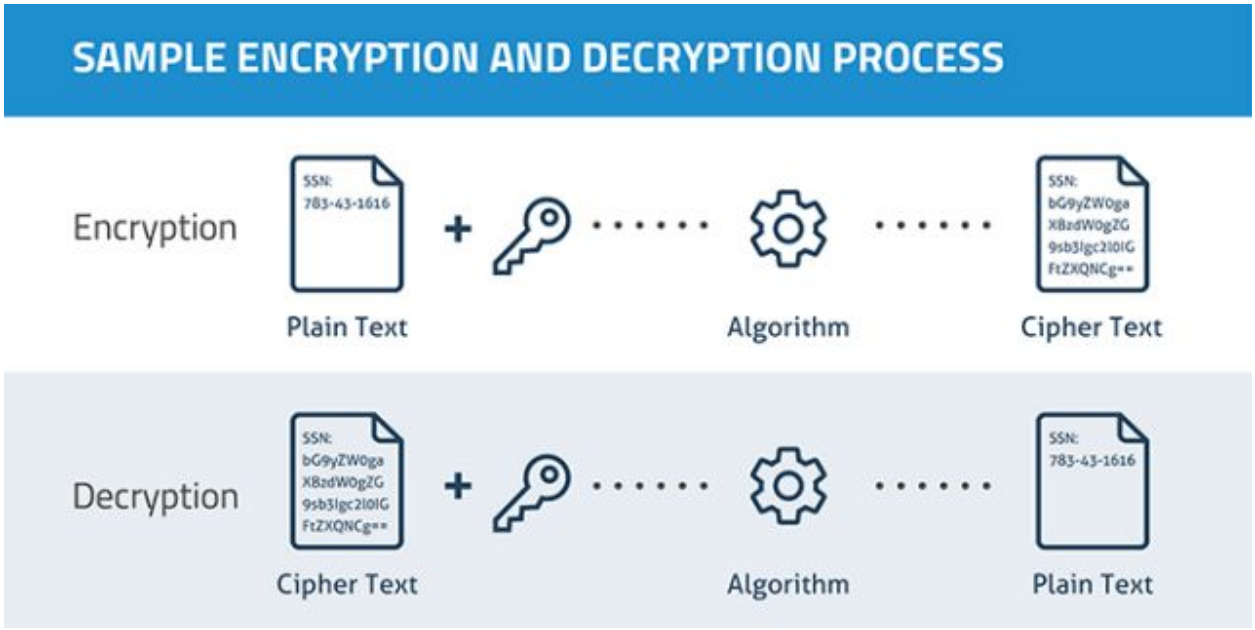
Computer Security

Home work 2

Encryption & Decryption with cryptographic libraries



Introduction

The assignment required me to learn about new cryptographic libraries, how to use the functions provided by those libraries. pycrypto and pycryptodome have been used. Different encryption and decryption methods have been analyzed. The encryption/decryption time and speed is also analyzed.

Index

(a) Create a 128-bit AES key, encrypt and decrypt each of the two files using AES in CBC mode. AES implementations must be based on hardware implementation of AES, so ensure that your libraries are chosen or configured properly.

Code:

```
import os
from Crypto.Cipher import AES
from Crypto import Random
import filecmp
import time

kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

key = Random.get_random_bytes(16)
print("Key: ", key)
IV = Random.get_random_bytes(16)
print("IV: ", IV)

def aesEnc_CBC(filename):
        aes = AES.new(key, AES.MODE_CBC, IV)
        with open(filename, 'rb') as i:
                plaintext = i.read()
                enc = aes.encrypt(plaintext)
                with open("enc.txt", 'wb') as j:
                        j.write(enc)

def aesDec_CBC(filename):
        aes = AES.new(key, AES.MODE_CBC, IV)
        with open(filename, 'rb') as i:
                ciphertext = i.read()
                dec = aes.decrypt(ciphertext)
                with open("dec.txt", 'wb') as j:
                        j.write(dec)

#time in seconds
```

```
Begin = time.time()
aesEnc_CBC("small_file.txt")
End = time.time()
print("Encryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin = time.time()
aesDec_CBC("enc.txt")
End = time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("small_file.txt", "dec.txt"))

Begin = time.time()
aesEnc_CBC('large_file.txt')
End = time.time()
print("Encryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin = time.time()
aesDec_CBC("enc.txt",1)
End = time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("large_file.txt", "dec.txt"))
exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/AES_cbc_1.py

Key:  b'\x0e\xdf\x1f\xc1\xf3\x13c\x03\xc3ZO\xf1yt\xbe\xb2'
Key generation time:  6.389617919921875e-05
IV:  b'\xde\xa6\\\xfc~\x1f\xfe\xcd\xb5\x87\xde\x91\x9ah\x03\x8c'
IV generation time:  9.5367431640625e-06

Encryption time for 1 kb file:  0.0010497570037841797
Encryption speed for 1 kb file:  975463.8419259596 bytes/sec
Decryption time for 10 mb file:  0.00025177001953125
Decryption speed for 1 kb file:  4067203.878787879 bytes/sec
The input file and decrypted file match:  True

Encryption time for 10 mb file:  0.048165321350097656
Encryption speed for 10 mb file:  217703520.00316802 bytes/sec
Decryption time for 10 mb file:  0.038666099029541
Decryption speed for 10 mb file:  271188054.4777619 bytes/sec
The input file and decrypted file match:  True

Process finished with exit code 0
```

(b) Repeat part (a) using AES in CTR mode.

Code:

```
import os
from Crypto.Cipher import AES
from Crypto import Random
import filecmp
from Crypto.Util import Counter
import time

kB = 1024# 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

Begin = time.time()
key = Random.get_random_bytes(16)
print("Key: ", key)
End = time.time()
print("Key generation time: ",End-Begin)

def aesEnc_CTR(filename):
        with open(filename, 'rb') as i:
                data = i.read()
                ctr=Counter.new(128)
                cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
                ct=cipher.encrypt(data)
                with open('enc.txt', 'wb') as j:
                        j.write(ct)

def aesDec_CTR(filename,sz):
        with open("enc.txt", 'rb') as i:
                data = i.read()
                ctr=Counter.new(128)
                cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
                ct=cipher.decrypt(data)
                with open('dec.txt', 'wb') as j:
                        j.write(ct)

#time in seconds
Begin = time.time()
aesEnc_CTR("small_file.txt")
End = time.time()
```

```
print("Encryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin = time.time()
aesDec_CTR("enc.txt",0)
End = time.time()
print("Decryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ",filecmp.cmp("small_file.txt", "dec.txt"))

Begin = time.time()
aesEnc_CTR("large_file.txt")
End = time.time()
print("Encryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin = time.time()
aesDec_CTR("enc.txt",1)
End = time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
         print("Decryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ",filecmp.cmp("large_file.txt", "dec.txt"))
exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/AES_ctr_2.py

Key:  b']\xc5\xaet\x9dr\x8af\xeb\xea\x8e\x88m\xb8-\x19'
Key generation time:  7.295608520507812e-05

Encryption time for 1 kb file:  0.004037380218505859
Encryption speed for 1 kb file:  253629.81551907407 bytes/sec
Decryption time for 1 kb file:  0.004230976104736328
Decryption speed for 1 kb file:  242024.52924602726 bytes/sec
The input file and decrypted file match:  True

Encryption time for 10 mb file:  0.03460860252380371
Encryption speed for 10 mb file:  302981317.80351204 bytes/sec
Decryption time for 10 mb file:  0.03656148910522461
Decryption speed for 10 mb file:  286797946.5995435 bytes/sec
The input file and decrypted file match:  True

Process finished with exit code 0
```

(c) Repeat part (b) with a 256-bit key.

Code:

```
import os
from Crypto.Cipher import AES
from Crypto import Random
import filecmp
from Crypto.Util import Counter
import time

kB = 1024# 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

Begin = time.time()
key = Random.get_random_bytes(32)
print("Key: ", key)
End = time.time()
print("Key generation time: ",End-Begin)


def aesEnc_CTR(filename):
        with open(filename, 'rb') as i:
                data = i.read()
                ctr=Counter.new(128)
                cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
                ct=cipher.encrypt(data)
                with open("enc.txt", 'wb') as j:
                        j.write(ct)

def aesDec_CTR(filename,sz):
        with open("enc.txt", 'rb') as i:
                data = i.read()
                ctr=Counter.new(128)
                cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
                ct=cipher.decrypt(data)
                with open("dec.txt", 'wb') as j:
                        j.write(ct)

#time in seconds
Begin = time.time()
```

```
aesEnc_CTR("small_file.txt")
End = time.time()
print("Encryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")


Begin = time.time()
aesDec_CTR("enc.txt",0)
End = time.time()
print("Decryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ",filecmp.cmp("small_file.txt", "dec.txt"))

Begin = time.time()
aesEnc_CTR("large_file.txt")
End = time.time()
print("Encryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin = time.time()
aesDec_CTR("enc.txt",1)
End = time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ",filecmp.cmp("large_file.txt", "dec.txt"))

exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/AES_ctr_3.py

Key:
b'\xfb\x92\xec.\xa4\xe3g\xc3\x9c\xca\xa1\xa8m\x19\x9e\n\x84\n\xfb\xf6\xaax}m\xde\xacw\xf4\x93
$\xc7\x1f'
Key generation time:  6.008148193359375e-05

Encryption time for 1 kb file:  0.0016484260559082031
Encryption speed for 1 kb file:  621198.6253977437 bytes/sec
Decryption time for 1 kb file:  0.004892587661743164
Decryption speed for 1 kb file:  209296.19882072025 bytes/sec
The input file and decrypted file match:  True

Encryption time for 10 mb file:  0.03234696388244629
Encryption speed for 10 mb file:  324165199.49466735 bytes/sec
Decryption time for 10 mb file:  0.0363156795501709
Decryption speed for 10 mb file:  288739192.81928056 bytes/sec
The input file and decrypted file match:  True

Process finished with exit code 0
```

(d) Compute a hash of each of the files using hash functions SHA-256, SHA-512, and SHA3-256.

Code:

```
import hashlib
import os
import time

kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
f.write(os.urandom(mB))


def SHA256(filename):
        h = hashlib.sha256()
        with open(filename, 'rb') as f:
        block = f.read(h.block_size)
        h.update(block)
        return h.hexdigest()

def SHA512(filename):
        h = hashlib.sha512()
        with open(filename, 'rb') as f:
        block = f.read(h.block_size)
        h.update(block)
        return h.hexdigest()

def SHA3_256(filename):
        h = hashlib.sha3_256()
        with open(filename, 'rb') as f:
        block = f.read(h.block_size)
        h.update(block)
        return h.hexdigest()

Begin=time.time()
s1=SHA256('small_file.txt')
End=time.time()
print("Time taken forSHA256 with 1 kb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
```

```
s2=SHA512('small_file.txt')
End=time.time()
print("Time taken forSHA512 with 1 kb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
s3=SHA3_256('small_file.txt')
End=time.time()
print("Time taken forSHA3_256 with 1 kb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
l1=SHA256('large_file.txt')
End=time.time()
print("Time taken forSHA256 with 10 mb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin=time.time()
l2=SHA512('large_file.txt')
End=time.time()
print("Time taken forSHA512 with 10 mb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin=time.time()
l3=SHA3_256('large_file.txt')
End=time.time()
print("Time taken forSHA3_256 with 10 mb file: ",End-Begin)
if(End-Begin != 0):
        print("Hashing speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

with open('HashDigest.txt','w') as i:
        i.write(s1)
        i.write("\n")
        i.write(s2)
        i.write("\n")
        i.write(s3)
        i.write("\n")
        i.write(l1)
        i.write("\n")
        i.write(l2)
        i.write("\n")
        i.write(l3)

exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/SHA_4.py

Time taken for  SHA256 with 1 kb file:  0.00029468536376953125
Hashing speed for 1 kb file:  3474892.6343042073 bytes/sec
Time taken for  SHA512 with 1 kb file:  0.00044465065002441406
Hashing speed for 1 kb file:  2302931.526005362 bytes/sec
Time taken for  SHA3_256 with 1 kb file:  0.000232696533203125
Hashing speed for 1 kb file:  4400581.24590164 bytes/sec

Time taken for  SHA256 with 10 mb file:  0.00016307830810546875
Hashing speed for 10 mb file:  64298925600.93567 bytes/sec
Time taken for  SHA512 with 10 mb file:  5.7220458984375e-05
Hashing speed for 10 mb file:  183251937962.66666 bytes/sec
Time taken for  SHA3_256 with 10 mb file:  0.0001423358917236328
Hashing speed for 10 mb file:  73669120789.01172 bytes/sec

Process finished with exit code 0
```

(e) Create a 2048-bit RSA key, encrypt and decrypt the files above with PKCS #1 v2 padding (at least v2.0, but v2.2 is preferred if available; it may also be called OAEP).

Code:

```
import os
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP
import time
import filecmp

kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

code = 'FairyTail' #if there is no passphrase, private key is exported in clear
key = RSA.generate(2048)
encrypted_key = key.exportKey(passphrase=code, pkcs=8,protection="scryptAndAES128-CBC")

with open('private_rsa_key.pem', 'wb') as f:
        f.write(encrypted_key)
with open('public_rsa_key.pem', 'wb') as f:
        f.write(key.publickey().exportKey())

def RSA_Enc(filename):
        with open('encrypted_data.txt', 'wb') as f:
        recipient_key = RSA.import_key(open('public_rsa_key.pem').read())
        session_key = get_random_bytes(16)

        cipher_rsa = PKCS1_OAEP.new(recipient_key)
        f.write(cipher_rsa.encrypt(session_key))

        cipher_aes = AES.new(session_key, AES.MODE_EAX)
        data = open(filename,'rb').read()

        ciphertext, tag = cipher_aes.encrypt_and_digest(data)
        f.write(cipher_aes.nonce)
        f.write(tag)
        f.write(ciphertext)
```

```
def RSA_Dec(filename):
        with open('encrypted_data.txt', 'rb') as i:
                private_key = RSA.import_key(open('private_rsa_key.pem').read(),passphrase=code)
                enc_session_key, nonce, tag, ciphertext = [i.read(x) for x in (private_key.size_in_bytes(),
                16, 16, -1)]
                cipher_rsa = PKCS1_OAEP.new(private_key)
                session_key = cipher_rsa.decrypt(enc_session_key)
                cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
                data = cipher_aes.decrypt_and_verify(ciphertext, tag)
                with open('decrypted_data.txt','wb') as j:
                        j.write(data)

Begin=time.time()
RSA_Enc('small_file.txt')
End=time.time()
print("Encryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
RSA_Dec('small_file.txt')
End=time.time()
print("Decryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("small_file.txt", "decrypted_data.txt"))

Begin=time.time()
RSA_Enc('large_file.txt')
End=time.time()
print("Encryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin=time.time()
RSA_Dec('large_file.txt')
End=time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("large_file.txt", "decrypted_data.txt"))

exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/RSA_5.py

key generation time:  3.7069709300994873
Encryption time for 1 kb file:  0.015608787536621094
Encryption speed for 1 kb file:  65604.07062992606 bytes/sec
Decryption time for 1 kb file:  0.6119966506958008
Decryption speed for 1 kb file:  1673.2117713974055 bytes/sec
The input file and decrypted file match:  True

Encryption time for 10 mb file:  0.17244243621826172
Encryption speed for 10 mb file:  60807306.07823293 bytes/sec
Decryption time for 10 mb file:  0.6558933258056641
Decryption speed for 10 mb file:  15986989.937913846 bytes/sec
The input file and decrypted file match:  True

Process finished with exit code 0
```

(f) Repeat part (e) with a 3072-bit key.

Code:

```
import os
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP
import time
import filecmp

kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

code = 'FairyTail' #if there is no passphrase, private key is exported in clear
key = RSA.generate(3072)
encrypted_key = key.exportKey(passphrase=code, pkcs=8,protection="scryptAndAES128-CBC")

with open('private_rsa_key.pem', 'wb') as f:
        f.write(encrypted_key)
with open('public_rsa_key.pem', 'wb') as f:
        f.write(key.publickey().exportKey())

def RSA_Enc(filename):
        with open('encrypted_data.txt', 'wb') as f:
                recipient_key = RSA.import_key(open('public_rsa_key.pem').read())
                session_key = get_random_bytes(16)
                cipher_rsa = PKCS1_OAEP.new(recipient_key)
                f.write(cipher_rsa.encrypt(session_key))
                cipher_aes = AES.new(session_key, AES.MODE_EAX)
                data = open(filename,'rb').read()
                ciphertext, tag = cipher_aes.encrypt_and_digest(data)
                f.write(cipher_aes.nonce)
                f.write(tag)
                f.write(ciphertext)

def RSA_Dec(filename):
        with open('encrypted_data.txt', 'rb') as i:
                private_key = RSA.import_key(open('private_rsa_key.pem').read(),passphrase=code)
                enc_session_key, nonce, tag, ciphertext = [i.read(x) for x in (private_key.size_in_bytes(),
                16, 16, -1)]
                cipher_rsa = PKCS1_OAEP.new(private_key)
```

```
                session_key = cipher_rsa.decrypt(enc_session_key)
                cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
                data = cipher_aes.decrypt_and_verify(ciphertext, tag)
                with open('decrypted_data.txt','wb') as j:
                        j.write(data)

Begin=time.time()
RSA_Enc('small_file.txt')
End=time.time()
print("Encryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
RSA_Dec('small_file.txt')
End=time.time()
print("Decryption time for 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("small_file.txt", "decrypted_data.txt"))

Begin=time.time()
RSA_Enc('large_file.txt')
End=time.time()
print("Encryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Encryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

Begin=time.time()
RSA_Dec('large_file.txt')
End=time.time()
print("Decryption time for 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("Decryption speed for 10 mb file: ",10485760/(End-Begin),"bytes/sec")

print("The input file and decrypted file match: ", filecmp.cmp("large_file.txt", "decrypted_data.txt"))

exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/RSA_6.py

key generation time:  3.4060747623443604

Encryption time for 1 kb file:  0.013727664947509766
Encryption speed for 1 kb file:  74593.89516829344 bytes/sec
Decryption time for 1 kb file:  0.5815393924713135
Decryption speed for 1 kb file:  1760.8437420694806 bytes/sec
The input file and decrypted file match:  True

Encryption time for 10 mb file:  0.14530563354492188
Encryption speed for 10 mb file:  72163478.75981203 bytes/sec
Decryption time for 10 mb file:  0.7455847263336182
Decryption speed for 10 mb file:  14063807.411349865 bytes/sec
The input file and decrypted file match:  True

Process finished with exit code 0
```

(g) Create a 2048-bit DSA key, sign the two files and verify the corresponding signatures. If creating a key takes two parameters, use 224 bits for the exponent sizes. If the hash function algorithm needs to specified separately, use SHA-256.

Code:

```
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256
import os
import time

kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))

Begin = time.time()
key = DSA.generate(2048)
with open("public_key.pem", "wb") as f:
        f.write(key.publickey().export_key())
        f.close()
End = time.time()
print("Key Generation Time: ", End-Begin)

def DSA_2048(filename,key):
        with open(filename, 'rb') as f:
        message = f.read()
        hash_obj = SHA256.new(message)
        signer = DSS.new(key, 'fips-186-3')
        signature = signer.sign(hash_obj)

        # Load the public key
        f = open("public_key.pem", "r")
        hash_obj = SHA256.new(message)
        pub_key = DSA.import_key(f.read())
        verifier = DSS.new(pub_key, 'fips-186-3')
        # Verify the authenticity of the message

        Try:
                verifier.verify(hash_obj, signature)
                print ("The message is authentic.")
        except ValueError:
                print ("The message is not authentic.")
```

```
Begin=time.time()
DSA_2048('small_file.txt',key)
End=time.time()
print("Time taken forDSA_2048 with 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("DSA_2048 speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
DSA_2048('large_file.txt',key)
End=time.time()
print("Time taken forDSA_2048 with 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("DSA_2048 speed for 1 kb file: ",10485760/(End-Begin),"bytes/sec")
exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/DSA_7.py

Key Generation Time:  6.715607166290283
The message is authentic.
Time taken for  DSA_2048 with 1 kb file:  0.17205381393432617
DSA_2048 speed for 1 kb file:  5951.62627659545 bytes/sec
The message is authentic.
Time taken for  DSA_2048 with 10 mb file:  0.44951725006103516
DSA_2048 speed for 1 kb file:  23326713.265344657 bytes/sec

Process finished with exit code 0
```

(h) Repeat part (g) with a 3072-bit DSA key (if the second parameter is required, use 256).

Code:

```
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256
import os
import time

Begin=time.time()
kB = 1024 # 1kB
with open('small_file.txt', 'wb') as f:
        f.write(os.urandom(kB))

mB = 10485760 # 1GB
with open('large_file.txt', 'wb') as f:
        f.write(os.urandom(mB))
End=time.time()
print("File Generation Time: ", End-Begin)

Begin=time.time()
key = DSA.generate(3072)
with open("public_key.pem", "wb") as f:
        f.write(key.publickey().export_key())
        f.close()
End=time.time()
print("Key Generation Time: ", End-Begin)

def DSA_2048(filename,key):
        with open(filename, 'rb') as f:
                message = f.read()
                hash_obj = SHA256.new(message)
                signer = DSS.new(key, 'fips-186-3')
                signature = signer.sign(hash_obj)
                 # Load the public key
                 f = open("public_key.pem", "r")
                hash_obj = SHA256.new(message)
                pub_key = DSA.import_key(f.read())
                verifier = DSS.new(pub_key, 'fips-186-3')
                # Verify the authenticity of the message
                Try:
                        verifier.verify(hash_obj, signature)
                        print ("The message is authentic.")
                except ValueError:
                        print ("The message is not authentic.")
```

```
Begin=time.time()
DSA_2048('small_file.txt',key)
End=time.time()
print("Time taken forDSA_2048 with 1 kb file: ",End-Begin)
if End-Begin != 0:
        print("DSA_2048 speed for 1 kb file: ",1024/(End-Begin),"bytes/sec")

Begin=time.time()
DSA_2048('large_file.txt',key)
End=time.time()
print("Time taken forDSA_2048 with 10 mb file: ",End-Begin)
if End-Begin != 0:
        print("DSA_2048 speed for 1 kb file: ",10485760/(End-Begin),"bytes/sec")

exit()
```

Output:

```
/root/PycharmProjects/untitled/venv/bin/python
/home/malinian/Desktop/pycharm-community-2019.2.2/bin/DSA_8.py
File Generation Time:  0.3818848133087158
Key Generation Time:  6.969684362411499
The message is authentic.
Time taken for  DSA_2048 with 1 kb file:  0.44915771484375
DSA_2048 speed for 1 kb file:  2279.822802011143 bytes/sec
The message is authentic.
Time taken for  DSA_2048 with 10 mb file:  0.6394944190979004
DSA_2048 speed for 1 kb file:  16396953.103659114 bytes/sec

Process finished with exit code 0
```

Analysis

1. How per byte speed changes for different algorithms between small and large files.

	Small file (1 kb) byte speed	Large file (10 mb)byte speed
AES _CBC (encryption)	975463.8419259596 bytes/sec	217703520.00316802 bytes/sec
AES _CBC (decryption)	4067203.878787879 bytes/sec	271188054.4777619 bytes/sec
AES_CTR (encryption)	253629.81551907407 bytes/sec	302981317.80351204 bytes/sec
AES_CTR (decryption)	242024.52924602726 bytes/sec	286797946.5995435 bytes/sec
RSA (encryption)	65604.07062992606 bytes/sec	60807306.07823293 bytes/sec
RSA (decryption)	1673.2117713974055 bytes/sec	15986989.937913846 bytes/sec
SHA	3474892.6343042073 bytes/sec	64298925600.93567 bytes/sec
DSA	5951.62627659545 bytes/sec	23326713.265344657 bytes/sec

- The Encryption/Decryption speed increases as the size of the file increases
- AES_CBC has highest encryption/decryption byte speed(encryption or decryption speed)
- Hashing is very fast and has huge byte/sec rate

2. How encryption and decryption times differ for a given encryption algorithm.

	Small file (1 kb) (time is in secs)	Large file (10 mb) (time is in secs)
AES _CBC (encryption)	0.0010497570037841797	0.048165321350097656
AES _CBC (decryption)	0.00025177001953125	0.0386660099029541
AES_CTR (encryption)	0.004037380218505859	0.03460860252380371
AES_CTR (decryption)	0.004230976104736328	0.03656148910522461
RSA (encryption)	0.015608787536621094	0.17244243621826172
RSA (decryption)	0.6119966506958008	0.6558933258056641

- The Encryption/Decryption time increases as the size of the file increases
- AES_CBC has lowest encryption/decryption time
- RSA has highest encryption/decryption time

3. How key generation, encryption, and decryption times differ with the increase in the key size.

Key Generation

	Time in secs
AES _CBC	6.389617919921875e-05
AES_CTR	7.295608520507812e-05
RSA	3.7069709300994873
DSA	6.715607166290283

encryption, and decryption times with the increase in the key size

	Encryption time (Large file)	Decryption time (Large file)
AES_CTR (128 bit)	0.03460860252380371	0.03656148910522461
AES_CTR (256 bit)	0.03234696388244629	0.0363156795501709
RSA (2048 bit)	0.17244243621826172	0.6558933258056641
RSA (3072 bit)	0.14530563354492188	0.7455847263336182
DSA (2048 bit)	0.44951725006103516 *For msg authentication*	-
DSA (3072 bit)	0.6394944190979004 *For msg authentication*	-

4. How hashing time differs between the algorithms and with increase of the hash size

	Time in secs (small file)	TIme in secs (large file)
SHA256	0.0002946853637695312 5	0.0001630783081054687 5
SHA512	0.0004446506500244140 6	5.7220458984375e-05
SHA3_256	0.000232696533203125	0.0001423358917236328

- SHA3_256 is fastest out of the three SHA()

5. How performance of symmetric key encryption (AES), hash functions, and public-key encryption (RSA) compare to each other.

- AES is the fastest and cost and time efficient while RSA is not costor time efficient
- AES is a block cipher while RSA is a stream cipher. Thus, it's harder to implement RSA if the data is large. The RSA is used to encrypt key rather than the file.
- AES's performance is better than RSA in certain aspects.

References

AES_CBC

https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html

AES_CTR

https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#ctr-mode

RSA

https://dzone.com/articles/an-intro-to-encryption-in-python-3

SHA

https://www.geeksforgeeks.org/sha-in-python/

DSA

https://pycryptodome.readthedocs.io/en/latest/src/public_key/dsa.html