

對稱式金鑰加密實作

如何執行

- 先用指令產生一個大型隨機檔案，這個檔案是要當作我們的加密的 input：

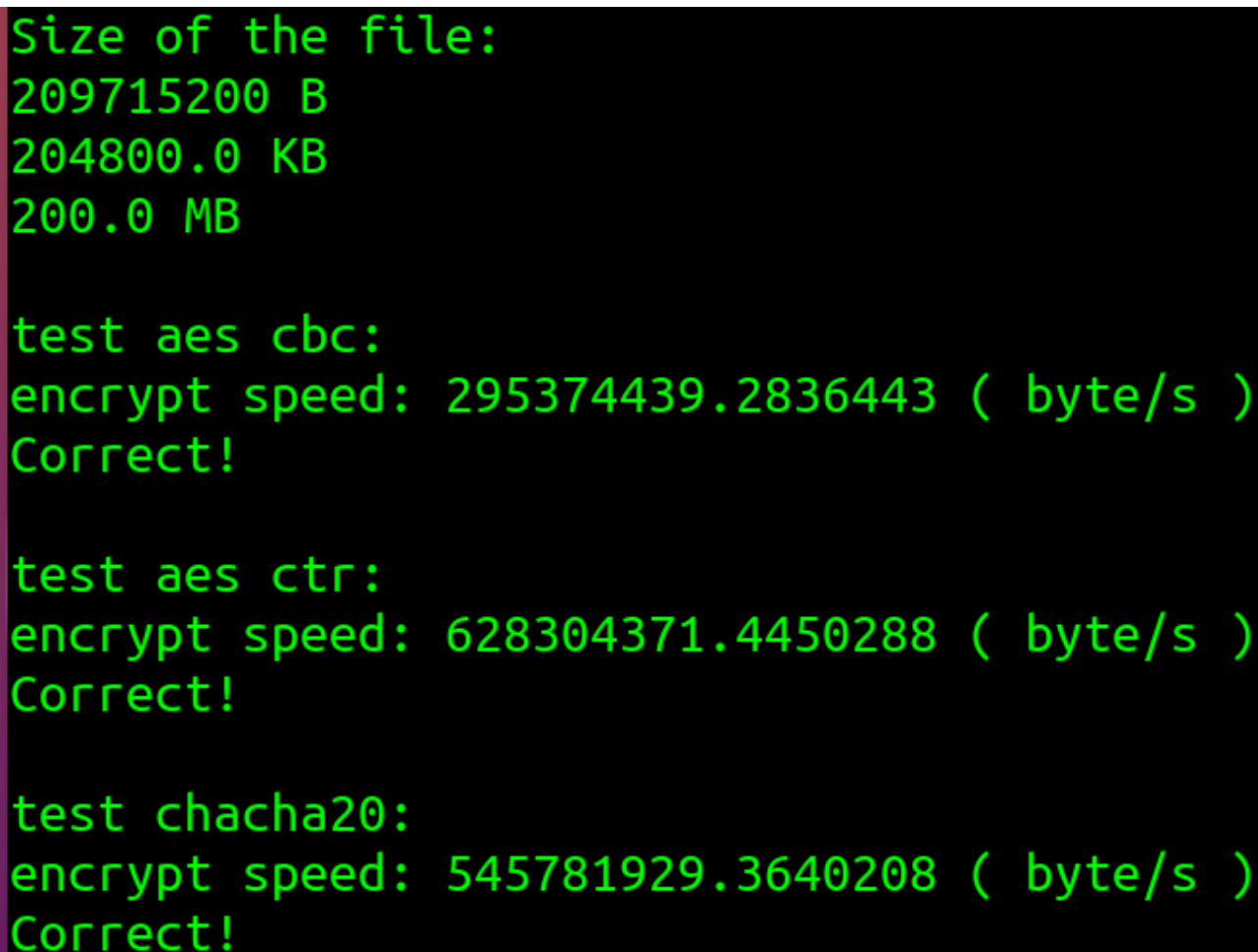
```
$ dd if=/dev/urandom of=random_test_file bs=1M count=200
```

- 執行程式：

```
$ python3 sym_enc.py
```

執行結果

首先印出檔案大小，接著先後執行 AES-CBC mode 加密、AES-CTR mode (counter mode) 加密以及 ChaCha20 加密，計算每種方式的加密速度，並且都執行解密以驗證加密的正確性 (輸出 **Correct!** 代表驗證正確)，以下為結果之截圖



```
Size of the file:
209715200 B
204800.0 KB
200.0 MB

test aes cbc:
encrypt speed: 295374439.2836443 ( byte/s )
Correct!

test aes ctr:
encrypt speed: 628304371.4450288 ( byte/s )
Correct!

test chacha20:
encrypt speed: 545781929.3640208 ( byte/s )
Correct!
```

```

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

def compare_result(s1, s2):
    if s1.__eq__(s2):
        print("Correct!")
    else:
        print("WrongQQ")

# AES CBC ENC & DEC

def aes_cbc_enc(data, key=os.urandom(32), iv=os.urandom(16)):
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
    return encrypted_data, key, iv

def aes_cbc_dec(data, key, iv):
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(data) + decryptor.finalize()
    return decrypted_data[:-decrypted_data[-1]]

# AES CTR ENC & DEC

def aes_ctr_enc(data, key=os.urandom(16), nonce=os.urandom(16)):
    cipher = Cipher(algorithms.AES(key), modes.CTR(nonce),
backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(data) + encryptor.finalize()
    return encrypted_data, key, nonce

def aes_ctr_dec(data, key, nonce):
    cipher = Cipher(algorithms.AES(key), modes.CTR(nonce),
backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(data) + decryptor.finalize()
    return decrypted_data

# chacha20 ENC & DEC

def chacha20_enc(data, key=os.urandom(32), nonce=os.urandom(16)):
    cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None,
backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(data) + encryptor.finalize()
    return encrypted_data, key, nonce

```

```

def chacha20_dec(data, key, nonce):
    cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None,
backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(data) + decryptor.finalize()
    return decrypted_data

# dd if=/dev/urandom of=random_test_file bs=1M count=200
with open('random_test_file', 'rb') as rtf:
    data = rtf.read()

    # size of the file
    print("Size of the file:")
    print(len(data), "B")
    print(len(data) / 1024, "KB")
    print(len(data) / 1024 / 1024, "MB")
    print()

    # test aes cbc
    print("test aes cbc:")
    tic = time.process_time()
    enc, key, iv = aes_cbc_enc(data)
    toc = time.process_time()
    print("encrypt speed: " + str(len(data) / (toc-tic)) + " ( byte/s )")
    dec = aes_cbc_dec(enc, key, iv)
    compare_result(data, dec)
    print()

    # test aes ctr
    print("test aes ctr:")
    tic = time.process_time()
    enc, key, nonce = aes_ctr_enc(data)
    toc = time.process_time()
    print("encrypt speed: " + str(len(data) / (toc-tic)) + " ( byte/s )")
    dec = aes_ctr_dec(enc, key, nonce)
    compare_result(data, dec)
    print()

    # test chacha20
    print("test chacha20:")
    tic = time.process_time()
    enc, key, nonce = chacha20_enc(data)
    toc = time.process_time()
    print("encrypt speed: " + str(len(data) / (toc-tic)) + " ( byte/s )")
    dec = chacha20_dec(enc, key, nonce)
    compare_result(data, dec)
    print()

```