[과제2] PKI 시뮬레이션

- 컴퓨터시스템보안 -

1분반 20180018 강 승아

[문제]

PKI 시뮬레이션을 실행하는 Python 코드를 작성하며, 각 단계에서의 핵심적 함수와 파라미터, 반환 값을 함께 작성한다.

1. 코드내용

(1) 인증서를 생성하는 함수 genCertificate(myPubKey, CAPrivKey)

```python
# 1) 인증서를 만드는 함수 genCertificate
def genCertificate(myPubKey, CAPrivKey):
    h = SHA256.new(str(myPubKey.export_key('PEM')).encode('utf-8'))
    S = pkcs1_15.new(CAPrivKey).sign(h)
    return [myPubKey.export_key('PEM'), S]
```

함수의 인자 :

- 인증서를 생성하려는 주체의 공개키(myPubKey)
- 인증기관인 CA의 개인키(CAPrivKey)

함수의 내용 :

- h = 주체의 공개키에 SHA256를 적용한다. key 값이지만 str로 불러와 인코딩을 한다.
- S = h값을 CA의 개인키를 사용해 RSA서명을 한 signature 값이다.
- 반환 값 = list형태로, 인증서 주체의 공개키와 주체의 서명 값

(2) 인증서로 인증서를 검증하는 함수 veriCertificate

```python
# 2) 인증서로 인증서를 검증하는 함수 veriCertificate
def veriCertificate(aCertificate, CACertificate):
    CA = RSA.import_key(CACertificate[0])
    try:
        pkcs1_15.new(CA).verify(SHA256.new(str(aCertificate[0]).encode('utf-8')), aCertificate[1])
        print("인증서 검증에 성공 하였습니다!")
    except (ValueError, TypeError):
        print("인증서 검증에 실패 하였습니다.")
        exit()
```

함수의 인자 :

- 검증될 인증서 소유자의 공개키(aCertificate)
- 검증될 인증서를 서명한 개인키에 대응되는 검증할 인증기관인 CA의 인증서(CAprivKey)

함수의 내용 :

- 인증기관의 RSA 키 값으로 verify하며 검증될 인증서의 키의 값은 인코딩을 하기위해 str형태로 형변환을 해 해시한다. 서명과 함께 검증된다. 성공할 경우 성공 출력, 실패할 경우 프로그램을 종료

(a) CA의 RSA 개인키를 만들어 CAPriv.pem 파일에 저장

```
# a) CA의 개인키를 만들어 CAPriv.pem에 저장한다.
CAPriv = RSA.generate(2048)
f = open('CAPriv.pem', 'wb')
f.write(CAPriv.export_key('PEM', passphrase="!@#$"))
f.close()
```

- CAPriv의 변수에 RSA의 generate 함수를 통해 키 값을 생성한다.
- 파일 CAPriv.pem을 binary형으로 작성할 수 있도록 생성한다.
- CAPriv의 키 값은 passphrase로 암호화 하여 작성된다.

(b) CA의 개인키로부터 공개키를 추출해 CAPub.pem 파일에 저장

```
# b) CA의 RSA 개인키에서 공개키 CA_pub를 추출한다. 이를 파일 CAPub.pem에 저장한다.
f = open('CAPub.pem', 'wb')
f.write(CAPriv.publickey().export_key('PEM'))
f.close()
```

- 파일 CAPub.pem을 binary형으로 작성할 수 있도록 생성한다.
- 파일에 CA의 비밀키로부터 공개키를 추출해 작성한다.

(c) CA의 인증서를 작성해 CACertCA.plk 파일에 저장

```
# c) CA는 자신의 공개키에 SAH256을 적용하고, 자신의 개인키로 서명하여 서명 S_CA를 만들고,
# 이를 이용하여 자신의 root 인증서 [CA_pub, S_CA]를 만들어 CACertCA.plk 파일에 저장한다.
# 인증서의 저장은 pickle.dump()를 쓰고, 인증서를 읽는 것은 pickle.load()를 쓴다.
f = open('CAPub.pem', 'r')
CAPub = RSA.import_key(f.read())
f.close()
f = open('CAPriv.pem', 'r')
CAPriv = RSA.import_key(f.read(), passphrase="!@#$")
f.close()
```

- 작성된 CAPub.pem 파일로부터 공개키 값을 읽어와 CAPub 변수에 저장 한다.
- 작성된 CAPriv.pem 파일로부터 비밀키 값을 읽어와 CAPriv 변수에 저장 한다.
- 비밀키 값은 passphrase로 암호화 된다.
- 읽어온 값들은 인증서 작성을 위해 사용된다.

```
# 공개키에 SHA256 적용, 개인키 서명으로 S_CA를 만듬
root = genCertificate(CAPub, CAPriv)
```

- 구현해둔 인증서 작성 함수 genCertificate에 CA의 공개키와 비밀키를 전달해 인증서를 작성한다.
- 작성된 인증서는 root에 저장된다.

```
f = open('CACertCA.plk', 'wb')
pickle.dump(root, f)
f.close()
```

- 인증서를 저장하기 위한 파일 CACertCA.plk을 binary형으로 작성할 수 있도록 생성한다.
- pickle 파일이므로 dump()를 통해 인증서 값인 root를 저장한다.

(d) Bob의 RSA 개인키를 생성해 BobPriv.pem에 저장

```
# d) Bob은 자신의 RSA 개인키를 만든다. 이를 파일 BobPriv.pem에 저장한다.
BobPriv = RSA.generate(2048)
f = open('BobPriv.pem', 'wb')
f.write(BobPriv.export_key('PEM', passphrase="!@#$"))
f.close()
```

- Bob의 개인키가 될 BobPriv 변수에 RSA 키를 생성한다.
- Bob의 개인키가 입력될 파일 BobPriv.pem을 binary형으로 작성할 수 있도록 생성한다.
- 비밀키 값은 passphrase로 암호화 된다.

(e) Bob의 개인키로부터 공개키를 추출하여 BobPub.pem에 저장

```
# e) Bob은 개인키에서 공개키 Bob_pub를 추출하여 파일 BobPub.pem에 저장한다.
f = open('BobPub.pem', 'wb')
f.write(BobPriv.publickey().export_key('PEM'))
f.close()
```

- Bob의 공개키가 입력될 파일 BobPub.pem을 binary형으로 작성할 수 있도록 생성한다.
- Bob의 개인키로부터 공개키를 추출하여 파일에 저장한다.

(f) CA가 CA의 개인키로 서명한 Bob의 공개키 인증서를 작성해 BobCertCA.plk에 저장

```
# f) CA는 자신의 개인키로 서명한 Bob의 공개키 인증서[Bob_pub, S_Bob_CA] 를 만들어 BobCertCA.plk에 저장
f = open('BobPub.pem', 'r')          # Bob의 공개키 불러오기
BobPub = RSA.import_key(f.read())
f.close()
```

- 공개키가 작성된 파일인 BobPub.pem을 읽어 저장한다.
- BobPub 변수에 Bob의 RSA 공개키를 저장한다.

```
BobCertCA = genCertificate(BobPub, CAPriv)

f = open('BobCertCA.plk', 'wb')
pickle.dump(BobCertCA, f)
f.close()
```

- 인증서 작성 함수에 Bob의 공개키와 CA의 비밀키를 전달한다.
- 반환된 인증서를 BobCertCA 변수에 넣는다.
- 파일 BobCertCA.plk을 binary형으로 작성할 수 있도록 생성 후 읽었던 변수를 dump()로 넣는다.

(g) Bob이 작성한 메시지를 SHA256으로 해시 후 Alice에게 서명과 메시지, 인증서를 전송

```
# g) Bob은 M = "I bought 100 doge coins." 메시지에 SHA256을 적용한 후
# 자신의 개인키로 서명한 서명 S, 메시지 M, 그리고 공개키 인증서 [Bob_pub, S_Bob_CA]를
# Alice에게 보낸다. (보냈다고 가정하고 print로 출력한다.)
M = SHA256.new('I bought 100 doge coins'.encode('utf-8'))
S = pkcs1_15.new(BobPriv).sign(M)
print("Bob sent (", M, S, BobCertCA, ") to Bob.")
```

- 'I bought 100 doge coins' 메시지를 SHA256로 해싱 후에 M 변수에 저장한다.
- 저장된 해싱 메시지에 Bob의 개인키로 서명을 하여 S에 저장한다.
- 서명 S, 메시지 M, Bob의 공개키 인증서를 Alice에게 전송한다. (가정)

(h) Bob이 보낸 메시지를 Alice가 수신

```
# h) Alice는 메시지 [M, S, [Bob_pub, S_Bob_CA]]를 받는다.
print("Alice received message (", M, S, BobCertCA, ")from Alice.")
```

- Bob으로부터 Alice는 서명 S, 메시지 M, Bob의 공개키 인증서를 전달 받는다. (가정)

(i) CA의 root 인증서를 CA의 root 인증서로 검증하기 위해 읽음

```
# i) Alice는 Bob의 공개키 인증서를 검증하기 위해 CA의 root 인증서 [CA_pub, S_CA]를 파일 CACertCA.plk에서 읽는다.
f = open('CACertCA.plk', 'rb')
rootRead = pickle.load(f)
f.close()
```

- CACertCA.plk 파일을 binary로 f값에 읽는다.
- plk 파일이므로 load()함수를 통해 파일을 읽는다.
- rootRead에 읽어온 인증서를 저장한다.

(j~k) 인증서 검증 및 성공여부 출력

```
# j)  CA의 root 인증서를 CA의 root 인증서로 검증한다. 검증 실패의 경우 오류 메시지를 출력하고 종료한다.
veriCertificate(rootRead, rootRead)
# k) Bob의 인증서를 CA의 root 인증서로 검증한다. 검증 실패의 경우 오류 메시지를 출력하고 종료한다.
veriCertificate(BobCertCA, rootRead)
```

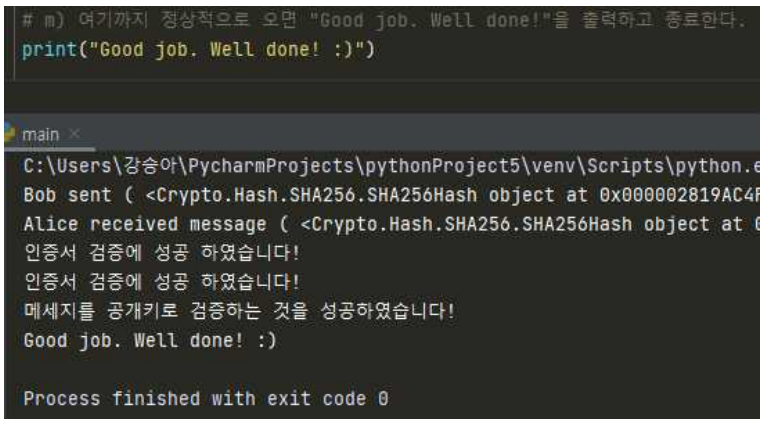
- 작성된 인증서 검증 함수 veriCertificate를 통해 각각을 검증한다.
- 인증서 검증 함수를 통해 검증 성공 여부를 출력하게 된다.

(l) 메시지 [M, S]를 Bob의 인증서에 있는 공개키로 검증 후 성공여부 출력

```
# l) 메시지 [M, S]를 Bob의 인증서에 있는 공개키로 검증한다. 검증 실패의 경우 오류 메시지를 출력하고 종료한다.
try:
    pkcs1_15.new(BobPub).verify(M, S)
    print("메세지를 공개키로 검증하는 것을 성공하였습니다!")
except (ValueError, TypeError):
    print("메세지를 공개키로 검증하는 것을 실패하였습니다.")
    exit()
# m) 여기까지 정상적으로 오면 "Good job. Well done!"을 출력하고 종료한다.
print("Good job. Well done! :)")
```

- pkcs1_15를 통해 Bob의 공개키인 BobPub을 사용하여 M, S를 검증한다.
- 검증 오류를 판별하여 성공과 실패에 따라 다른 문자열을 출력한다.
- 검증 실패시에는 exit() 함수가 호출되어 프로그램이 종료된다.
- 모든 함수가 정상적으로 작동 시에는 프로그램이 종료되지 않는다.
- 중간에 종료되지 않고 프로그램이 정상 종료 될 때 “Good job. Well done! :)“ 문자열이 출력된다.

2. 실행 스냅샷



[오류 없이 작동 후 문자열 출력 화면]

3. 결론 및 느낀 점

처음 작성할 때에는 자료형에 관한 오류가 많이 발생하였었다. 같은 수업을 듣는 학우 분들의 질의응답을 통해 도움을 받았으며, 키 값을 사용하기 위해서 export_key와 import_key, str로의 형변환 등을 통해 오류를 해결하였다. RSA 강의를 듣던 때에 작성하였던 코드를 참고하였고 코드를 하나하나 차분히 따라가며 Python에 보안과 관련된 모듈들이 생각보다 잘 갖추어져 있었다고 생각했다. 사용을 해보는 것이 이번 과목이 처음이라 그럴 것이다. 오류들을 수정해나가며 코드를 완성시키는 것이 재밌었던 것 같다.

4. 참고 문헌

- 금오공과대학교 최태영 교수님 컴퓨터시스템보안 (7주차 강의 RSA1, RSA2)
- 금오공과대학교 최태영 교수님 컴퓨터시스템보안 (12주차 강의 키 관리 - 공개키 배분)
- 금오공과대학교 최태영 교수님 컴퓨터시스템보안 (질의응답)