Week 12 Homework due on Friday Nov 30, 22:00 Hour

Group 5

Wong Ann Yi (1004000)
Liu Bowen (1004028)
Tan Chin Leong Leonard (1004041)

Exercise 1

Design and implement a simple digital certificate framework. Your framework should allow to create a certificate chain and validate it.

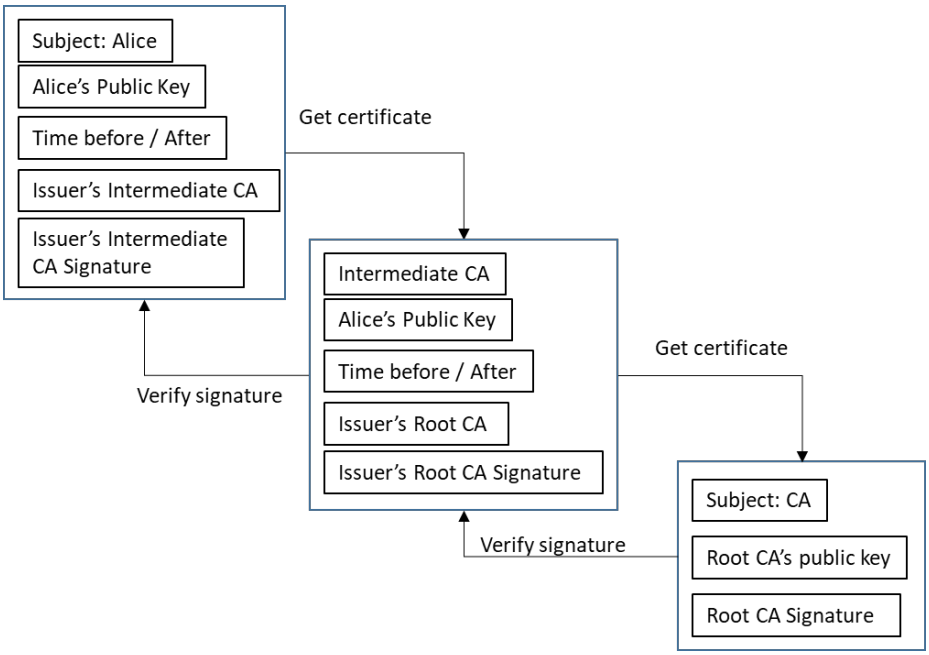
Answers:

We will create scenario to illustrate the process of key creation and certification by an intermediate CA and the Root CA to complete a certificate framework which include a chain validation.

During creation, Alice first creates a public/private key pair (public key A & private key A') and stores the private key in a secure manner. In the next phase of certification, Alice takes the public key to a chain of intermediate CAs for certification. This process includes that each certificate in the chain is signed by the sub-CA identified by the next certificate in the chain.

In this scenario, we assume there is one intermediate CA between Alice and the Root CA. The intermediate CA is the issuer / signer of Alice's public key certificate while the Root CA is the issuer / signer of the intermediate certificate. The Root CA is always signed by the CA itself. The signatures of all the certificates in the chain must be verified all the way up to the Root CA certificate.

The below figure illustrates the certification path from Alice to the Intermediate CA and to the Root CA and back to Alice. She can then connect with Bob and transmit her public key to him. Bob will verify her certificate because he trusts the CA's certificate. The date structure in the certificate conforms to the X.509 standard.



The codes in Python is written as per below.

```
from Cryptodome.Cipher import AES
import hashlib
from Crypto.Hash import SHA256
from Crypto.Signature import PKCS1_v1_5
from Crypto.PublicKey import RSA
import time

class RootCA(object):
   def _init_(self):
      self.key = RSA.generate(1024)
      self.publicKey = self.key.publickey()

   def generateCertiChain(self, public_key, intermedia):
      owner = 'IntermediaCA'
      issuer = 'RootCA'
      not_before = int(time.time())
      not_after = int(time.time() + 10*365*24*60*60)
      publicKey = public_key
      message = owner+issuer+str(not_before)+str(not_after)
      hash_message = SHA256.new(message)
      signature = PKCS1_v1_5.new(self.key).sign(hash_message)
      return owner, issuer, not_before, not_after, publicKey, signature

   def applyCertificateChain(self, message, public_key, signature):
      hash_message = SHA256.new(message);
      verified = PKCS1_v1_5.new(public_key).verify(hash_message, signature)
      if verified == True:
```

```
        return self.generateCertiChain(public_key);

    def getPublicKey(self):
        return self.publicKey;

class IntermediaCA(object):
    def _init_(self):
        self.key = RSA.generate(1024)
        self.publicKey = self.key.publickey()
        self.CAdic = {}

    def getPKIKey(self, name, PKI_object):
        key = PKI_object.getPublicKey();
        self.CAdic[name] = key;

    def getPublicKey(self):
        return self.publicKey;

    def generateCertiChain(self, public_key):
        owner = 'Alice'
        issuer = 'IntermediaCA'
        not_before = int(time.time())
        not_after = int(time.time() + 10*365*24*60*60)
        publicKey = public_key;
        message = owner+issuer+str(not_before)+str(not_after)
        hash_message = SHA256.new(message)
        signature = PKCS1_v1_5.new(self.key).sign(hash_message)
        return owner, issuer, not_before, not_after, publicKey, signature

    def applyToRootCA(self, message, intermedia):
        hash_message = SHA256.new(message);
        signature = PKCS1_v1_5.new(self.publicKey ).sign(hash_message)
        return intermedia.applyCertificateChain(message, self.publicKey, signature)

    def applyCertificateChain(self, message, public_key, signature, intermedia):
        hash_message = SHA256.new(message);
        verified = PKCS1_v1_5.new(public_key).verify(hash_message, signature)
        if verified == True:
            return self.generateCertiChain(public_key);


class Alice(object):
    def _init_(self):
        self.key = RSA.generate(1024)
        self.publicKey = self.key.publickey()
        self.CAdic = {}

    def getPKIKey(self, name, PKI_object):
```

```
        key = PKI_object.getPublicKey();
        self.CAdic[name] = key;

    def applyCertificateChain(self, message, PKI_object):
        hash_message = SHA256.new(message)
        signature = PKCS1_v1_5.new(self.key).sign(hash_message)
        return PKI_object.applyCertificate(message, self.publicKey, signature)

    def sendToBob(self, certificate, send_objetc):
        return send_objetc.BobVerify(certificate)

class Bob(object):
    def _init_(self):
        self.key = RSA.generate(1024)
        self.publicKey = self.key.publickey();
        self.CAdic = {}

    def getPKIKey(self, name, PKI_object):
        key = PKI_object.getPublicKey();
        self.CAdic[name] = key;

    def applyCertificate(self, message, PKI_object):
        hash_message = SHA256.new(message)
        signature = PKCS1_v1_5.new(self.key).sign(hash_message)
        message, signature = PKI_object.applyCertificate(message,self.publicKey, signature)
        return message, signature

    def BobVerify(self, certificate):
        for cert in certificate:
            if time.time() < cert[2] or time.time() > cert[3]:
                return False;
            message = cert[0]+cert[1]+str(cert[2])+str(cert[3]);
            hash_message = SHA256.new(message);
            verified = PKCS1_v1_5.new(self.CAdic[cert[1]]).verify(hash_message, cert[5])
            if verified == False:
                return False;
        return True;

alice = Alice()
bob = Bob()
rootca = RootCA()
intermediaca = IntermediaCA()

alice.getPKIKey('RootCA', rootca)
alice.getPKIKey('IntermediaCA', intermediaca)
bob.getPKIKey('RootCA', rootca)
bob.getPKIKey('IntermediaCA', intermediaca)
```

```
ceriFromIntermedia = alice.applyCertificateChain('I am Alice!', intermediaca)
ceriFromRoot = intermediaca.applyToRootCA('I am intermedia!', rootca)

certificateChian = [ceriFromIntermedia, ceriFromRoot]

alice.sendToBob(certificateChian, bob)
bob.BobVerify(certificateChian)
```

Below are the certificates generated by the Intermediate CA and the Root CA.

Certificate From Intermediate CA:

Owner: 'Alice'
Issuer: 'IntermediaCA'
Not_before: 1543585448
Not_after: 1858945448(Not_before+10 years, 10*365*24*60*60)
Public_key(Alice's public key):
+GnigO0i6QOHpVi1HQfMIEHnXGZP4Q5eLm38svzxcZj2chqoUHwY77
GckIclvTSyUc9MAt1PKm5dqEoQ9XtpOxYix++J8uwHdJIVIo8C1S3rIkyC5IaPdM
5n0zL0iveOA0CotRKPyOB87AsVSVhb7cWN5lZsfUE//ivPnwnncISoB0VT3u74CL
mwIDAQAB
signature:

```
.@R 'M k @@5w@@ @6x@#@M@|&@bb@'TSY@:@@@+@@<@Z3)@>@@@&S@!
@(@@B@@^@O i@ @@u +@K@@@@@m@+D}@i@@J @@@&@ @?,@ !
K@@@43@@@@@@
```

Certificate from the Root CA

Owner: 'IntermediaCA
Issuer: 'RootCA'
Not_before: 1543585567
Not_after: 1858945567 (Not_before+10 years, 10*365*24*60*60)
Public_key(IntermediaCA's public key):
lYrhZj74ZCE8+6Ep9bpj0+lQg6ck7eAGOoyVbr2VtSf2KYyNdYQJBAO8F
MlJyCCM8F2yDA+FxjG5vuUDBo/6t+YUfEIW+1kkdvT1eyByEVwu7VnzOmMtKJ/wz
UQXs948m3ZhaRKpXcn0CQEtl1nvaeHebMYIqxpaiuBSYSy+lPGXpuVoGXxG5pRSn
signature:

```
@a@z@@8) o@@¿@@@;@y @7@ @@@Jh@z:5@@a@@z□}Y
@@ @ @K@@*@)#@@@7@6U@@@^Ik $@@y@Z@"{$@@@p@>@@7@-
@F@@ZJ@ @@Kz@*@L@ @`Yb@Ifm@@
```

```
The certificate authentication is: True
```