RSA 암호, 전자서명

RSA 키 생성

$ pip3 install pycryptodome

- 키 생성

```
>>> from Crypto.PublicKey import RSA
>>> keyPair = RSA.generate (2048)
>>> print ( keyPair.n )
>>> print ( keyPair.e  )
>>> print ( hex ( keyPair.n ) )
>>> print ( len ( hex (keyPair.n ) )  # 키 길이 확인
```

RSA 암복호화

```
>>> from Crypto.Cipher import PKCS1_OAEP
```

- 암호화

```
>>> pubkey = keyPair.publickey( )   # public key 분리
>>> encryptor = PKCS1_OAEP.new ( pubkey ) # 암호화 객체 생성
>>> encrypted=encryptor.encrypt( b"hello") # 암호화
>>> import binascii
>>> print ("Encrypted: ", binascii.hexlify(encrypted) )   # 암호문 출력
```

- 복호화 : 비밀키 필요

```
>>> decryptor = PKCS1_OAEP.new( keyPair )  # 개인키 소유자는 pub key도 갖고 있음
>>> decrypted=decryptor.decrypt(encrypted)
>>> print ("Decrypted: ", decrypted)
```

키쌍 저장

개인키

- 비밀번호로 암호화하여 저장

```
>>> priKeyPEM= keyPair.export_key (passphrase="1234")
```

- PEM encoding 된 값 : 다음 페이지 PEM encoding 참조

```
>>> print (priKeyPEM)
```

- 파일에 저장할 수 있음
- 저장된 비밀키 사용

```
>>> keyPair= RSA.importKey(priKeyPEM, passphrase="1234" )
```

공개키

- 암호화 없음

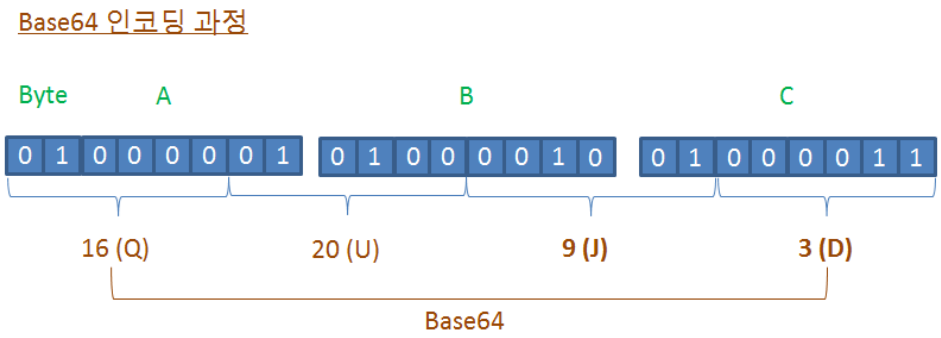
```
>>> pubKeyPEM=pubkey.export_key()
>>> pubKey= RSA.importKey(pubKeyPEM)
```

- 저장, 전송, 자유롭게

PEM 인코딩

- Encoding
  - 정보를 다른 형태나 형식으로 변환하는 처리, 처리 방식
- Base64 인코딩
  - Binary data -> text로 바꿈 (메일 본문에 보내거나, url 파라메터에 많이 사용)
  - Text중에서 ascii 영역에서 display되는 문자 64개만 사용해서 표현
  - 6비트 (2^6= 64) 씩 끊어서 한 문자씩 표현
  - 맨 뒤에 패딩 : = , 3byte=> 4글자 ,

3byte로 끊어지지 않아도 4글자 단위로 끊기 위해 패딩사용

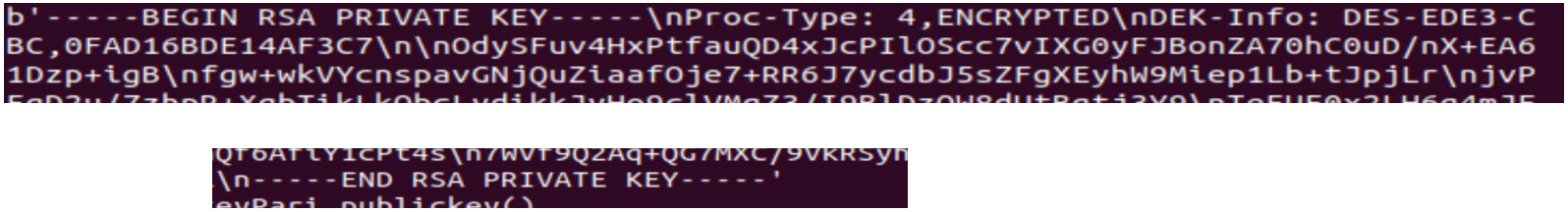


Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

PEM 인코딩

```
>>> import base64
>>> base64.b64encode( bytes ( [0b00000000] ) )
>>> base64.b64encode( bytes ( [0b00000000, 0x00, 0x00] ) )
```

- PEM encoding
  - Privacy Enhancing Mail 표준 인코딩
  - Base64
  - 앞 뒤에 설명 및 정보 포함
  - Txt 이므로 메일로 보내는 등 text로 처리할 수 있음



과제1

- 1a-1.py
  - 키쌍 생성
  - 개인키를 Alice_private.pem으로 저장
  - 공개키를 Alice_public.pem으로 저장

- 1b.py
  - Alice_public.pem을 읽어 들여서
  - 임의 파일 1.txt를 이 공개키로 암호화하여 enc.txt로 저장
    - 1.txt 는 너무 길 필요 없음
    - Enc.txt 파일 구조는 알아서..
- 1a-2.py
  - Passphrase를 사용자에게 입력받은 후
  - Alice_private.pem으로 부터 private key를 읽고
  - Enc.txt를 읽어서 복호화 후 화면에 print

- 다음 순서대로 실행 후 화면 캡처 1.jpg

```
$ python3 1a-1.py
$ cat Alice_private.pem
$ python3 1b.py
$ python3 1a-2.py
```

RSA 전자서명

- 키 쌍 생성

```
>>> from Crypto.PublicKey import RSA
>>> key=RSA.generate(2048)
```

- 서명

```
>>> from Crypto.Signature import PKCS1_PSS
>>> from Crypto.Hash import SHA
>>> msg = b"To be signed"
>>> h= SHA.new( )
>>> h.update ( msg )
>>> signer=PKCS1_PSS.new ( key )
>>> sig=signer.sign ( h )  # hash값을 넣어줘야 함
>>> print (sig)
```

RSA 전자서명

- 검증

```
>>> h = SHA.new( )
>>> h.update ( msg )
>>> pubKey=key.publickey( ) # 공개키만 있으면 검증 가능
>>> verifier=PKCS1_PSS.new ( pubKey )
>>> verifier.verify ( h, sig )
```

ASN.1 구조

- **Abstract Syntax Notation One**
  - Interface description language
  - Serialize/Deserialize
  - 통신이나, 암호 분야에서 많이 사용 : X.509 인증서, public key 저장..
  - 요새 json 과 같은 개념

```
FooProtocol DEFINITIONS ::= BEGIN

    FooQuestion ::= SEQUENCE {
        trackingNumber  INTEGER,
        question        IA5String
    }

    FooAnswer ::= SEQUENCE {
        questionNumber  INTEGER,
        answer          BOOLEAN
    }

END
```

<구조 정의>

```
myQuestion FooQuestion ::= {
    trackingNumber      5,
    question            "Anybody there?"
}
```

<객체 생성>

DER 인코딩

- encoding 하는 방식 중 하나
- ASN.1 객체를 encoding할 때 주로 사용
  - Type, length, value
  - myQuestion 객체를 encoding

```
30 13 02 01 05 16 0e 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f
```

```
30 - type tag indicating SEQUENCE
13 - length in octets of value that follows
  02 - type tag indicating INTEGER
  01 - length in octets of value that follows
    05 - value (5)
  16 - type tag indicating IA5String
     (IA5 means the full 7-bit ISO 646 set, including variants,
      but is generally US-ASCII)
  0e - length in octets of value that follows
    41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f - value ("Anybody there?")
```

ASN1 사용 예

```
>>> from Crypto.Util.asn1 import *
>>> from binascii import hexlify
>>> seq = DerSequence ( )
>>> seq.append(9)
>>> seq.append(5)
>>> enc=seq.encode()          #der encoding
>>> hexlify (enc)
```

- 실제 운영은
  - 컴파일러 : asn.1 definitions을 언어의 객체로 변경
  - 혹은 미리 만들어진 structure에 값을 넣고
  - Encoding 함

RSA KEY DER ENCODING

- RSA Key도 DER encoding으로 저장하는 것이 일반적
  - X.509 인증서 등도 …
  - 추후 PKI 실습시 공인인증서 개인키 저장 방식

```
>>> pubKeyDER= pubKey.export_key( format="DER" )
>>> hexlify (pubKeyDER)
```

https://pycryptodome.readthedocs.io

export_key(*format='PEM', passphrase=None, pkcs=1, protection=None, randfunc=None*)

Export this RSA key.

Parameters:

- **format** (*string*) –
  The format to use for wrapping the key:
  - *'PEM'*. (*Default*) Text encoding, done according to RFC1421/RFC1423.
  - *'DER'*. Binary encoding.
  - *'OpenSSH'*. Textual encoding, done according to OpenSSH specification. Only suitable for public keys (not private keys).
- **passphrase** (*string*) – (*For private keys only*) The pass phrase used for protecting the output.
- **pkcs** (*integer*) –
  (*For private keys only*) The ASN.1 structure to use for serializing the key. Note that even in case of PEM encoding, there is an inner ASN.1 DER structure.

  With `pkcs=1` (*default*), the private key is encoded in a simple PKCS#1 structure (`RSAPrivateKey`).

  With `pkcs=8`, the private key is encoded in a PKCS#8 structure (`PrivateKeyInfo`).

Crypto.PublicKey.RSA.import_key(*extern_key, passphrase=None*)

Import an RSA key (public or private).

Parameters:

- **extern_key** (*string or byte string*) –
  The RSA key to import.

  The following formats are supported for an RSA **public key**:
  - X.509 certificate (binary or PEM format)
  - X.509 `subjectPublicKeyInfo` DER SEQUENCE (binary or PEM encoding)
  - PKCS#1 `RSAPublicKey` DER SEQUENCE (binary or PEM encoding)
  - An OpenSSH line (e.g. the content of `~/.ssh/id_ecdsa`, ASCII)

  The following formats are supported for an RSA **private key**:
  - PKCS#1 `RSAPrivateKey` DER SEQUENCE (binary or PEM encoding)
  - PKCS#8 `PrivateKeyInfo` or `EncryptedPrivateKeyInfo` DER SEQUENCE (binary or PEM encoding)
  - OpenSSH (text format, introduced in OpenSSH 6.5)

  For details about the PEM encoding, see RFC1421/RFC1423.
- **passphrase** (*string or byte string*) – For private keys only, the pass phrase that encrypts the key.

과제2

- 2a.py
  - 키쌍 생성
  - 공개키를 public.der로 저장 (der 인코딩으로 저장)
  - 1.txt 를 읽어서 서명 후 서명 값은 Base64 encoding하여 sig.txt에 저장
  - ☞ 문서와 sign 을 저장하기 위해 표준화된 asn.1 포맷 (PKCS#7) 이 따로 있음

- 2b.py
  - Public.der와 sig.txt, 1.txt을 읽어서
  - 1.txt에 대한 서명을 검증하는 코드를 작성 , 맞으면 "verified" 틀리면 "not verified"를 출력
    - 1.txt의 한글자만 바꿔도 not verified가 되어야 함
- 실행화면 캡처 : 2.jpg

```
$ python3 2a.py
$ python3 2b.py
```