# 資料安全與密碼學assignment1

李易儒

2017/10/7 start

## 1 系統環境

macOS Sierra 10.12.6
python3.6.3

## 2 作業目標

### 2.1 產生出一個512MB+7byte的檔案

主要是為了需要做padding才多加7bytes

### 2.2 用Python的PyCrypto套件來執行

1.AES-256-ECB
2.AES-256-CBC
3.AES-256-CTR
4.RSA-2048
5.SHA-512

### 2.3 用Python的Cryptography套件來執行

1.AES-256-ECB
2.AES-256-CBC
3.AES-256-CTR
4.RSA-2048
5.SHA-512

### 2.4 padding

如果有要做padding需要先做

### 2.5 測量

測量2個套件所需時間並且比較

## 3 Latex

### 3.1 前置

由於我是mac，我在mac中已經有安裝homebrew，還沒安裝的請先去安裝homebrew，上網搜尋homebrew即可。

### 3.2 搜尋

打開terminal，輸入”brew search mactex” 先找到有沒有mactex套件。

### 3.3 安裝

輸入”brew install caskroom/cask/mactex” 等他載好，到application中找到TeX，進入Tex 開啓TeXShop，即可開始編輯。

## 4 PyCrypto

我是用python3來執行的，打開terminal，輸入”pip3 install pycrypto”安裝，完成之後就可以開始寫程式囉！

### 4.1 AES-256-ECB

```
# -*- coding: utf-8 -*-
from __future__ import absolute_import, division,
                                    unicode_literals
from Crypto.Cipher import AES
from Crypto import Random
import time
```

```
# padding
BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS
                                    )
unpad = lambda s : s[0:-ord(s[-1])]

# encrypt data
def encrypt(data, key):
    cryptor = AES.new(key, AES.MODE_ECB)
    return cryptor.encrypt(data)

# decrypt data
def decrypt(data, key):
  cryptor = AES.new(key, AES.MODE_ECB)
  return cryptor.decrypto(data)

# load 512MB file
origin_file = open('origin_file.txt', 'r')

# set key , 256 bits = 32 bytes
key = 'abcdefghijklmnopqrstuvwxyz123456'

# open file and load it
file_data = origin_file.read()
origin_file.close()

start = time.time()
encode_file_data = encrypt(pad(file_data),key)
end = time.time()
running_time = end-start
print('PyCrypyo AES-256-ECB encode time : '+str(running_time)
                                    )

start = time.time()
encrypt(encode_file_data,key)
end = time.time()
running_time = end-start
print('PyCrypyo AES-256-ECB decode time : '+str(running_time)
                                    )
```

## 4.2 AES-256-CBC

```
# -*- coding: utf-8 -*-
```

```
from __future__ import absolute_import, division,
                                unicode_literals

from Crypto.Cipher import AES

from Crypto import Random

import time

#padding
BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS
                                )
unpad = lambda s : s[0:-ord(s[-1])]


def encrypt(data, key):
  #encode
    cryptor = AES.new(key, AES.MODE_CBC, _IV)
    return cryptor.encrypt(data)

def decrypt(data, key):
  #decode
    cryptor = AES.new(key, AES.MODE_CBC, _IV)
    return cryptor.decrypt(data)



origin_file = open('origin_file.txt', 'r')
#265 bits = 32 bytes
key = 'abcdefghijklmnopqrstuvwxyz123456'


# open file and load it

file_data = origin_file.read()
origin_file.close()

_IV = Random.new().read(AES.block_size)

start = time.time()
encode_file_data = encrypt(pad(file_data),key)
end = time.time()
running_time = end-start
```

```
print('PyCrypto AES-256-CBC encode time : '+str(running_time)
                                    )



start = time.time()
encrypt(encode_file_data,key)
end = time.time()
running_time = end-start
print('PyCrypto AES-256-CBC decode time : '+str(running_time)
                                    )
```

## 4.3 AES-256-CTR

```
# -*- coding: utf-8 -*-

from __future__ import absolute_import, division,
                                    unicode_literals

from Crypto.Cipher import AES

from Crypto import Random

from Crypto.Util import Counter

import time

# padding
BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS
                                    )
unpad = lambda s : s[0:-ord(s[-1])]

# set counter
ctr = Counter.new(128)

def encrypt(data, key):
  cryptor = AES.new(key, AES.MODE_CTR, counter=ctr)
  return cryptor.encrypt(data)

def decrypt(data, key):
  cryptor = AES.new(key, AES.MODE_CTR, counter=ctr)
  return cryptor.decrypt(data)
```

```
# open file and load
origin_file = open('origin_file.txt', 'r')
# set key
key = 'abcdefghijklmnopqrstuvwxyz123456'

file_data = origin_file.read()
origin_file.close()

start = time.time()
encode_file_data = encrypt(pad(file_data),key)
end = time.time()
running_time = end-start
print('PyCrypto AES-256-CTR encode time : '+str(running_time)
                                    )




start = time.time()
decrypt(encode_file_data,key)
end = time.time()
running_time = end-start
print('PyCrypto AES-256-CTR decode time : '+str(running_time)
                                    )
```

## 4.4 RSA-2048

要先製作公鑰跟私鑰

```
from Crypto.PublicKey import RSA

def create_RSA_Key():
  key = RSA.generate(2048)
  file = open('rsa_private_key.txt','w')
  file.write(key.exportKey())
  file.close()
  file = open('rsa_public_key.txt','w')
  file.write(key.publickey().exportKey())
  file.close()
create_RSA_Key()
```

然後再開始加解密

```
# -*- coding: utf-8 -*-
```

```
from __future__ import absolute_import, division,
                                unicode_literals

from Crypto.Cipher import AES, PKCS1_OAEP

from Crypto.Random import get_random_bytes

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1_v1_5

from Crypto.Hash import SHA512

from Crypto import Random

import time

#padding
BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS
                                )
unpad = lambda s : s[0:-ord(s[-1])]



def encrypt(data,public_key,length = 200):
  public_recipient_key = RSA.importKey(public_key)
  cipher_rsa = PKCS1_v1_5.new(public_recipient_key)


  # because plaintext will too long , so load 200 at one time
  # the come out ciphertext will same length
  res = []
  for i in range(0,len(data),length):
    res.append(cipher_rsa.encrypt(data[i:i+length]))

  return res


def decrypt(data,private_key):

  private_recipient_key = RSA.importKey(private_key)
  plain_rsa = PKCS1_v1_5.new(private_recipient_key)
```

```
  dsize = SHA512.digest_size

  for i in range(0,len(data)):
    sentinel = Random.new().read(15 + dsize)
    plain_rsa.decrypt(data[i], sentinel)



# open file
origin_file = open('origin_file.txt', 'r')

file_data = origin_file.read()
origin_file.close()
# string into bytes by utf-8
file_data = bytes(file_data, 'utf-8')


start = time.time()

file = open('rsa_public_key.txt','r')
public_key = file.read()
file.close()

encode_file_data = encrypt(file_data,public_key)

end = time.time()
running_time = end-start
print('PyCrypto RSA-2048 encode time : '+str(running_time))


start = time.time()

file = open('rsa_private_key.txt','r')
private_key = file.read()
file.close()

decrypt(encode_file_data,private_key)

end = time.time()
running_time = end-start
print('PyCrypto RSA-2048 decode time : '+str(running_time))
```

## 4.5 SHA-512

```
# -*- coding: utf-8 -*-


# import library

from __future__ import absolute_import, division,
                                unicode_literals

from Crypto.Cipher import AES

from Crypto import Random

import time

from Crypto.Hash import SHA512


# padding
BS = AES.block_size
pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS
                                    )
unpad = lambda s : s[0:-ord(s[-1])]

# encrypt data
def encrypt(data):
    h = SHA512.new()
    h.update(data)
    return h

# open file
origin_file = open('origin_file.txt', 'r')

# read file_data
file_data = origin_file.read()

origin_file.close()


start = time.time()
# do padding
file_data = pad(file_data)


file_data = str.encode(file_data)
```

```
a=encrypt(file_data)
print(a)
end = time.time()
running_time = end-start
print('PyCrypto SHA-512 encode time : '+str(running_time))
```

# 5 Cryptography

我是用python3來執行的，打開terminal，輸入"pip3 install cryptography"安裝，完成之後就可以開始寫程式囉！

## 5.1 AES-256-ECB

```
from cryptography.hazmat.primitives import padding

from cryptography.hazmat.primitives.ciphers import Cipher,
                                algorithms, modes

from cryptography.hazmat.backends import default_backend

import time, os

# do padding
def padding_data(data):
  padder = padding.PKCS7(128).padder()
  padded_data = padder.update(data)
  padded_data += padder.finalize()
  return padded_data

# encrypt
def encrypt(data,key):
  cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=
                                backend)
  encryptor = cipher.encryptor()
  encode_data = encryptor.update(data) + encryptor.finalize()
  return encode_data

# decrypt
def decrypt(data,key):
  cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=
                                backend)
  decryptor = cipher.decryptor()
```

```
  decryptor.update(data) + decryptor.finalize()



# open file
origin_file = open('origin_file.txt', 'r')
file_data = origin_file.read()
origin_file.close()

# str to bytes
file_data = bytes(file_data, 'utf-8')


file_data = padding_data(file_data)

backend = default_backend()
# set key
key = os.urandom(32)



start = time.time()

encode_data = encrypt(file_data,key)


end = time.time()
running_time = end-start
print('Cryptography AES-256-ECB encode time : '+str(
                                running_time))

start = time.time()

decrypt(encode_data,key)

end = time.time()
running_time = end-start
print('Cryptography AES-256-ECB decode time : '+str(
                                running_time))
```

## 5.2 AES-256-CBC

```
from cryptography.hazmat.primitives import padding
```

```
from cryptography.hazmat.primitives.ciphers import Cipher,
                                    algorithms, modes

from cryptography.hazmat.backends import default_backend

import time, os

def padding_data(data):
  padder = padding.PKCS7(128).padder()
  padded_data = padder.update(data)
  padded_data += padder.finalize()
  return padded_data

def encrypt(data,key,iv):
  cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend
                                    =backend)
  encryptor = cipher.encryptor()
  encode_data = encryptor.update(data) + encryptor.finalize()
  return encode_data


def decrypt(data,key,iv):
  cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend
                                    =backend)
  decryptor = cipher.decryptor()
  decryptor.update(data) + decryptor.finalize()




origin_file = open('origin_file.txt', 'r')
file_data = origin_file.read()
origin_file.close()

# str to bytes
file_data = bytes(file_data, 'utf-8')

# padding
file_data = padding_data(file_data)

backend = default_backend()
key = os.urandom(32)
iv = os.urandom(16)
```

```
start = time.time()

encode_data = encrypt(file_data,key,iv)


end = time.time()
running_time = end-start
print('Cryptography AES-256-CBC encode time : '+str(
                                running_time))

start = time.time()

decrypt(encode_data,key,iv)

end = time.time()
running_time = end-start
print('Cryptography AES-256-CBC decode time : '+str(
                                running_time))
```

## 5.3 AES-256-CTR

```
from cryptography.hazmat.primitives import padding

from cryptography.hazmat.primitives.ciphers import Cipher,
                                algorithms, modes

from cryptography.hazmat.backends import default_backend

import time, os


# encrypt
def encrypt(data,key,iv):
  cipher = Cipher(algorithms.AES(key), modes.CTR(iv), backend
                                =backend)
  encryptor = cipher.encryptor()
  encode_data = encryptor.update(data) + encryptor.finalize()
  return encode_data

# decrypt
def decrypt(data,key,iv):
```

```
  cipher = Cipher(algorithms.AES(key), modes.CTR(iv), backend
                                  =backend)
  decryptor = cipher.decryptor()
  decryptor.update(data) + decryptor.finalize()




# open file
origin_file = open('origin_file.txt', 'r')
file_data = origin_file.read()
origin_file.close()
file_data = bytes(file_data, 'utf-8')


# set key, iv, backend
backend = default_backend()
key = os.urandom(32)
iv = os.urandom(16)



start = time.time()

encode_data = encrypt(file_data,key,iv)


end = time.time()
running_time = end-start
print('Cryptography AES-256-CTR encode time : '+str(
                                  running_time))

start = time.time()

decrypt(encode_data,key,iv)

end = time.time()
running_time = end-start
print('Cryptography AES-256-CTR decode time : '+str(
                                  running_time))
```

## 5.4 RSA-2048

```
from cryptography.hazmat.backends import default_backend
```

```
from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives.asymmetric import padding

from cryptography.hazmat.primitives import hashes

import time


###### open data
origin_file = open('origin_file.txt', 'r')
file_data = origin_file.read()
origin_file.close()
file_data = bytes(file_data, 'utf-8')



###### create and load private key
private_key = rsa.generate_private_key(public_exponent=65537,
                                key_size=2048,backend=
                                default_backend())

###### create and load public key
public_key = private_key.public_key()
pem = public_key.public_bytes(encoding=serialization.Encoding
                                .PEM,format=serialization.
                                PublicFormat.
                                SubjectPublicKeyInfo)

####### encrypt
start = time.time()

encode_data = []
length = 200
# plaintext will too long so load part of to encrypt
for i in range(0,len(file_data),length):
  encode_data.append(public_key.encrypt(file_data[i:i+length]
                                ,padding.OAEP(mgf=padding.MGF1
                                (algorithm=hashes.SHA1()),
                                algorithm=hashes.SHA1(),label=
                                None)))
end = time.time()
```

```
running_time = end-start
print('Cryptography RSA-2048 encode time : '+str(running_time
                                ))

####### decrypt
start = time.time()
plaintext = []
for i in range(0,len(encode_data)):
  plaintext.append(private_key.decrypt(encode_data[i],padding
                                .OAEP(mgf=padding.MGF1(
                                algorithm=hashes.SHA1()),
                                algorithm=hashes.SHA1(),label=
                                None)))

end = time.time()
running_time = end-start
print('Cryptography RSA-2048 decode time : '+str(running_time
                                ))
```

## 5.5 SHA-512

```
from cryptography.hazmat.backends import default_backend

from cryptography.hazmat.primitives import hashes

import time




###### open data
origin_file = open('origin_file.txt', 'r')
file_data = origin_file.read()
origin_file.close()
file_data = bytes(file_data, 'utf-8')


####### encrypt
start = time.time()
```

```
digest = hashes.Hash(hashes.SHA512(), backend=default_backend
                                    ())
digest.update(file_data)
digest.finalize()


end = time.time()
running_time = end-start
print('Cryptography SHA-512 encode time : '+str(running_time)
                                    )
```

# 6 比較

Table 1: 時間比較表

| | ***PyCrypto*** | ***Cryptography*** |
|---|---|---|
| *en AES-ECB* | 4.44237 | 1.24611 |
| *de AES-ECB* | 4.23712 | 0.95573 |
| *en AES-CBC* | 4.87203 | 1.40536 |
| *de AES-CBC* | 5.99750 | 1.02816 |
| *en AES-CTR* | 5.92283 | 1.03212 |
| *de AEC-CTR* | 5.44241 | 1.10142 |
| *en RSA* | 1700.11902 | 119.33044 |
| *de RSA* | 28927.44398 | 1771.46918 |
| *en SHA* | 1.36848 | 0.89161 |

由上表可以發現：
1.在Package速度上，Cryptography比PyCrypto還要快。
2.在演算法速度上：
在PyCrypto是SHA最快，再來是ECB、CBC、CTR，RSA最慢。
在Cryptography是SHA最快，再來是ECB、CTR、CBC，RSA最慢。
3.RSA解密的時間比加密還要多很多。