Assignment #1
Cryptography

2018007956 김채아

컴파일 환경 : Python 3.8.3

대칭키 암호화 – AES
hash 함수 – SHA256
비대칭키 암호화 – RSA

위 방법들을 사용하여,
문자열 및 key가 필요한 경우 key (RSA와 같이 key 길이가 긴 경우 key 길이)를 입력 받고,
암호화 및 복호화가 가능한 경우 복호화 하는 과정을 보인다

사용한 API : Crypto / hashlib

```
import hashlib
from Crypto.Cipher import AES
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
```

[대칭키 암호화 – AES]

```python
BS = 16    AES에서는 block size가 128bit 즉 16byte로 고정되는데, 아래 코드를 통해 자동 패딩처리한다
pad = lambda s: s + (BS - len(s.encode('utf-8')) % BS) * chr(BS - len(s.encode('utf-8')) % BS)
unpad = lambda s : s[:-ord(s[len(s)-1:])]

                                                    키 길에 따라
def aes(str):                                       AES128(128bit==16byte)
    print('cipher type : AES')                      AES192(192bit==24byte)
    key = input('key(16/24/32):')  키 길이는 16/24/32만 받는다  AES256(256bit==32byte) 로 나뉜다
    key = key.encode('utf-8')  문자열(유니코드)를 utf-8형식의 byte로 변환
    iv = (chr(0)*16).encode('utf-8') initial vector를 0으로 초기화하여 16바이트 할당
    encrypted = AES.new(key, AES.MODE_CBC, iv)
    str = pad(str)  str의 길이가 block size의 배수가 아닐 때 패딩해준다
    cipherText = encrypted.encrypt(str.encode('utf-8'))
    print('encrypted:', cipherText)

    encrypted = AES.new(key, AES.MODE_CBC, iv)
    plainText = encrypted.decrypt(cipherText)
    plainText = unpad(plainText)
    print('decrypted:', plainText)
    print()
```

input : 메시지, (길이가 16/24/32인) 키
output : 암호화/복호화 된 값

코드 :
부족한 길이만큼 padding 하고 CBC방식으로 키와 초기 벡터를 이용해서 암호화/복호화 한다

AES는 암호화 블록의 크기가 128bit이며 암호화 키의 길이가 128, 192, 256비트인 세 종류가 AES표준이다

[hash 함수 – SHA256]

```
def hash(str):
    print('hash type : SHA256')
    hash = hashlib.sha256(str.encode('utf-8'))
    print(hash.hexdigest())
    print()
```

input : 메시지
output : hash value

SHA256은 256bits로 구성되어 64자리 문자열을 반환한다(경우의 수 : 2^256)
만들어지는 경우의 수가 너무 많기 때문에 무차별 대입에는 매우 안전한 암호화 방식이다

코드 :
인코딩 할 때 문자열을 (컴퓨터가 이해할 수 있는 형태인) 바이트로 변환한다
utf-8은 유니코드를 인코딩하는 방식이다
hashlib 사용하여 해시함수 적용하고 값을 만들어낸다

hexdigest() : return the current digest as a string of hexadecimal digits

[비대칭키 암호화 – RSA]

```
def rsa(str):
    print('RSA')
    k_length = input('key length(x256, >=1024):')
    privateKey = RSA.generate(int(k_length))  개인 키 생성
    publicKey = privateKey.publickey()        공개 키 생성

    encrypted = PKCS1_OAEP.new(publicKey)  공개키로 암호화
    cipherText = encrypted.encrypt(str.encode('utf-8'))
    print('encrypted:', cipherText)
    decrypted = PKCS1_OAEP.new(privateKey) 공개키로 암호화한 메시지를 개인키로 복호화
    plainText = decrypted.decrypt(cipherText)
    print('decrypted:', plainText)
```

input : 메시지, 키 길이
output : 암호화/복호화 된 값

RSA는 두 개의 key로 메시지를 암호화/복호화 한다. 암호화된 메시지는 개인키를 가진 사람만 복호화 할 수 있다
개인키로 암호화하여 공개키로 복호화 할 수도 있는데 이것이 전자서명 방식이다

코드 :
암호화 할 메시지를 받아서 개인키와 공개키를 생성한 후
공개키로 암호화하고, 공개키로 암호화한 메시지를 개인키로 복호화 한다

[실행 화면]

```
original data:Hello World!

cipher type : AES
key(16/24/32):1234123412341234
encrypted: b'\x1b\x16\xa3\xeb\x90\xfd\xdc-~\xcb/\x8c\x80\xf2\t\xc4'
decrypted: b'Hello World!'

hash type : SHA256
7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069

RSA
key length(x256, >=1024):1024
encrypted: b'9\xcc\xdd\xa0\x01\xba\xa0\xd1\xd6\xab\xe9d,\x9e~\xcb\xc7\x07|\x89
decrypted: b'Hello World!'
```