

# REPORT

## 디지털 서명



과목명 :		컴퓨터보안			
담당교수 :		정준호		교수님	
제출일 :	2021	년 05	월 27	일	
공과	대학	컴퓨터공학			과
학번 :	2016112154	이름:		정동구	

# 1. 전자 서명

디지털 서명은 문서에 하는 인감 날인이나 사인에 해당하는 기능을 컴퓨터로 구현한 것으로 파일의 변조를 검출 할 수 있고 부인을 방지 할 수 있다. 디지털 서명은 크게 두가지 과정을 거친다. 첫째로 메시지의 서명을 작성하는 행위, 둘째로 메시지의 서명을 검증하는 행위이다.

디지털 서명은 공개키 암호를 기반으로 작성이 된다. 서명을 작성하는 행위는 다음과 같이 이루어 진다. 우선 암호화 할 메시지를 일방향 해시함수를 이용하여 해싱한다. 이후 해싱된 결과를 공개키 암호의 개인 키를 이용하여 암호화한다. 이렇게 생성된 것이 서명이다. 해당 서명을 수신자에게 보낼 때, 서명과 함께 해싱된 결과를 함께 보낸다. 수신자는 공개키를 이용하여 암호화된 해시값을 복호화 한다. 이후 같이 받은 해시 값과 비교하여 일치 할 경우 메시지가 변조되지 않음을 확인 할 수 있다.

디지털 서명은 기밀성을 실현하기 위해 암호화를 행하는 것이 아니므로 암호화를 별도로 행해서 보내야 한다. 또한 디지털 서명은 문서의 변조를 방지하는 것이 아닌 변조를 검출하는 것이므로, 변조 방지를 위해서는 다른 조치를 취하여야 한다.

디지털 서명이 메시지 인증코드와 가지는 차이점은 메시지 인증코드에서는 불가능한 부인방지가 디지털 서명에서는 가능하다. 메시지 인증코드의 경우 공유키를 송신자와 수신자 모두가 소유하고 있기 때문에 어느 쪽 에서라도 MAC 를 계산할 수 있다. 반면 디지털 서명의 경우 서명을 작성하는데 필요한 개인 키는 송신자만 소유하고 있기 때문에 송신자가 부인하는 것을 방지하는 것이 가능하다.

## 2. 코드 및 결과

```
from
OpenSSL.crypto
import
load_privatekey,
FILETYPE_PEM,
sign,
load_publickey,
verify, X509

from Crypto.Hash import SHA256 as SHA
from Crypto.Cipher import AES
from OpenSSL import crypto
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox
import sys
pkey = crypto.PKey()
pkey.generate_key(crypto.TYPE_RSA, 1024) # RSA 형식의 키 생성
# 공개키, 개인키 생성
#with open("public.pem", 'ab+') as f:
#    f.write(crypto.dump_publickey(crypto.FILETYPE_PEM, pkey))
#with open("private.pem", 'ab+') as f:
#    f.write(crypto.dump_privatekey(crypto.FILETYPE_PEM, pkey))
# 공개키, 개인키 읽어오기
with open("private.pem", 'rb+') as f:
    priv_key = crypto.load_privatekey(crypto.FILETYPE_PEM, f.read())
with open("public.pem", 'rb+') as f:
    pub_key = crypto.load_publickey(crypto.FILETYPE_PEM, f.read())
# 데이터 해시화
content = '_message' # 가져와야 하는것
_hash = SHA.new(content.encode('utf-8')).digest() # 데이터 해시화

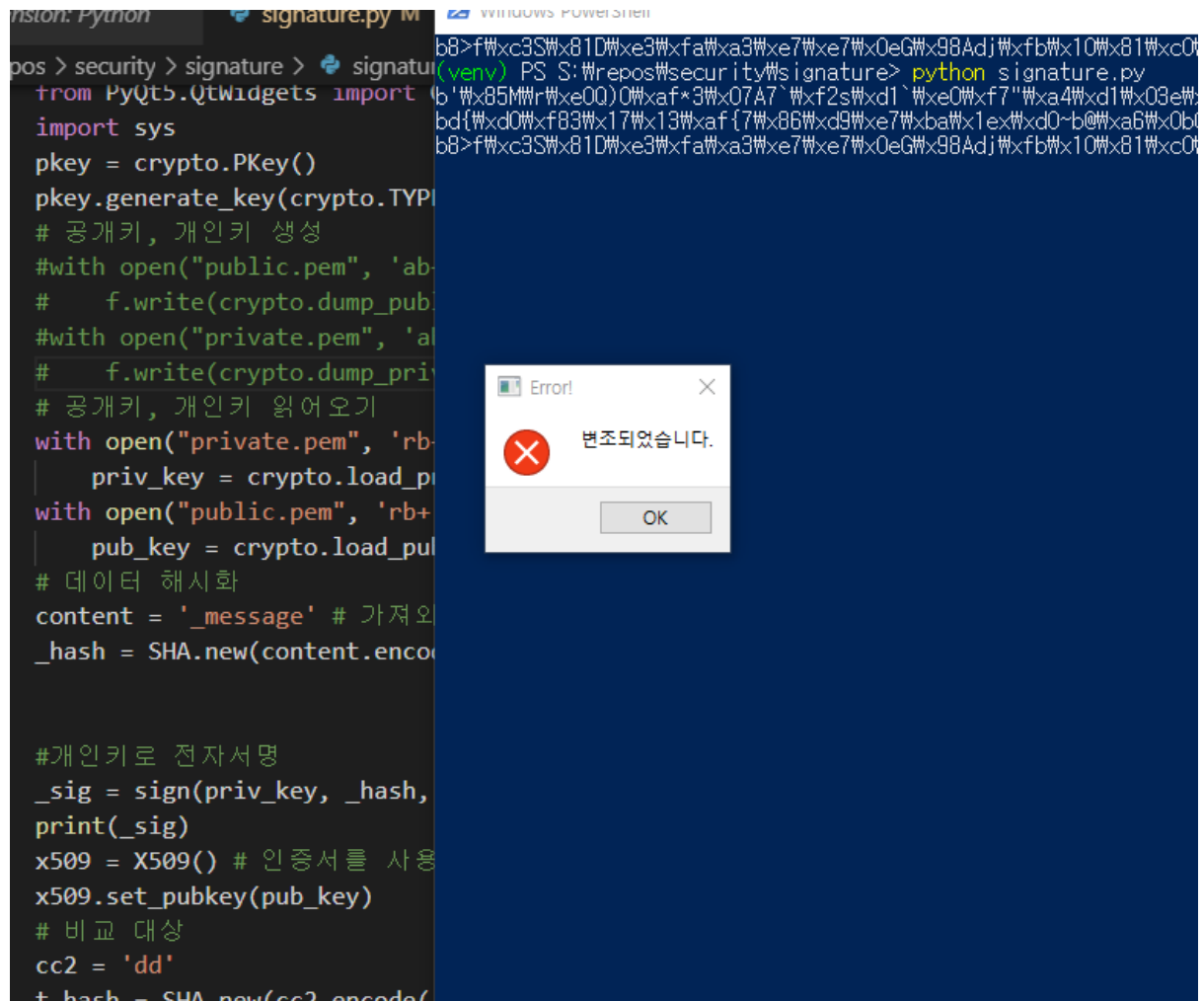
#개인키로 전자서명
_sig = sign(priv_key, _hash, 'sha256') # 개인키로 해시화 되어 있는
데이터의 전자서명 생성
print(_sig)
x509 = X509() # 인증서를 사용할 수 있도록 하는 메서드 제공
x509.set_pubkey(pub_key)
# 비교 대상
cc2 = 'HI'
t_hash = SHA.new(cc2.encode('utf-8')).digest()
# 사용법: verify(x509, 전자서명, 비교할 해시, 'sha256')
try:
```

```

app = QtWidgets.QApplication(sys.argv)
QMessageBox.information(None, 'Success!', u"변조되지않았습니다.",
QMessageBox.Ok)
sys.exit(1)
except:
app = QtWidgets.QApplication(sys.argv)
QMessageBox.critical(None, 'Error!', u"변조되었습니다.",
QMessageBox.Ok)
sys.exit(1)

```

프로그램의 전체적 코드는 실습 예제 코드를 이용하였다. 파이썬의 openssl 라이브러리를 이용하면 암호화와 해싱, 그리고 전자서명 생성까지 쉽게 구현 할 수 있었다. 절차는 개인키와 공개키를 생성하고, 메시지를 해싱한다. 이후 해시값과 개인키를 이용하여 서명을 생성한다. 수신자는 해당 서명을 공개키를 이용하여 복호화 하고, 기존 해시값과 비교하여 변조 여부를 확인 할 수 있다.



```

pos > security > signature > python signature.py
(venv) PS S:\repos\security\signature> python signature.py
b'8>f\x3S\x81D\x3\xfa\x3\x7\x7\x0eG\x98Adj\xfb\x10\x81\x0e'
b'85M\x7r\x0Q)0\xaf*3\x07A7` \xf2s\x0d1` \xe0\x0f7" \xa4\x0d1\x03e'
bd{\x00\x0f83\x17\x13\xaf{7\x86\x09\x07\xba\x1e\x0d0~b@\xa6\x0b'
b'8>f\x3S\x81D\x3\xfa\x3\x7\x7\x0eG\x98Adj\xfb\x10\x81\x0e'

```

```

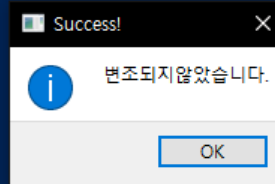
from PyQt5.QtWidgets import
import sys
pkey = crypto.PKey()
pkey.generate_key(crypto.TYP
# 공개키, 개인키 생성
#with open("public.pem", 'ab
# f.write(crypto.dump_pub
#with open("private.pem", 'a
# f.write(crypto.dump_pri
# 공개키, 개인키 읽어오기
with open("private.pem", 'rb
    priv_key = crypto.load_p
with open("public.pem", 'rb+
    pub_key = crypto.load_pu
# 데이터 해싱화
content = '_message' # 가져오
_hash = SHA.new(content.encode

#개인키로 전자서명
_sig = sign(priv_key, _hash,
print(_sig)
x509 = X509() # 인증서를 사용
x509.set_pubkey(pub_key)
# 비교 대상
cc2 = 'dd'
t_hash = SHA.new(cc2.encode(

```

디지털 서명을 한 메시지가 \_message 이고 수신자가 검증하는 해시값이 dd 를 해싱한 경우 변조되었다고 결과 창이 나온다.

```
11 #with open("public.pem", 'ab')
12 # f.write(crypto.dump_public_key(pub_key, f))
13 #with open("private.pem", 'ab')
14 # f.write(crypto.dump_private_key(priv_key, f))
15 # 공개키, 개인키 읽어오기
16 with open("private.pem", 'rb') as f:
17     priv_key = crypto.load_private_key(f.read())
18 with open("public.pem", 'rb') as f:
19     pub_key = crypto.load_public_key(f.read())
20 # 데이터 해시화
21 content = '_message' # 가져오기
22 _hash = SHA.new(content.encode('utf-8')).digest()
23
24
25 #개인키로 전자서명
26 _sig = sign(priv_key, _hash, X509())
27 print(_sig)
28 x509 = X509() # 인증서를 사용
29 x509.set_pubkey(pub_key)
30 # 비교 대상
31 cc2 = '_message'
32 t_hash = SHA.new(cc2.encode('utf-8')).digest()
33 # 사용법: verify(x509, 전자서명, 메시지)
```



해싱된 문자열이 동일하여 verify에 성공했을 경우 변조되지 않았음을 알린다.

서명 행위를 통해 인증을 할 수 있고, 송신자와 수신자가 서명을 확인함으로써 메시지의 무결성을 확인 할 수 있다. 디지털 서명 과정에서 송신자가 개인키를 이용하여 해시값을 암호화 하기 때문에 부인을 방지할 수 있다.

### 3. 소감

과제에 대해 잘 이해가 되지 않았던 부분이 있었다. 해시알고리즘 혹은 암호알고리즘을 이용하여 전자서명 프로그램을 구현해보려 하였는데, 전자서명 과정에 RSA 암호와 해싱 모두가 필요하기 때문에 둘 중 하나만 사용하면서 하는 방법이 있나 싶었다. 기존까지는 내가 직접 구현했던 것들을 사용하려 하였으나, 해싱함수가 제대로 되지 않았기 때문에, 이번에는 파이썬 오픈 라이브러리를 이용하였다. 디지털 서명을 하는 openssl 에서 키 생성, RSA 암호화, 서명 생성, 해싱 까지 다 해주었기 때문에, 상당히 편했다. 이래서 사람들이 파이썬 파이썬 하나 싶었다. 하지만 아직까지는 그래도 c++언어가 가장 마음에 드는 것 같다.