMACrun('HMAC' + '_' + str(HMAC[0]) + '.txt')
        for sha1 in self.type_size['SHA-1']:
            sha1[1] = sha1run('SHA-1' + '_' + str(sha1[0]) + '.txt')
        for rsa in self.type_size['RSA']:
            rsa[1], rsa[2] = rsarun('RSA' + '_' + str(rsa[0]) + '.txt')



    def timer(self, flag='Stop'):
        if flag == 'Start':
            self.time = time.time()
        else:
            return time.time() - self.time


    def visualization(self):
        for type in self.type_size.keys():
            plot.bar(range(len(self.type_size[type])), [x[1] * 1000000 for x in self.type_size[type]],
            tick_label=[y[0] for y in self.type_size[type]], color='rgb')
            for a, b in enumerate([y[1] * 1000000 for y in self.type_size[type]]):
                plot.text(a, b + b * 0.01, '%.4f' %b, ha='center', va='bottom', fontsize=8)
            plot.title(type + ' Encryption')
            plot.xlabel('File Size (Bytes)')
            plot.ylabel('Time Consumed (Microseconds)')
            plot.savefig('./' + type + '_Encryption' + '.png')
```

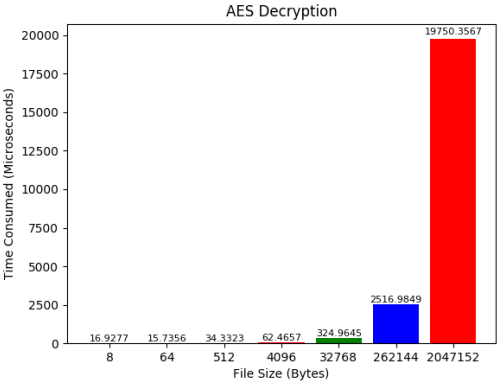
```
            plot.show()


            if type in ['SHA-1', 'HMAC']:
                continue
            plot.bar(range(len(self.type_size[type])), [x[2] * 1000000 for x in self.type_size[type]],
            tick_label=[y[0] for y in self.type_size[type]], color='rgb')
            for a, b in enumerate([y[2]  * 1000000 for y in self.type_size[type]]):
                plot.text(a, b + b * 0.01, '%.4f' %b, ha='center', va='bottom', fontsize=8)
            plot.title(type + ' Decryption')
            plot.xlabel('File Size (Bytes)')
            plot.ylabel('Time Consumed (Microseconds)')
            plot.savefig('./' + type + '_Decryption' + '.png')
            plot.show()



if __name__ == "__main__":
    testrun = Testrun()
    testrun.generator()
    testrun.function_call()
    testrun.visualization()
```

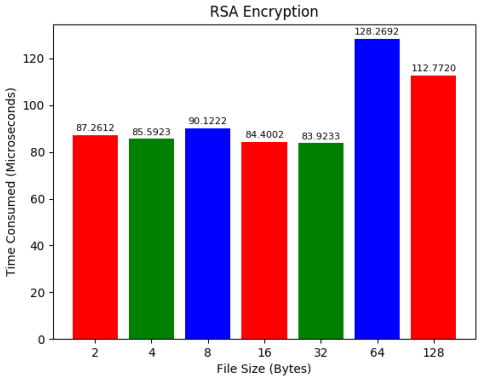
GRAPHS

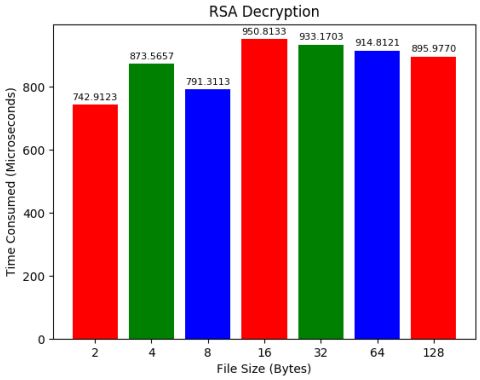

DES Encryption
Time Consumed (Microseconds)
File Size (Bytes)
58.8894
38.3854
70.0951
132.5607
849.4854
5716.3239
40817.4992
8
64
512
4096
32768
262144
2047152

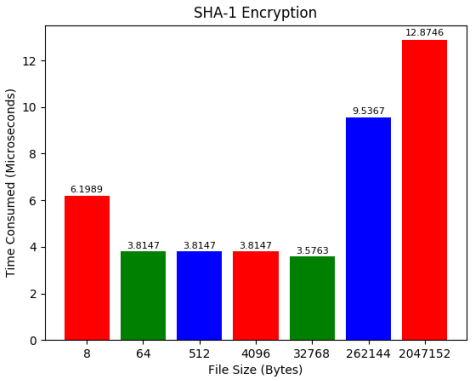

DES Decryption
Time Consumed (Microseconds)
File Size (Bytes)
39.5775
47.4453
75.5787
149.9653
836.8492
5439.9967
40457.2487
8
64
512
4096
32768
262144
2047152

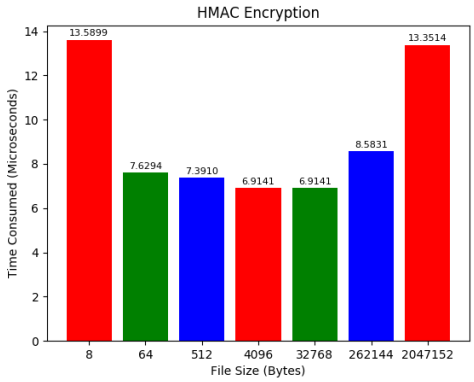

AES Encryption
Time Consumed (Microseconds)
File Size (Bytes)
14.3051
14.5435
15.9740
43.3922
278.2345
2243.7572
16195.5357
8
64
512
4096
32768
262144
2047152


AES Decryption
Time Consumed (Microseconds)
File Size (Bytes)
16.9277
15.7356
34.3323
62.4657
324.9645
2516.9849
19750.3567
8
64
512
4096
32768
262144
2047152


RSA Encryption
Time Consumed (Microseconds)
File Size (Bytes)
87.2612
85.5923
90.1222
84.4002
83.9233
128.2692
112.7720
2
4
8
16
32
64
128


RSA Decryption
Time Consumed (Microseconds)
File Size (Bytes)
742.9123
873.5657
791.3113
950.8133
933.1703
914.8121
895.9770
2
4
8
16
32
64
128


SHA-1 Encryption
Time Consumed (Microseconds)
File Size (Bytes)
6.1989
3.8147
3.8147
3.8147
3.5763
9.5367
12.8746
8
64
512
4096
32768
262144
2047152


HMAC Encryption
Time Consumed (Microseconds)
File Size (Bytes)
13.5899
7.6294
7.3910
6.9141
6.9141
8.5831
13.3514
8
64
512
4096
32768
262144
2047152

QUESTIONS

1. **Compare DES encryption and AES encryption. Explain your observations.**
   From the graphs, it is clear that DES encryption takes more time than AES encryption. Typically, consider within file size 2047152 Bytes, DES (40817μs) is about three-times inefficient than AES (16195μs).
   Thus, DES is comparatively slower than AES. This is because DES uses the Feistel network with permutation encryption steps, while AES uses permutation-substitution steps to encrypt.

2. **Compare DES encryption and RSA encryption. Explain your observations.**
   As the sizes of test files between DES and RSA are not all the same, we can just compare the time consumption within the file (size 8-Bytes and 64-Bytes) only.
   DES takes 58μs to encrypt 8-Bytes file, while RSA takes 90μs.
   DES takes 38μs to encrypt 64-Bytes file, while RSA takes 128μs.
   Thus, DES is timely more efficient than RSA, that's mainly because DES is a symmetric method, but RSA is an asymmetric method.

3. **Compare DES encryption and SHA-1 digest generation. Explain your observations.**
   DES runs much slower than SHA-1 digest. for the file size 2047152 Bytes, DES (40817μs) is considerably inefficient than SHA-1 (12μs).
   This is because SHA-1 is a hash-digest algorithm, which is generally faster than symmetric encryption method.

4. **Compare HMAC signature generations and SHA-1 digest generation. Explain your observations.**
   In my test result, SHA-1 is slightly efficient than HMAC, with the time difference up to 7μs. The maximum difference is shown in 8-Bytes test file, SHA-1 runs 6μs, while HMAC runs 13μs.
   I think this is possibly because HMAC uses salt strings, once the length of input string is less than a block length, padding is required, which results in more time consumption.

5. **Compare RSA encryption and decryption times. Can you explain your observations?**
   According to the figures, RSA encryption is much faster than its decryption. (ten-times faster in average).
   The reason is, RSA is based on large prime number decomposition. When encrypting, multiple large prime numbers can be less time-consuming, but decryption requires decomposition, which takes more time.