

資訊安全作業 Assignment1

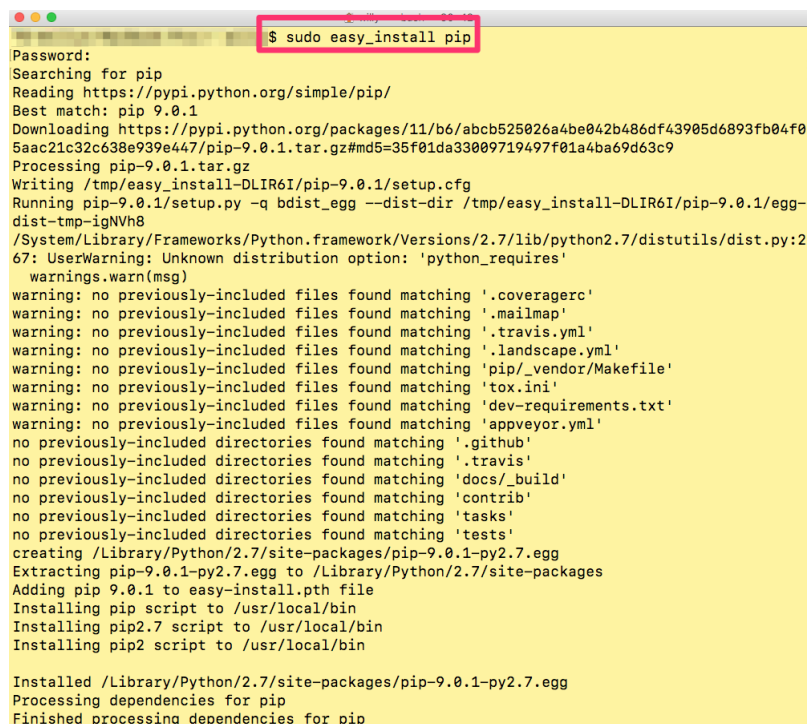
吳嘉偉 5105056013

2017/03/26

1 安裝 Python Packages

1.1 使用終端機安裝 pip

打開 terminal 後，先輸入指令安裝 pip
`sudo easy_install pip`

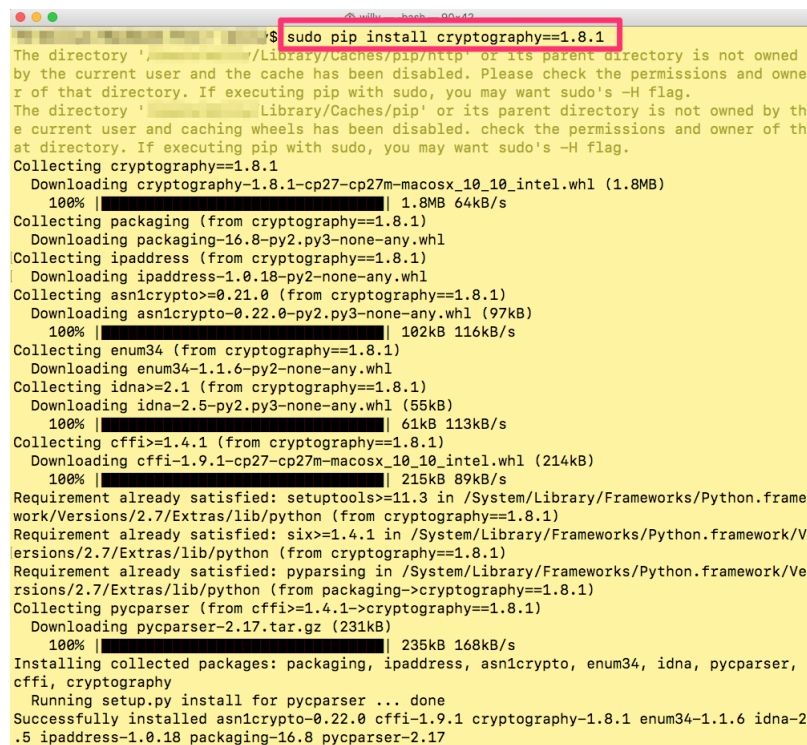
A screenshot of a macOS terminal window. The title bar shows standard macOS window controls. The terminal prompt is '\$ sudo easy_install pip', which is highlighted with a red rectangular box. Below the prompt, the terminal displays the output of the command, including searching for pip, downloading the source code from PyPI, and installing it to the system's site-packages directory. The output also shows several warnings about previously included files and directories, and finally confirms the installation of pip-9.0.1-py2.7.egg and the installation of the pip script to /usr/local/bin.

```
$ sudo easy_install pip
Password:
Searching for pip
Reading https://pypi.python.org/simple/pip/
Best match: pip 9.0.1
Downloading https://pypi.python.org/packages/11/b6/abcb525026a4be042b486df43905d6893fb04f0
5aac21c32c638e939e447/pip-9.0.1.tar.gz#md5=35f01da33009719497f01a4ba69d63c9
Processing pip-9.0.1.tar.gz
Writing /tmp/easy_install-DLIR6I/pip-9.0.1/setup.cfg
Running pip-9.0.1/setup.py -q bdist_egg --dist-dir /tmp/easy_install-DLIR6I/pip-9.0.1/egg-
dist-tmp-igNVh8
/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/distutils/dist.py:2
67: UserWarning: Unknown distribution option: 'python_requires'
  warnings.warn(msg)
warning: no previously-included files found matching '.coveragerc'
warning: no previously-included files found matching '.mailmap'
warning: no previously-included files found matching '.travis.yml'
warning: no previously-included files found matching '.landscape.yml'
warning: no previously-included files found matching 'pip/_vendor/Makefile'
warning: no previously-included files found matching 'tox.ini'
warning: no previously-included files found matching 'dev-requirements.txt'
warning: no previously-included files found matching 'appveyor.yml'
no previously-included directories found matching '.github'
no previously-included directories found matching '.travis'
no previously-included directories found matching 'docs/_build'
no previously-included directories found matching 'contrib'
no previously-included directories found matching 'tasks'
no previously-included directories found matching 'tests'
creating /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg
Extracting pip-9.0.1-py2.7.egg to /Library/Python/2.7/site-packages
Adding pip 9.0.1 to easy-install.pth file
Installing pip script to /usr/local/bin
Installing pip2.7 script to /usr/local/bin
Installing pip2 script to /usr/local/bin

Installed /Library/Python/2.7/site-packages/pip-9.0.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip
```

Figure 1: 安裝 pip

接著輸入下圖指令，安裝需要的 package
這邊是安裝 cryptography，版本為 1.8.1
sudo pip install packagename==version
packagename：要安裝的 package 名稱
version：要安裝的版本號碼

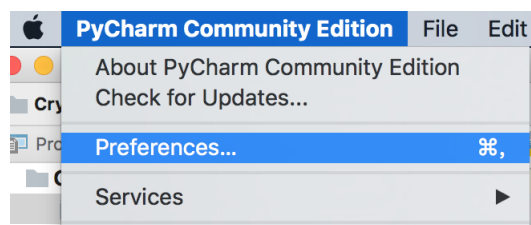


```
$ sudo pip install cryptography==1.8.1
The directory '/Library/Caches/pip/http' or its parent directory is not owned
by the current user and the cache has been disabled. Please check the permissions and own
er of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/Library/Caches/pip' or its parent directory is not owned by th
e current user and caching wheels has been disabled. check the permissions and owner of th
at directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting cryptography==1.8.1
  Downloading cryptography-1.8.1-cp27-cp27m-macosx_10_10_intel.whl (1.8MB)
    100% |#####| 1.8MB 64kB/s
Collecting packaging (from cryptography==1.8.1)
  Downloading packaging-16.8-py2.py3-none-any.whl
Collecting ipaddress (from cryptography==1.8.1)
  Downloading ipaddress-1.0.18-py2-none-any.whl
Collecting asn1crypto>=0.21.0 (from cryptography==1.8.1)
  Downloading asn1crypto-0.22.0-py2.py3-none-any.whl (97kB)
    100% |#####| 102kB 116kB/s
Collecting enum34 (from cryptography==1.8.1)
  Downloading enum34-1.1.6-py2-none-any.whl
Collecting idna>=2.1 (from cryptography==1.8.1)
  Downloading idna-2.5-py2.py3-none-any.whl (55kB)
    100% |#####| 61kB 113kB/s
Collecting cffi>=1.4.1 (from cryptography==1.8.1)
  Downloading cffi-1.9.1-cp27-cp27m-macosx_10_10_intel.whl (214kB)
    100% |#####| 215kB 89kB/s
Requirement already satisfied: setuptools>=11.3 in /System/Library/Frameworks/Python.frame
work/Versions/2.7/Extras/lib/python (from cryptography==1.8.1)
Requirement already satisfied: six>=1.4.1 in /System/Library/Frameworks/Python.framework/V
ersions/2.7/Extras/lib/python (from cryptography==1.8.1)
Requirement already satisfied: pyparsing in /System/Library/Frameworks/Python.framework/Ve
rsions/2.7/Extras/lib/python (from packaging->cryptography==1.8.1)
Collecting pycparser (from cffi>=1.4.1->cryptography==1.8.1)
  Downloading pycparser-2.17.tar.gz (231kB)
    100% |#####| 235kB 168kB/s
Installing collected packages: packaging, ipaddress, asn1crypto, enum34, idna, pycparser,
cffi, cryptography
  Running setup.py install for pycparser ... done
Successfully installed asn1crypto-0.22.0 cffi-1.9.1 cryptography-1.8.1 enum34-1.1.6 idna-2
.5 ipaddress-1.0.18 packaging-16.8 pycparser-2.17
```

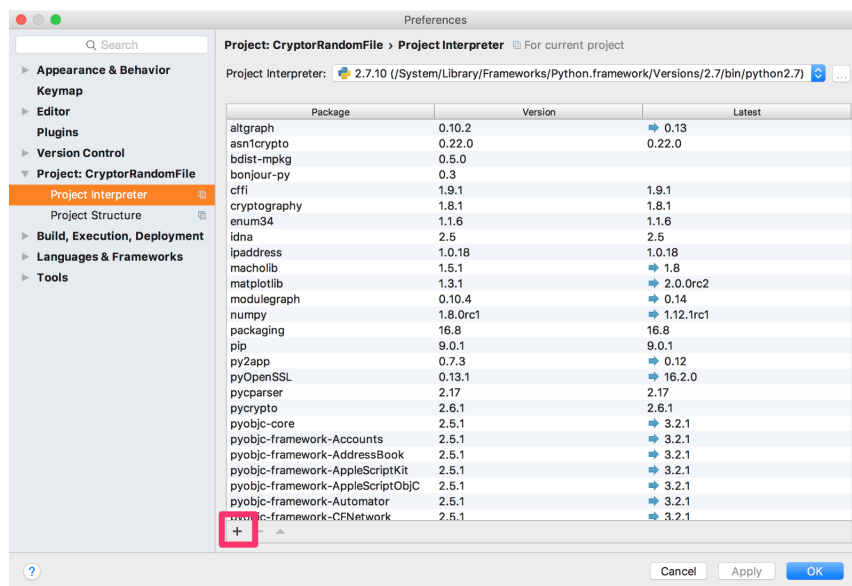
Figure 2: 安裝 package

1.2 使用 PyCharm

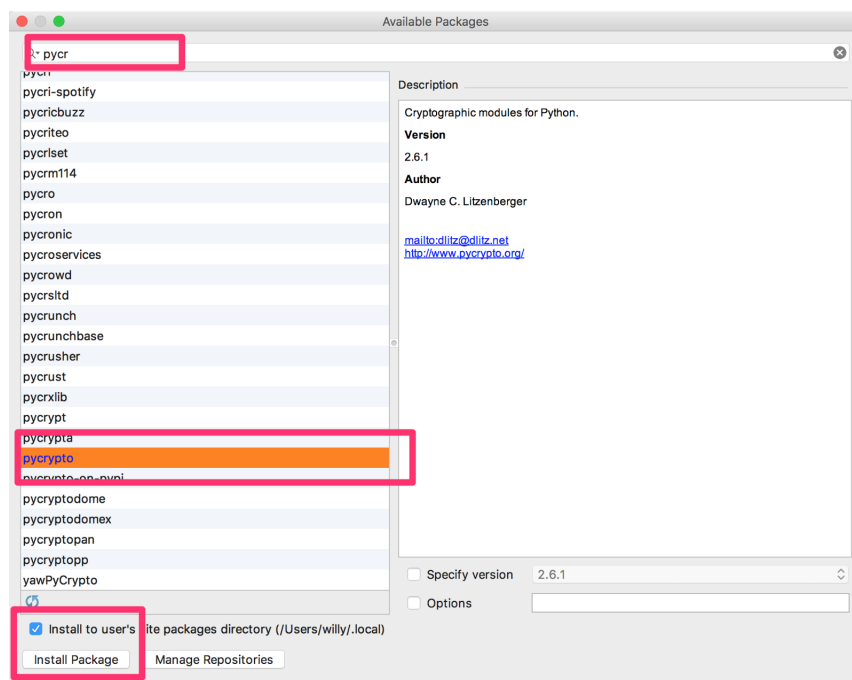
打開 PyCharmCE，先點選 Preferences...



選擇 Project: ProjectName » Project Interpreter » +



輸入要加入的 package name



2 PyCrypto

2.1 PyCrypto AES-256 ECB

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

from __future__ import absolute_import, division, unicode_literals

import os
from Crypto.Cipher import AES

_KEY = os.urandom(32)          # 設定Key
_BlockSize = AES.block_size    # Block Size

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
        chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用ECB加密
def aes_encrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_ECB)
    return cryptor.encrypt(_Pad(data))

# 使用ECB解密
def aes_decrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_ECB)
    return _Unpad(cryptor.decrypt(data))
```

2.2 PyCrypto AES-256 CBC

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

from __future__ import absolute_import, division, unicode_literals

import os
from Crypto.Cipher import AES
```

```

_IV = os.urandom(16)           # 產生隨機亂數 IV
_KEY = os.urandom(32)          # 設定 Key
_BlockSize = AES.block_size    # Block Size

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
      chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用CBC加密
def aes_encrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_CBC, _IV)
    return cryptor.encrypt(_Pad(data))

# 使用CBC解密
def aes_decrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_CBC, _IV)
    return _Unpad(cryptor.decrypt(data))

```

2.3 PyCrypto AES-256 CTR

```

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

from __future__ import absolute_import, division, unicode_literals

import os
from Crypto.Cipher import AES

_IV = os.urandom(16)           # 產生隨機亂數 IV
_KEY = os.urandom(32)          # 設定 Key
_COUNTER = os.urandom(16)
_BlockSize = AES.block_size    # Block Size

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
      chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用CTR加密

```

```

def aes_encrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_CTR, counter=lambda: _COUNTER)
    return cryptor.encrypt(_Pad(data))

# 使用CTR解密
def aes_decrypt(data):
    cryptor = AES.new(_KEY, AES.MODE_CTR, counter=lambda: _COUNTER)
    return _Unpad(cryptor.decrypt(data))

```

2.4 PyCrypto RSA-2048

```

# 使用RSA加密
def rsa_encrypt(data):
    encrypted = _PUBLICKEY.encrypt(_Pad(data), 16)
    return encrypted

# 使用RSA解密
def rsa_decrypt(data):
    decrypted = _KEY.decrypt(data)
    return _Unpad(decrypted)

```

2.5 PyCrypto SHA-512

```

# -*- coding: utf-8 -*-
from __future__ import absolute_import, division, unicode_literals

from Crypto.Hash import SHA512

def hashSHA512(data):
    hash = SHA512.new()
    hash.update(data)
    return hash.hexdigest()

```

3 Cryptography

3.1 Cryptography AES-256 ECB

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

backend = default_backend()
_KEY = os.urandom(32)    # 設定Key
_BlockSize = 16         # Block Size

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
    chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用ECB加密
def aes_encrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.ECB(), backend=backend)
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(_Pad(data))
    return ciphertext

# 使用ECB解密
def aes_decrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.ECB(), backend=backend)
    decryptor = cipher.decryptor()
    plaintext = _Unpad(decryptor.update(data))
    return plaintext
```

3.2 Cryptography AES-256 CBC

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

```

backend = default_backend()
_IV = os.urandom(16)      # 產生隨機亂數 IV
_KEY = os.urandom(32)     # 設定 Key
_BlockSize = 16           # Block Size

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
      chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用CBC加密
def aes_encrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.CBC(_IV),
                    backend=backend)
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(_Pad(data))
    return ciphertext

# 使用CBC解密
def aes_decrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.CBC(_IV),
                    backend=backend)
    decryptor = cipher.decryptor()
    plaintext = _Unpad(decryptor.update(data))
    return plaintext

```

3.3 Cryptography AES-256 CTR

```

import os
from cryptography.hazmat.primitives.ciphers import Cipher,
    algorithms, modes
from cryptography.hazmat.backends import default_backend

backend = default_backend()
_IV = os.urandom(16)      # 產生隨機亂數 IV
_KEY = os.urandom(32)     # 設定 Key
_BlockSize = 16           # Block Size

```



```

#padding
_Pad = lambda s: s + (_BlockSize - len(s) % _BlockSize) *
        chr(_BlockSize - len(s) % _BlockSize)
_Unpad = lambda s : s[0:-ord(s[-1])]

# 使用CTR加密
def aes_encrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.CTR(_IV),
                    backend=backend)
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(_Pad(data))
    return ciphertext

# 使用CTR解密
def aes_decrypt(data):
    cipher = Cipher(algorithms.AES(_KEY), modes.CTR(_IV),
                    backend=backend)
    decryptor = cipher.decryptor()
    plaintext = _Unpad(decryptor.update(data))
    return plaintext

```

3.4 Cryptography RSA-2048

```

def rsa_encrypt(data):
    # ciphertext= ''
    for text in data:
        cipher = _PUBLICKEY.encrypt(
            text,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA1()),
                algorithm=hashes.SHA1(),
                label=None
            )
        )

```

3.5 Cryptography SHA-512

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

backend = default_backend()

def hashSHA512(data):
    digest = hashes.Hash(hashes.SHA512(), backend=default_backend())
    digest.update(data)
    return digest.finalize()
```

4 PyCrypto 與 Cryptography 加密的執行速度

利用 Python 隨機產生一個大小為 1MB+7byte 的字串檔案並且丟到每一個加密演算法計算時間

4.1 隨機產生字串檔案

```
# -*- coding: utf-8 -*-
import random, string

# 隨機產生大小為 size 的字串檔案
def generateStringFile(size):
    text = ''.join(random.choice(string.ascii_letters +
                                string.digits) for x in range(size))
    file = open('RandomString.txt', 'wb')
    file.write(text)
```

4.2 AES-256 ECB

```
# PyCrypto_AES_256_ECB
print ('PyCrypto_AES_256_ECB')
start = time.time()
encrypt = PyCrypto_AES_256_ECB.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ('Time taken: ' + str(elapsed) + 'seconds.')
```



```
# Cryptography_AES_256_ECB
print ('Cryptography_AES_256_ECB')
start = time.time()
encrypt = Cryptography_AES_256_ECB.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ('Time taken: ' + str(elapsed) + 'seconds.')
```

輸出的結果為

PyCrypto AES256 ECB Time taken: 0.00929403305054seconds.

Cryptography AES256 ECB Time taken: 0.00262188911438seconds.

4.3 AES-256 CBC

```
# PyCrypto_AES_256_CBC
print ( 'PyCrypto_AES_256_CBC' )
start = time.time()
encrypt = PyCrypto_AES_256_CBC.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )

# Cryptography_AES_256_CBC
print ( 'Cryptography_AES_256_CBC' )
start = time.time()
encrypt = Cryptography_AES_256_CBC.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )
```

輸出的結果為

PyCrypto AES256 CBC Time taken: 0.0101928710938seconds.

Cryptography AES256 CBC Time taken: 0.00265407562256seconds.

4.4 AES-256 CTR

```
# PyCrypto_AES_256_CTR
print ( 'PyCrypto_AES_256_CTR' )
start = time.time()
encrypt = PyCrypto_AES_256_CTR.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )

# Cryptography_AES_256_CTR
```

```

print ( 'Cryptography_AES_256_CTR' )
start = time.time()
encrypt = Cryptography_AES_256_CTR.aes_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )

```

輸出的結果為

PyCrypto AES256 CTR Time taken: 0.0238711833954seconds.

Cryptography AES256 CTR Time taken: 0.000817060470581seconds.

4.5 RSA-2048

```

# PyCryo_RSA_2048
print ( 'PyCryo_RSA2048' )
start = time.time()
encrypt = PyCryo_RSA_2048.rsa_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )

# Cryptography_RSA_2048
print ( 'Cryptography_RSA2048' )
start = time.time()
encrypt = Cryptography_RSA_2048.rsa_encrypt(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.' )

```

輸出的結果為

PyCrypto RSA2048 Time taken: 103.83675909seconds.

Cryptography RSA2048 Time taken: 87.8018479347seconds.

4.6 SHA-512

```

# PyCryo_SHA_512
print ( 'PyCryo_SHA_512' )

```

```

start = time.time()
hash = PyCryo_SHA_512.hashSHA512(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.')

# Cryptography_SHA_512
print ( 'Cryptography_SHA_512 ')
start = time.time()
hash = Cryptography_SHA_512.hashSHA512(plaintext)
end = time.time()
elapsed = end - start
print ( 'Time taken: ' + str(elapsed) + 'seconds.')

```

輸出的結果為

PyCryo SHA512 Time taken: 0.0035879611969seconds.

Cryptography SHA512 Time taken: 0.00237917900085seconds.

5 比較 PyCrypto 與 Cryptography 的加密執行速度

5.1 隨機產生大小為 1MB+7byte 的字串檔案

	ECB	CBC	CTR	RSA2048	SHA512
PyCrypto	0.00929	0.01019	0.02387	103.83676	0.00359
Cryptography	0.00262	0.00265	0.00081	60.44971	0.00238

5.2 隨機產生大小為 512MB+7byte 的字串檔案

	ECB	CBC	CTR	RSA2048	SHA512
PyCrypto	5.21309	5.64221	13.40350	—	1.96496
Cryptography	1.22992	2.13249	1.34670	—	1.24736

6 結論

依照加密演算法來看 SHA512 > AES > RSA248。

其中，SHA>AES 我想應該是因為 SHA 不需考慮解密，所以演算法的效率就可以比 AES 來得高。

而 AES>RSA2048 則是可以預期的結果，而且 RSA 在實務上也不會對字串作加密。

另外，Cryptography 速度也都比 PyCrypto 來得快。

以上內容以及程式碼可以參考我的 github 以及 medium

<https://github.com/WillyWu0201/CryptorRandomFile>

<https://medium.com/@willywu/%E5%88%A9%E7%94%A8pip%E5%AE%89%E8%A3%9Dpython-package-dc3e82348c29>