교과목 : 정보보호

PyCryptodome
Introduction 1

2023학년도 2학기
Suk-Hwan Lee

- PyCryptodome – Cryptographic library for Python

참조자료

- PyCryptodome Homepage : https://www.pycryptodome.org/en/latest/
- Download : https://pypi.org/project/pycryptodome/
- GitHub : https://github.com/Legrandin/pycryptodome

코드 참조

- https://www.pycryptodome.org/en/latest/src/examples.html
- 화이트 해커를 위한 암호와 해킹 (2판), White Hat Python, 장삼용 저, 정보문화사, 2019년9월

Pycryptodome

- *PyCryptodome* is a self-contained Python package of low-level cryptographic primitives.
- *PyCryptodome* is a fork of *PyCrypto*. It brings the following enhancements with respect to the last official version of PyCrypto (2.6.1):

- PyCryptodome is not a wrapper to a separate C library like OpenSSL.
- To the largest possible extent, algorithms are implemented in pure Python. Only the pieces that are extremely critical to performance (e.g. block ciphers) are implemented as C extensions.

- Installation procedure depends on the package

1. **an almost drop-in replacement for the old PyCrypto library**. You install it with:

```
pip install pycryptodome
```

In this case, all modules are installed under the `Crypto` package.

One must avoid having both PyCrypto and PyCryptodome installed at the same time, as they will interfere with each other.

This option is therefore recommended only when you are sure that the whole application is deployed in a `virtualenv`.

2. **a library independent of the old PyCrypto**. You install it with:

```
pip install pycryptodomex
```

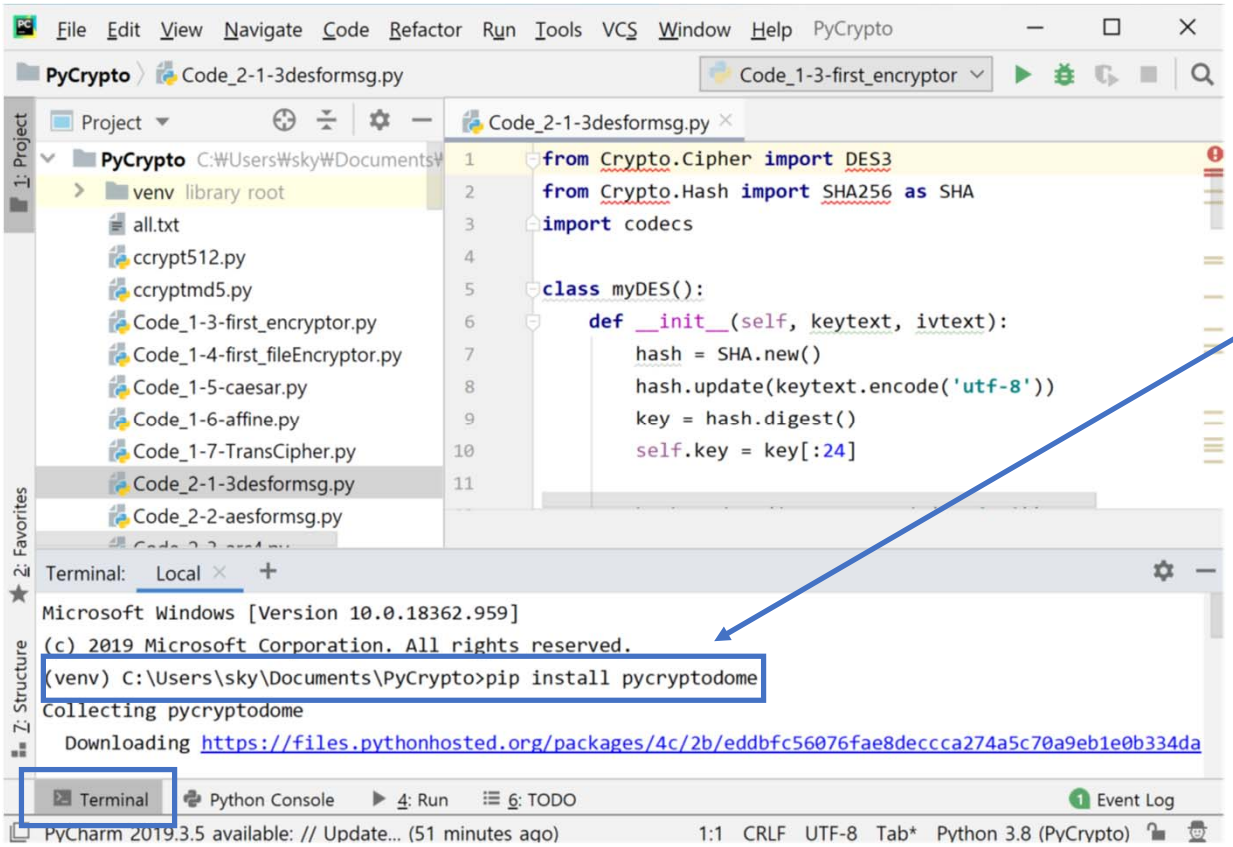
In this case, all modules are installed under the `Cryptodome` package. PyCrypto and PyCryptodome can coexist.

Python 실행하고, Pycryptodome 모듈 import하여 설치 확인

```
>> import pycryptodome
```

Pycryptodome

- 본 강의는 Pycharm 로 실행함 (편한거 사용하여도 좋음…..)



Settings

- Project : PyCrypto
- Project interpreter : Python3.8

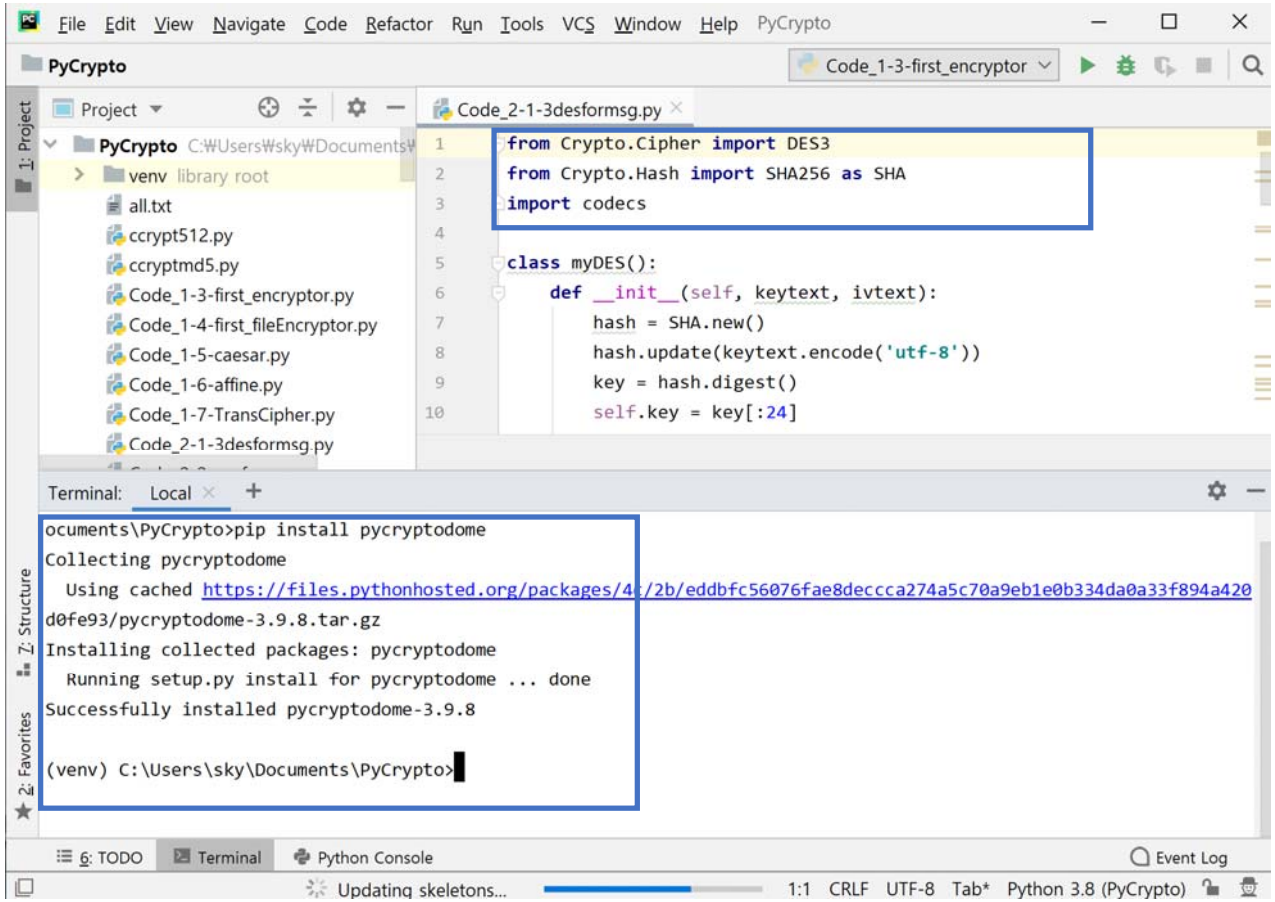
- Terminal : pip install pycryptogdome

❖ distutils.errors.DistutilsPlatformError: Microsoft Visual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools “

❖ Visual Studio Community 2019 설치하고 pycryptodome 다시 설치

Pycryptodome

- 본 강의는 Pycharm 로 실행함 (편한거 사용하여도 좋음…..)



Pycryptodome Examples

Encrypt data with AES

- Generates a new AES128 key (16byte) and encrypts a piece of data into a file.
- Use the EAX mode because it allows the receiver to detect any unauthorized modification (similarly, we could have used other authenticated encryption modes like GCM, CCM or SIV).

[Wikipedia] EAX mode

- EAX mode (encrypt-then-authenticate-then-translate) is a mode of operation for cryptographic block ciphers.
- It is an Authenticated Encryption with Associated Data (AEAD) algorithm designed to simultaneously provide both authentication and privacy of the message (authenticated encryption) with a two-pass scheme, one pass for achieving privacy and one for authenticity for each block.

PyCryptoExample_Encrypt-data-with-AES.py

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Encryption #

Origin_data = "I'm a Student in Dong-A University.".encode("utf-8")

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(Origin_data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
file_out.close()

# Decryption #

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in (16, 16, -1) ]

# let's assume that the key is somehow available again
cipher = AES.new(key, AES.MODE_EAX, nonce)
Decrypted_data = cipher.decrypt_and_verify(ciphertext, tag)

print(Decrypted_data)
```

Pycryptodome Examples

Crypto.Cipher.AES.new(*key, mode, *args, **kwargs*)

Create a new AES cipher.

Parameters:

- **key** (*bytes/bytearray/memoryview*) –
  The secret key to use in the symmetric cipher.

  It must be 16, 24 or 32 bytes long (respectively for *AES-128*, *AES-192* or *AES-256*).

  For `MODE_SIV` only, it doubles to 32, 48, or 64 bytes.
- **mode** (One of the supported `MODE_*` constants) – The chaining mode to use for encryption or decryption. If in doubt, use `MODE_EAX`.

Keyword Arguments:

Return: an AES object, of the applicable mode.

https://www.pycryptodome.org/en/latest/src/cipher/aes.html?highlight=AES.new

encrypt_and_digest(*plaintext, output=None*)

Perform `encrypt()` and `digest()` in one go.

Parameters: **plaintext** (*bytes*) – the last piece of plaintext to encrypt

Keyword Arguments:

output (*bytes/bytearray/memoryview*) – the pre-allocated buffer where the ciphertext must be stored (as opposed to being returned).

Returns: a tuple with two items

- the ciphertext, as `bytes`
- the MAC tag, as `bytes`

The first item becomes `None` when the `output` parameter specified a location for the result.

PyCryptoExample_Encrypt-data-with-AES.py

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Encryption #

Origin_data = "I'm a Student in Dong-A University.".encode("utf-8")

key = get_random_bytes(16)          #16byte
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(Origin_data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
file_out.close()

# Decryption #

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in (16, 16, -1) ]

# let's assume that the key is somehow available again
cipher = AES.new(key, AES.MODE_EAX, nonce)
Decrypted_data = cipher.decrypt_and_verify(ciphertext, tag)

print(Decrypted_data)
```

Pycryptodome Examples

EAX mode

An AEAD mode designed for NIST by Bellare, Rogaway, and Wagner in 2003.

The `new()` function at the module level under `Crypto.Cipher` instantiates a new EAX cipher object for the relevant base algorithm.

`Crypto.Cipher.<algorithm>.new(key, mode, *, nonce=None, mac_len=None)`

Create a new EAX object, using <algorithm> as the base block cipher.

Parameters:

- **key** (*bytes*) – the cryptographic key
- **mode** – the constant `Crypto.Cipher.<algorithm>.MODE_EAX`
- **nonce** (*bytes*) – the value of the fixed nonce. It must be unique for the combination message/key. If not present, the library creates a random nonce (16 bytes long for AES).
- **mac_len** (*integer*) – the desired length of the MAC tag (default if not present: the cipher's block size, 16 bytes for AES).

Returns: an EAX cipher object

https://www.pycryptodome.org/en/latest/src/cipher/modern.html#eax-mode

Example (encryption):

```
>>> import json
>>> from base64 import b64encode
>>> from Crypto.Cipher import AES
>>> from Crypto.Random import get_random_bytes
>>>
>>> header = b"header"
>>> data = b"secret"
>>> key = get_random_bytes(16)
>>> cipher = AES.new(key, AES.MODE_EAX)
>>> cipher.update(header)
>>> ciphertext, tag = cipher.encrypt_and_digest(data)
>>>
>>> json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
>>> json_v = [ b64encode(x).decode('utf-8') for x in cipher.nonce, header, ciphertext, tag ]
>>> result = json.dumps(dict(zip(json_k, json_v)))
>>> print(result)
{"nonce": "CSIJ+e8KP7HJo+hC4RXIyQ==", "header": "aGVhZGVy", "ciphertext": "9YYjuAn6", "tag": "kXHrs9ZwYr
```

Example (decryption):

```
>>> import json
>>> from base64 import b64decode
>>> from Crypto.Cipher import AES
>>>
>>> # We assume that the key was securely shared beforehand
>>> try:
>>>     b64 = json.loads(json_input)
>>>     json_k = [ 'nonce', 'header', 'ciphertext', 'tag' ]
>>>     jv = {k:b64decode(b64[k]) for k in json_k}
>>>
>>>     cipher = AES.new(key, AES.MODE_EAX, nonce=jv['nonce'])
>>>     cipher.update(jv['header'])
>>>     plaintext = cipher.decrypt_and_verify(jv['ciphertext'], jv['tag'])
>>>     print("The message was: " + plaintext)
>>> except ValueError, KeyError:
>>>     print("Incorrect decryption")
```

Generate an RSA key

- Generates a new RSA key pair (secret) and saves it into a file, protected by a password.
- Use the *scrypt key derivation function* to thwart dictionary attacks.
- At the end, the code prints our the RSA public key in ASCII/PEM format:

- Reads the private RSA key back in, and then prints again the public key

❖ PEM (Privacy Enhanced Mail)은 Base64인코딩된 ASCII text file임

PyCryptoExample_Generate-RSA-Key.py

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
key = RSA.generate(2048)            #2048bit
encrypted_key = key.export_key(passphrase=secret_code, pkcs=8,
                               protection="scryptAndAES128-CBC")

file_out = open("rsa_key.bin", "wb")
file_out.write(encrypted_key)
file_out.close()

print(key.publickey().export_key())

# #
encoded_key = open("rsa_key.bin", "rb").read()
key = RSA.import_key(encoded_key, passphrase=secret_code)

print(key.publickey().export_key())
```

```
"C:\Users\computer office\Documents\PyCrypto\venv\Scripts\python.exe" "C:/Users/computer office/Documents/PyCrypto/02_Generate-an-RSA-key.py"
b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAplVlguD9dhPNCXuLLigW\ncKZS2QOQykRfeYBdmjHM8P3+FQ1O/Lsn1W/10+UzXDaTtBC+Fg9JITQr4qIT3z9+\n9Qr+8wyuY1ONjtkysQF
b'-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAplVlguD9dhPNCXuLLigW\ncKZS2QOQykRfeYBdmjHM8P3+FQ1O/Lsn1W/10+UzXDaTtBC+Fg9JITQr4qIT3z9+\n9Qr+8wyuY1ONjtkysQF
```

Pycryptodome Examples

Crypto.PublicKey.RSA.generate(*bits, randfunc=None, e=65537*)

Create a new RSA key pair.

The algorithm closely follows NIST FIPS 186-4 in its sections B.3.1 and B.3.3. The modulus is the product of two non-strong probable primes. Each prime passes a suitable number of Miller-Rabin tests with random bases and a single Lucas test.

Parameters:

- **bits** (*integer*) – Key length, or size (in bits) of the RSA modulus. It must be at least 1024, but **2048 is recommended.** The FIPS standard only defines 1024, 2048 and 3072.
- **randfunc** (*callable*) – Function that returns random bytes. The default is `Crypto.Random.get_random_bytes()`.
- **e** (*integer*) – Public RSA exponent. It must be an odd positive integer. It is typically a small number with very few ones in its binary representation. The FIPS standard requires the public exponent to be at least 65537 (the default).

Returns: an RSA key object (`RsaKey` , with private key).

PyCryptoExample_Generate-RSA-Key.py

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
key = RSA.generate(2048)          #2048bit
encrypted_key = key.export_key(passphrase=secret_code, pkcs=8,
                               protection="scryptAndAES128-CBC")

file_out = open("rsa_key.bin", "wb")
file_out.write(encrypted_key)
file_out.close()

print(key.publickey().export_key())

# #
encoded_key = open("rsa_key.bin", "rb").read()
key = RSA.import_key(encoded_key, passphrase=secret_code)

print(key.publickey().export_key())
```

Pycryptodome Examples

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`).
- **protection** (*string*) –
  (*For private keys only*) The encryption scheme to use for protecting the private key.

  If `None` (default), the behavior depends on `format` :

Returns: the encoded key

Return type: byte string

Raises: `ValueError` – when the format is unknown or when you try to encrypt a private key with *DER* format and PKCS#1.

PyCryptoExample_Generate-RSA-Key.py

```
from Crypto.PublicKey import RSA

secret_code = "Unguessable"
key = RSA.generate(2048)          #2048bit
encrypted_key = key.export_key(passphrase=secret_code, pkcs=8,
                               protection="scryptAndAES128-CBC")

file_out = open("rsa_key.bin", "wb")
file_out.write(encrypted_key)
file_out.close()

print(key.publickey().export_key())

# #
encoded_key = open("rsa_key.bin", "rb").read()
key = RSA.import_key(encoded_key, passphrase=secret_code)

print(key.publickey().export_key())
```

Generate public key and private key

- Generates public key stored in receiver.pem and private key stored in private.pem.
- These files will be used in the next examples.
- Every time, it generates different public key and private key pair.

PyCryptoExample_Generate-PublicKey-PrivateKey.py

```
from Crypto.PublicKey import RSA

key = RSA.generate(2048)
private_key = key.export_key()
file_out = open("private.pem", "wb")
file_out.write(private_key)
file_out.close()

public_key = key.publickey().export_key()
file_out = open("receiver.pem", "wb")
file_out.write(public_key)
file_out.close()
```

Pycryptodome Examples

Encrypt data with RSA

- Encrypts a piece of data for a receiver we have the RSA public key of.
- The RSA public key is stored in a file called receiver.pem.
- Since we want to be able to encrypt an arbitrary amount of data, we use a hybrid encryption scheme. We use RSA with PKCS#1 OAEP for asymmetric encryption of an AES session key. The session key can then be used to encrypt all the actual data.
- We use the EAX mode to allow detection of unauthorized modifications.

PyCryptoExample_Encrypt-Data-RSA.py

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP

data = "I met aliens in UFO. Here is the map.".encode("utf-8")
file_out = open("encrypted_data.bin", "wb")

recipient_key = RSA.import_key(open("receiver.pem").read())
session_key = get_random_bytes(16)

# Encrypt the session key with the public RSA key
cipher_rsa = PKCS1_OAEP.new(recipient_key)
enc_session_key = cipher_rsa.encrypt(session_key)

# Encrypt the data with the AES session key
cipher_aes = AES.new(session_key, AES.MODE_EAX)
ciphertext, tag = cipher_aes.encrypt_and_digest(data)
[ file_out.write(x) for x in (enc_session_key, cipher_aes.nonce, tag, ciphertext) ]
file_out.close()
```

Pycryptodome Examples

Decrypt the encrypted data with RSA

- The receiver has the private RSA key.
- They will use it to decrypt the session key first, and with that the rest of the file:

PyCryptoExample_Decrypt-EncryptedData-RSA.py

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

file_in = open("encrypted_data.bin", "rb")

private_key = RSA.import_key(open("private.pem").read())

enc_session_key, nonce, tag, ciphertext = \
   [ file_in.read(x) for x in (private_key.size_in_bytes(), 16, 16, -1) ]

# Decrypt the session key with the private RSA key
cipher_rsa = PKCS1_OAEP.new(private_key)
session_key = cipher_rsa.decrypt(enc_session_key)

# Decrypt the data with the AES session key
cipher_aes = AES.new(session_key, AES.MODE_EAX, nonce)
data = cipher_aes.decrypt_and_verify(ciphertext, tag)
print(data.decode("utf-8"))
```

```
"C:\Users\computer office\Documents\PyCrypto\venv\Scripts\python.exe" "C:/l
I met aliens in UFO. Here is the map.

Process finished with exit code 0
```

Pycryptodome Examples

Encrypt data with AES

- Generates a new AES128 key (16byte) and encrypts a piece of data into a file.
- Use the EAX mode because it allows the receiver to detect any unauthorized modification (similarly, we could have used other authenticated encryption modes like GCM, CCM or SIV).

[Wikipedia] EAX mode

- EAX mode (encrypt-then-authenticate-then-translate) is a mode of operation for cryptographic block ciphers.
- It is an Authenticated Encryption with Associated Data (AEAD) algorithm designed to simultaneously provide both authentication and privacy of the message (authenticated encryption) with a two-pass scheme, one pass for achieving privacy and one for authenticity for each block.

PyCryptoExample_Encrypt-data-with-AES.py

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

# Encryption #

Origin_data = "I'm a Student in Dong-A University.".encode("utf-8")

key = get_random_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(Origin_data)

file_out = open("encrypted.bin", "wb")
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
file_out.close()

# Decryption #

file_in = open("encrypted.bin", "rb")
nonce, tag, ciphertext = [ file_in.read(x) for x in (16, 16, -1) ]

# let's assume that the key is somehow available again
cipher = AES.new(key, AES.MODE_EAX, nonce)
Decrypted_data = cipher.decrypt_and_verify(ciphertext, tag)

print(Decrypted_data)
```


여기에 슬라이드 노트의 내용을 입력하십시오

교과목 : 정보보호

PyCryptodome
2 단문암호 3DES, AES

2023학년도 2학기
Suk-Hwan Lee

- PyCryptodome – Cryptographic library for Python

참조자료

- PyCryptodome Homepage : https://www.pycryptodome.org/en/latest/
- Download : https://pypi.org/project/pycryptodome/
- GitHub : https://github.com/Legrandin/pycryptodome

코드 참조

- https://www.pycryptodome.org/en/latest/src/examples.html
- 화이트 해커를 위한 암호와 해킹 (2판), White Hat Python, 장삼용 저, 정보문화사, 2019년9월

참고 : 유니코드

- Python3에서 문자열은 모두 유니코드로 처리
  - ✓ 예) 문자열 'I Love Python', "I Love Python" – 모두 유니코드로 처리
- 유니코드로 Encoding 되지 않은 문자열은 바이트 객체로 인식
  - ✓ 바이트 객체값이 112이면 그냥 112로 인식
  - ✓ ASCII 코드로 처리하는 경우 112는 소문자 'p'로 인식
  - ✓ 바이트 객체는 소문자 b를 사용하여 나타냄

```
>>> msg = b'python3x'     # 바이트 객체로 'python3x' 선언
```

- 바이트 객체는 이진 데이터로 인식하므로 유니코드 문자열과 비교하는 경우 올바른 결과가 나오지 않음

```
>>> msg = b'python3x'
>>> msg[0] == 'p'
False
>>> msg[0]
112
```

  - ✓ msg를 바이트 객체 'python3x'로 선언
  - ✓ msg[0]의 112는 ASCII 코드 소문자 'p'에 해당

참고 : 유니코드

- 바이트 객체로 선언된 'python3x' 를 한 문자씩 출력하면 숫자로 출력

```
>>> msg = b'python3x'
>>> for c in msg:
        print(c, end=" ")
112  121  116  104  111  110  51  120
```

- chr() 함수 사용하여 바이트 객체인 문자 하나를 유니코드 문자로 변경

```
>>> msg = b'python3x'
>>> for c in msg:
        print(chr(c), end=" ")
p y t h o n 3 x
```

- 바이트 객체로 선언된 문자열은 decode() 사용하여 유니코드 문자로 변경

```
>>> msg = msg.decode()
>>> msg[0] == 'p'
True
>>> msg[0]
'p'
```

- Pycryptodome은 유니코드 문자열을 지원하지 않음.
- 유니코드 문자열을 encode('utf-8') 이용하여 UTF-8로 인코딩함. 결과값은 decode() 이용하여 유니코드 문자열로 변경하며 됨

```
>>> msg = 'python3x'       # 유니코드 문자열로 선언
>>> msg.encode('utf-8')
b'python3x'
```

참고 : string.split()

- 문자열 객체 메서드 split() : 입력된 문자 또는 문자열을 구분자로 하여 분리하여 순서대로 리스트에 담아 리턴

```
>>> msg = 'I love Python'
>>> msg.split()
['I', 'love', 'Python']
>>> msg = '12##34##56##78'
>>> msg.split('##')
['12', '34', '56', '78']
```

- split() 은 문자열 또는 정형 데이터를 parsing할 때 유용
- join() : split()와 반대로 문자가 멤버인 리스트를 인자로 받아, 리스트의 각 멤버를 특정 문자 또는 문자열로 결합

```
>>> bond = '##'
>>> loglist = ['12', '34', '56', '78']
>>> bond.join(loglist)
'12##34##56##78'
```

단문암호 3DES Python Ex.

3DES 구현

'python3x'의 8문자로 된 문자열을 3DES CBC 모드로 암호화하고, 3DES로 복호화

```
from Crypto.Cipher import DES3
from Crypto.Hash import SHA256 as SHA
import codecs

class myDES():
        def __init__(self, keytext, ivtext):
                hash = SHA.new()
                hash.update(keytext.encode('utf-8'))
                key = hash.digest()
                self.key = key[:24]

                hash.update(ivtext.encode('utf-8'))
                iv = hash.digest()
                self.iv = iv[:8]


        def enc(self, plaintext):
                #plaintext = make8String(plaintext)
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                encmsg = des3.encrypt(plaintext.encode())
                return encmsg


        def dec(self, ciphertext):
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                decmsg = des3.decrypt(ciphertext)
                return decmsg
```

```
def make8String(msg):
        msglen = len(msg)
        filler = ''
        if msglen%8 != 0:
                filler = '0'*(8 - msglen%8)

        msg += filler

        return msg


def main():
        keytext = 'samsjang'
        iv = '1234'
        msg = 'python3x'
        #msg = 'python35ab'

        myCipher = myDES(keytext, iv)
        ciphered = myCipher.enc(msg)
        deciphered = myCipher.dec(ciphered)
        print('ORIGINAL:₩t%s' %msg)
        print('CIPHERED:₩t%s' %codecs.encode(ciphered, 'hex_codec'))
        print('DECIPHERED:₩t%s' %deciphered)

if __name__ == '__main__':
        main()
```

3DES 코드 설명

```
from Crypto.Cipher import DES3
from Crypto.Hash import SHA256 as SHA
import codecs
```

- Pycryptodome의 3DES 모듈인 Crypto.Cipher.DES3 Imoort함
- Crypto.Hash.SHA256 모듈은 3DES 암호키와 초기화 벡터 생성에 활용
- codecs 모듈 : 유니코드 문자열 다루기 위함

```
class myDES():
        def __init__(self, keytext, ivtext):
                hash = SHA.new()
                hash.update(keytext.encode('utf-8'))
                key = hash.digest()
                self.key = key[:24]

                hash.update(ivtext.encode('utf-8'))
                iv = hash.digest()
                self.iv = iv[:8]
```

- myDES 클래스 정의 : 암호화 enc()와 복호화 dec() 메서드 정의
- 클래스 생성자 __init__( )
  - ✓ 3DES 객체 생성시 사용할 키와 초기화 벡터 구함
    - ① keytext : 3DES 암호키 생성을 위한 문자열
    - ② ivtext : 초기화 벡터 위한 문자열
  - ✓ keytext='abcdefghijklnop'와 같이 16바이트 길이가 되어 3DES가 지원하는 키 길이와 같다면 바로 암호키로 활용함. (16자 이상이면 외우기 힘들고, 관리가 힘듬
  - ✓ SHA256 해시 → keytext 길이가 무엇이든 간에 3DES가 지원하는 길이로 키를 활용하면 편리함

❖ 뒷면 계속 설명

3DES 코드 설명

```
class myDES():
        def __init__(self, keytext, ivtext):
                hash = SHA.new()
                hash.update(keytext.encode('utf-8'))
                key = hash.digest()
                self.key = key[:24]

                hash.update(ivtext.encode('utf-8'))
                iv = hash.digest()
                self.iv = iv[:8]
```

- SHA256.new() : SHA256 객체 만들고 hash에 할당
- hash.update() : keytext 인자값으로 SHA256 해시 갱신
  - ***주의*** : *SHA256.update()는 유니코드 문자열을 인자로 받지 않음. Python3에서 모든 문자열은 유니코드이므로, UTF-8로 인코딩한 문자열을 입력함*
- hash.digest() : 해시 값을 추출하여 변수 key에 할당 (SHA256은 256비트 해시 생성하므로, key는 256비트 (32바이트임)
- Pycryptodome에서 제공하는 3DES 키 크기는 16바이트 또는 24바이트 크기임. 따라서 변수 key를 16바이트 또는 24바이트만큼 슬라이싱하여 3DES 키로 사용 (self.key에 할당)
- 블록 암호 CBC 모드로 암호하기 위해 초기화 벡터가 필요
- 3DES는 64비트 암호화 블록 크기를 가지므로, 64비트 (8바이트) 초기화 벡터 필요
- hash.update(ivtext)로 초기화 벡터를 위한 해시를 갱신
- hash.digest()로 해시를 얻은 후 변수 iv에 할당
- iv의 8바이트를 슬라이싱하여 self.iv 초기화 벡터값으로 할당

➢ SHA256.update() 인자로 유니코드 입력시 다음과 같은 오류 발생

```
>>> keytext = 'secretkey'
>>> hash.update(keytext)
TypeError: Object type <class 'str'> cannot be passed to C code
```

Pycryptodome – 단문암호 3DES

3DES 구현

```
class myDES():
        def enc(self, plaintext):
                #plaintext = make8String(plaintext)
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                encmsg = des3.encrypt(plaintext.encode())
                return encmsg



        def dec(self, ciphertext):
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                decmsg = des3.decrypt(ciphertext)
                return decmsg
```

- dec() : ciphertext를 3DES로 복호화
- DES3.new()로 3DES 객체 des3 생성
- des3.decrypt() : 암호문을 복호하하고 결과 리턴 (decmsg)

- enc() : plaintext를 3DES로 암호화
- DES3.new()로 3DES 객체 des3 생성 : 인자는 암호키(self.key), 운영 모드 (DES3.MODE_CBC), 초기화 벡터(self.iv)
- ECB, CTR 모드는 초기화 벡터가 필요 없음
- des3.encrypt() : 암호화 수행하고 결과 리턴 (encmsg)

Crypto.Cipher.DES3.new(*key*, *mode*, **args*, ***kwargs*)

Create a new Triple DES cipher.

Parameters:

- **key** (*bytes/bytearray/memoryview*) – The secret key to use in the symmetric cipher. It must be 16 or 24 byte long. The parity bits will be ignored.
- **mode** (One of the supported `MODE_*` constants) – The chaining mode to use for encryption or decryption.

Keyword Arguments:

- **iv** (*bytes*, *bytearray*, *memoryview*) – (Only applicable for `MODE_CBC`, `MODE_CFB`, `MODE_OFB`, and `MODE_OPENPGP` modes).

  The initialization vector to use for encryption or decryption.

  For `MODE_CBC`, `MODE_CFB`, and `MODE_OFB` it must be 8 bytes long.

  For `MODE_OPENPGP` mode only, it must be 8 bytes long for encryption and 10 bytes for decryption (in the latter case, it is actually the *encrypted* IV which was prefixed to the ciphertext).

  If not provided, a random byte string is generated (you must then read its value with the `iv` attribute).

3DES 구현

```
def main():
    keytext = 'samsjang'
    ivtext = '1234'
    msg = 'python3x'

    myCipher = myDES(keytext, ivtext)
    ciphered = myCipher.enc(msg)
    deciphered = myCipher.dec(ciphered)
    print('ORIGINAL:\t%s' %msg)
    print('CIPHERED:\t%s' %ciphered)
    print('DECIPHERED:\t%s' %deciphered)
```

- 결과

```
ORIGINAL:    python3x
CIPHERED:    b'S\x9d\xf2\xfe#\xf7\xfa\x06'
DECIPHERED:  b'python3x'
```

- 암호키 keytext " 'samsjang', 초기화 벡터 ivtext '1234'
- 코드 내부에서 운영되는 암호키와 초기화 벡터는 SHA256 해시값 중 24 바이트와 8바이트가 됨

- 결과에서 b'는 문자열이 아닌 바이트 객체를 나타냄
- 결과 ciphered는 유니코드가 아니라 바이트 스트림인 이진 데이터임

- msg를 10문자로 구성된 'python3xab'로 바꾸어 실행하면 오류 메시지 발생
- 암호화하려는 메시지 길이는 8바이트 배수여야 함

```
ValueError: Data must be padded to 8 byte boundary in CBC mode
```

- 문자열이 8바이트 배수가 아니더라도 오류없이 암호화/복호화 가능하도록 코드 수정이 필요함

➢ make8Sring(msg) 함수 추가

3DES 구현

```
def main():
        keytext = 'samsjang'
        iv = '1234'
        msg = 'python3x'
        #msg = 'python35ab'

        myCipher = myDES(keytext, iv)
        ciphered = myCipher.enc(msg)
        deciphered = myCipher.dec(ciphered)
        print('ORIGINAL:₩t%s' %msg)
        print('CIPHERED:₩t%s' %codecs.encode(ciphered, 'hex_codec'))
        print('DECIPHERED:₩t%s' %deciphered)
```

- **결과**

```
ORIGINAL:    python3x
CIPHERED:    b'539df2fe23f7fa06'
DECIPHERED: b'python3x'
```

3DES 구현

```
def make8String(msg):
        msglen = len(msg)
        filler = ''
        if msglen%8 != 0:
                filler = '0'*(8 - msglen%8)

        msg += filler

        return msg
```

- make8String(msg) : 문자열 msg의 길이를 8바이트 배수로 만들기 위해 문자 '0'을 추가한 후 리턴

```
def enc(self, plaintext):
    plaintext = make8String(plaintext)
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
    encmsg = des3.encrypt(plaintext.encode())
    return encmsg
```

- enc() 메서드 : plaintext = make8String(plaingtext) 추가
- main() 메서드 : msg = 'python35ab' 수정하여 실행

```
def main():
    keytext = 'samsjang'
    iv = '1234'
    #msg = 'python3x'
    msg = 'python35ab'

    myCipher = myDES(keytext, iv)
    ciphered = myCipher.enc(msg)
    deciphered = myCipher.dec(ciphered)
    print('ORIGINAL:\t%s' %msg)
    print('CIPHERED:\t%s' %codecs.encode(ciphered, 'hex_codec'))
    #print('CIPHERED:\t%s' % ciphered)
    print('DECIPHERED:\t%s' %deciphered)
```

- 결과

```
ORIGINAL:    python35ab
CIPHERED:    b'817d69664146d38c59af444efe4ee616'
DECIPHERED: b'python35ab000000'
```

- 맨 미자믹에 추가된 '0' 제외한 정보만을 읽어야 함

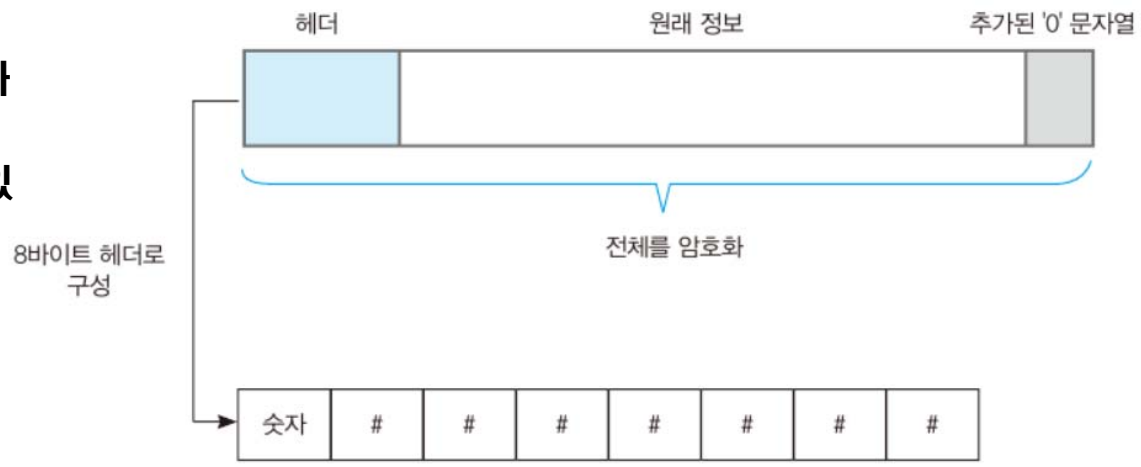
3DES 구현

➢ 데이터 무결성 보장 해결

- 원래 정보와 복원(복호화)된 정보의 동일함을 보장하는 것 (데이터 무결성 보장은 해시로 간단히 해결)
- make8String에서 추가된 문자 '0'의 개수를 알 수 있으므로, 원래 정보로 복원 가능

➢ "헤더+원래 정보+추가된 '0' 문자열 : 3DES 암호화

- 메타데이터는 헤더로 구성하여 원래 정보에 추가함
- 헤더 : 8바이트로 구성, 문자 '0'의 개수를 기록하고, 나머지 바이트는 '#' 문자로 채움



[그림 2.6] 3DES 암호화를 위한 헤더 구조

복호 과정

- 입력된 정보를 3DES 복호화
- 복호화 정보에서 첫 8바이트 추출
- 추출된 8바이트에서 '#' 문자 앞까지 읽어 추가된 '0' 개수 확인
- 남아있는 데이터에서 맨 미지막에 추가된 '0' 개수 제외한 정보를 읽음

단문암호 AES Python Ex.

Pycryptodome – 단문암호 AES

Crypto.Cipher.AES.new(*key*, *mode*, **args*, ***kwargs*)

Create a new AES cipher.

Parameters:

- **key** (*bytes/bytearray/memoryview*) –
  The secret key to use in the symmetric cipher.

  It must be 16, 24 or 32 bytes long (respectively for *AES-128*, *AES-192* or *AES-256*).

  For `MODE_SIV` only, it doubles to 32, 48, or 64 bytes.

- **mode** (One of the supported `MODE_*` constants) – The chaining mode to use for encryption or decryption. If in doubt, use `MODE_EAX`.

Keyword Arguments:

- **iv** (*bytes*, *bytearray*, *memoryview*) – (Only applicable for `MODE_CBC`, `MODE_CFB`, `MODE_OFB`, and `MODE_OPENPGP` modes).

  The initialization vector to use for encryption or decryption.

  For `MODE_CBC`, `MODE_CFB`, and `MODE_OFB` it must be 16 bytes long.

  For `MODE_OPENPGP` mode only, it must be 16 bytes long for encryption and 18 bytes for decryption (in the latter case, it is actually the *encrypted* IV which was prefixed to the ciphertext).

  If not provided, a random byte string is generated (you must then read its value with the `iv` attribute).

AES로 구현

key 크기 (128, 192, 256비트)와 암호화 블록 크기 (128비트) 제외하고, 3DES와 유사

AES import

클래스명 myAES

키와 초기화 벡터 16바이트로 생성

```
from Crypto.Cipher import AES
from Crypto.Hash import SHA256 as SHA
 import codecs
class myAES():
    def __init__(self, keytext, ivtext):
        hash = SHA.new()
        hash.update(keytext.encode('utf-8'))
        key = hash.digest()
        self.key = key[:16]

        hash.update(ivtext.encode('utf-8'))
        iv = hash.digest()
        self.iv = iv[:16]

    def makeEnabled(self, plaintext):
        fillersize = 0
        textsize = len(plaintext)
        if textsize%16 != 0:
            fillersize = 16-textsize%16

        filler = '0'*fillersize
        header = '%d' %(fillersize)
        gap = 16-len(header)
        header += '#'*gap

        return header+plaintext+filler
```

```
def enc(self, plaintext):
    plaintext = self.makeEnabled(plaintext)
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)
    encmsg = aes.encrypt(plaintext.encode())
    return encmsg


def dec(self, ciphertext):
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)
    decmsg = aes.decrypt(ciphertext)

    header = decmsg[:16].decode()
    fillersize = int(header.split('#')[0])
    if fillersize != 0:
        decmsg = decmsg[16:-fillersize]

    return decmsg
```

AES CBC 모드로 암호화

단문 메시지 길이에 상관없이 암호화 가능하도록 메시지 길이를 16바이트 배수로 만들고, 이에 대한 헤더정보를 포함하여 암호화

AES로 구현 key 크기 (128, 192, 256비트)와 암호화 블록 크기 (128비트) 제외하고, 3DES와 유사

```
def main():
    keytext = 'samsjang'
    iv = '1234'
    #msg = 'python35'
    msg = 'python35ab'

    myCipher = myAES(keytext, iv)
    ciphered = myCipher.enc(msg)
    deciphered = myCipher.dec(ciphered)
    print('ORIGINAL:\t%s' %msg)
    #print('CIPHERED:\t%s' %ciphered)
    print('CIPHERED:\t%s' %codecs.encode(ciphered, 'hex_codec'))
    print('DECIPHERED:\t%s' %deciphered)

if __name__ == '__main__':
    main()
```

- 결과

```
ORIGINAL:   python35ab
CIPHERED:   b'3d5194a98aa37c5be0a84cadc86e1ccd58ea5292da14c4a5f4dfdb6e201bfd3e'
DECIPHERED: b'python35ab'
```

AES로 구현

```
class myAES():
    def __init__(self, keytext, ivtext):
        hash = SHA.new()
        hash.update(keytext.encode('utf-8'))
        key = hash.digest()
        self.key = key[:16]

        hash.update(ivtext.encode('utf-8'))
        iv = hash.digest()
        self.iv = iv[:16]


    def makeEnabled(self, plaintext):
        fillersize = 0
        textsize = len(plaintext)
        if textsize%16 != 0:
            fillersize = 16-textsize%16

        filler = '0'*fillersize
        header = '%d' %(fillersize)
        gap = 16-len(header)
        header += '#'*gap

        return header+plaintext+filler
```

- AES CBC 모드로 암호화 및 복호화
- myAES 클래스 생성자에서 self.key 값을 16바이트로 만듬
- AES는 192비트(24바이트), 256비트(32바이트) 키를 지원함

```
self.key = key[:24]        # 192 비트 키로 암호화 할 경우
self.key = key             # 256 비트 키로 암호화 할 경우
```

- AES 블록 크기가 128비트이므로, 초기화 벡터는 16바이트 크기로 함

- plaintext의 크기가 16바이트 배수가 아닐 경우 끝에 '0' 문자를 추가(filler)하여 16바이트 배수로 만들고, '0'의 개수 정보를 헤더(header)로 구성한 후 리턴; header + plaintext + filler
- header 예 : '0'의 개수가 12개일 경우

```
header = '12###############'
```

AES로 구현

```
def enc(self, plaintext):
    plaintext = self.makeEnabled(plaintext)
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)
    encmsg = aes.encrypt(plaintext.encode())
    return encmsg
```

- 암호 하기 전 makeEnabled()로 plaintext를 "header+plaintext+filler"로 변환

```
def dec(self, ciphertext):
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)
    decmsg = aes.decrypt(ciphertext)

    header = decmsg[:16].decode()
    fillersize = int(header.split('#')[0])
    if fillersize != 0:
        decmsg = decmsg[16:-fillersize]

    return decmsg
```

- 복호화된 decmsg의 처음 16바이트를 유니코드로 변환하여 header 변수에 할당
- split('#')으로 heade를 '#' 구분자로 분리한 다음, 첫번째 값을 정수로 변환하여 fillersize 변수에 할당 – '0' 문자의 개수
- decmsg를 16-fillersize까지 슬라이싱 하여 원래 정보 추출

Pycryptodome – 단문암호 3DES

3DES 구현 'python3x'의 8문자로 된 문자열을 3DES CBC 모드로 암호화하고, 3DES로 복호화

```
from Crypto.Cipher import DES3
from Crypto.Hash import SHA256 as SHA
import codecs

class myDES():
        def __init__(self, keytext, ivtext):
                hash = SHA.new()
                hash.update(keytext.encode('utf-8'))
                key = hash.digest()
                self.key = key[:24]

                hash.update(ivtext.encode('utf-8'))
                iv = hash.digest()
                self.iv = iv[:8]


        def enc(self, plaintext):
                #plaintext = make8String(plaintext)
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                encmsg = des3.encrypt(plaintext.encode())
                return encmsg


        def dec(self, ciphertext):
                des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)
                decmsg = des3.decrypt(ciphertext)
                return decmsg
```

```
def make8String(msg):
        msglen = len(msg)
        filler = ''
        if msglen%8 != 0:
                filler = '0'*(8 - msglen%8)

        msg += filler

        return msg

def main():
        keytext = 'samsjang'
        iv = '1234'
        msg = 'python3x'
        #msg = 'python35ab'

        myCipher = myDES(keytext, iv)
        ciphered = myCipher.enc(msg)
        deciphered = myCipher.dec(ciphered)
        print('ORIGINAL:%s' %msg)
        print('CIPHERED:%s' %codecs.encode(ciphered, 'hex_codec'))
        print('DECIPHERED:%s' %deciphered)

if __name__ == '__main__':
        main()
```

7

여기에 슬라이드 노트의 내용을 입력하십시오

교과목 : 정보보호

PyCryptodome
3 파일암호 3DES, AES

2023학년도 2학기
Suk-Hwan Lee

- PyCryptodome – Cryptographic library for Python

참조자료

- PyCryptodome Homepage : https://www.pycryptodome.org/en/latest/
- Download : https://pypi.org/project/pycryptodome/
- GitHub : https://github.com/Legrandin/pycryptodome

코드 참조

- https://www.pycryptodome.org/en/latest/src/examples.html
- 화이트 해커를 위한 암호와 해킹 (2판), White Hat Python, 장삼용 저, 정보문화사, 2019년9월

파일암호 3DES Python Ex.

Pycryptodome – 파일암호 3DES

3DES 구현

텍스트 파일을 읽어
모든 내용을 3DES로
암호화

파일 크기를 리턴하는 함수 getsize()
이용하기 위해 os.path 모듈 import

전역변수 KSIZE 정의

❖ 모드 크기의 파일에 대해 3DES로
암호/복호 가능한 코드임

```python
from Crypto.Cipher import DES3
from Crypto.Hash import SHA256 as SHA
from os import path

KSIZE = 1024

class myDES():
    def __init__(self, keytext, ivtext):
        hash = SHA.new()
        hash.update(keytext.encode('utf-8'))
        key = hash.digest()
        self.key = key[:24]

        hash.update(ivtext.encode('utf-8'))
        iv = hash.digest()
        self.iv = iv[:8]

    def makeEncInfo(self, filename):
        fillersize = 0
        filesize = path.getsize(filename)
        if filesize%8 != 0:
            fillersize = 8-filesize%8

        filler = '0'*fillersize
        header = '%d' %(fillersize)
        gap = 8-len(header)
        header += '#'*gap

        return header, filler
```

암호 파일은 enc 파일로 저장

```python
def enc(self, filename):
    encfilename = filename + '.enc'
    header, filler = self.makeEncInfo(filename)
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)

    h = open(filename, 'rb')
    hh = open(encfilename, 'wb+')

    enc = header.encode('utf-8')
    content = h.read(KSIZE)
    content = enc + content
    while content:
        if len(content) < KSIZE:
            content += filler.encode('utf-8')

        enc = des3.encrypt(content)
        hh.write(enc)
        content = h.read(KSIZE)

    h.close()
    hh.close()
```

3DES 구현

plain.txt

For seven days and seven nights
Man will watch this awesome sight.
The tides will rise beyond their ken
To bite away the shores and then

A fiery dragon will cross the sky
Six times before this earth shall die
Mankind will tremble and frightened be
for the sixth heralds in this prophecy.

The great star will burn for seven days,
The cloud will cause two suns to appear
The big mastiff will howl all night
When the great pontiff will change country.

복호 파일은 dec 파일로 저장

```
def dec(self, encfilename):
    filename = encfilename + '.dec'
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)

    h = open(filename, 'wb+')
    hh = open(encfilename, 'rb')

    content = hh.read(8)
    dec = des3.decrypt(content)
    header = dec.decode()
    fillersize = int(header.split('#')[0])

    content = hh.read(KSIZE)
    while content:
        dec = des3.decrypt(content)
        if len(dec) < KSIZE:
            if fillersize != 0:
                dec = dec[:-fillersize]

        h.write(dec)
        content = hh.read(KSIZE)

    h.close()
    hh.close()
```

```
def main():
    keytext = 'samsjang'
    ivtext = '1234'
    filename = 'plain.txt'
    encfilename = filename + '.enc'

    myCipher = myDES(keytext, ivtext)
    myCipher.enc(filename)
    myCipher.dec(encfilename)

if __name__ == '__main__':
    main()
```

3DES 구현

```
def makeEncInfo(self, filename):
    fillersize = 0
    filesize = path.getsize(filename)
    if filesize%8 != 0:
        fillersize = 8-filesize%8

    filler = '0'*fillersize
    header = '%d' %(fillersize)
    gap = 8-len(header)
    header += '#'*gap

    return header, filler
```

- filename으로 지정된 파일 크기를 구하고,
- 파일 크기가 8바이트 배수가 아닐 경우 8바이트 배수로 만들기 위해 '0' 문자열 구성
- 추가할 문자 '0' 개수에 대하여 헤더 만들고, 헤더와 추가할 '0' 문자열을 리턴

Pycryptodome – 파일암호 3DES

3DES 구현

```
def enc(self, filename):
    encfilename = filename + '.enc'
    header, filler = self.makeEncInfo(filename)
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)

    h = open(filename, 'rb')
    hh = open(encfilename, 'wb+')

    enc = header.encode('utf-8')
    content = h.read(KSIZE)
    content = enc + content
    while content:
        if len(content) < KSIZE:
            content += filler.encode('utf-8')

        enc = des3.encrypt(content)
        hh.write(enc)
        content = h.read(KSIZE)

    h.close()
    hh.close()
```

- filename으로 지정된 파일 내용을 1KB씩 읽어서 3DES로 암호화한 후 enc 파일로 저장함

- 암호 파일명 (filename+.enc) 지정하고, filename 지정한 파일에 대한 header, filler 구함
- DES3.new로 객체 (des3) 생성

- KSIZE 만큼 읽고 → 암호화 → 저장 → 파일 다 읽을 때까지 반복
  - ✓ 파일에서 1KB(KSIZE=1024)만큼 읽어서 content에 저장하고, header를 content 앞에 추가
  - ✓ 파일 내용이 KSIZE(=1KB) 미만이면, file.read()는 남아있는 크기만큼 모두 읽고, '0'문자열을 content에 추가
  - ✓ content를 3DES로 암호화하고 파일에 저장한 후, 파일에서 다시 1KB 만큼 읽어 content에 저장

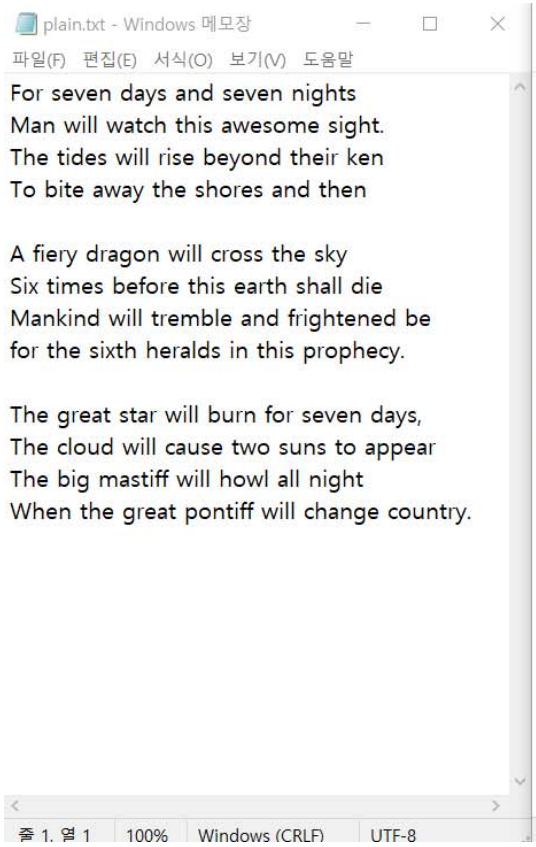
3DES 구현

```
def dec(self, encfilename):
    filename = encfilename + '.dec'
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)

    h = open(filename, 'wb+')
    hh = open(encfilename, 'rb')

    content = hh.read(8)
    dec = des3.decrypt(content)
    header = dec.decode()
    fillersize = int(header.split('#')[0])

    content = hh.read(KSIZE)
    while content:
        dec = des3.decrypt(content)
        if len(dec) < KSIZE:
            if fillersize != 0:
                dec = dec[:-fillersize]

        h.write(dec)
        content = hh.read(KSIZE)

    h.close()
    hh.close()
```

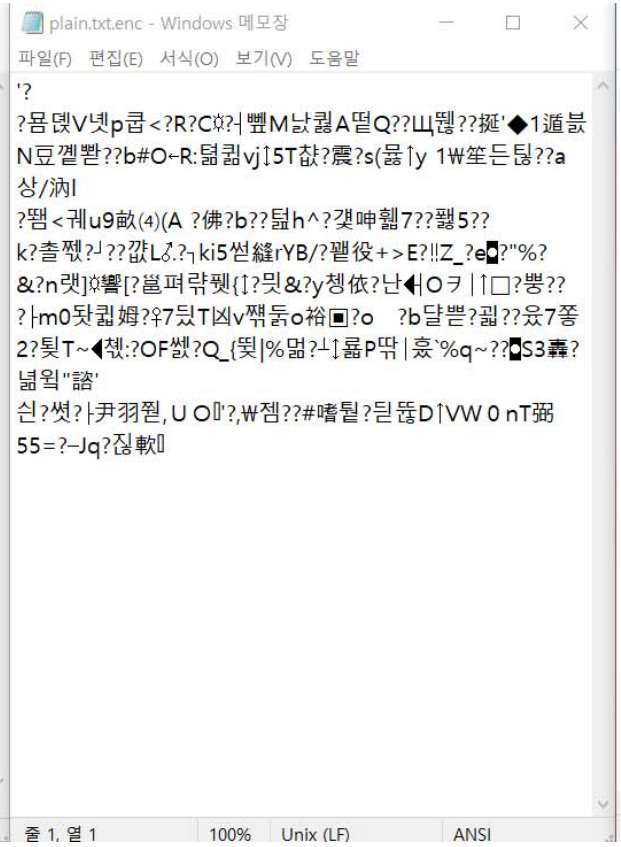
- encfilename으로 지정된 암호 파일 내용을 1KB씩 읽어서 3DES로 복호화한 후 dec 파일로 저장함
- 복호 파일명 (encfilename+.dec) 지정하고, DES3.new로 객체 (des3) 생성
- 암호 파일에서 첫 8바이트 읽어 3DES로 복호
  - ✓ Header로 '#' 구분자로 Header를 분리한 후 첫번째 멤버를 정수로 변환 – 파일 끝부분에 추가된 '0' 문자의 개수임
- KSIZE 만큼 읽고 → 복호화 → 저장 → 파일 다 읽을 때까지 반복
  - ✓ 파일에서 1KB(KSIZE=1024)만큼 읽어서 content에 저장
  - ✓ conten를 복호하여 dec에 저장
  - ✓ dec가 1KB보다 작으면 (복호화 파일의 마지막 부분임), 암호화 때 추가된 '0' 문자열 제거하고 파일에 저장
  - ✓ 다음 1KB 만큼 읽고 복호화하여 저장함. content 내용이 없을 때까지 반복 수행

3DES 결과

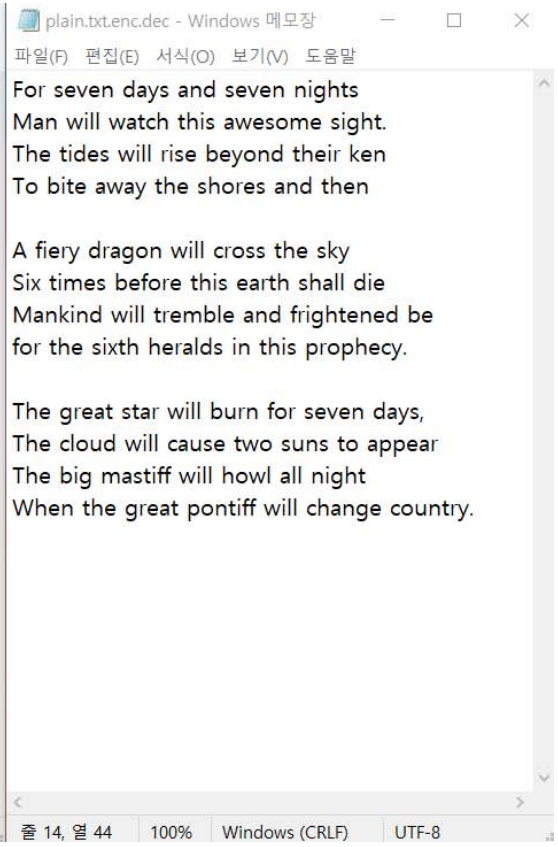
plain.txt (463Byte)



plain.txt.enc (472Byte)



plain.txt.enc.dec (463Byte)



파일암호 AES Python Ex.

Pycryptodome – 파일암호 AES

AES 구현

텍스트 파일을 읽어 모든 내용을 AES로 암호화

3DES 코드와 다른 부분만 체크

```python
from Crypto.Cipher import AES
from Crypto.Hash import SHA256 as SHA
from os import path
KSIZE = 1024

class myAES():
    def __init__(self, keytext, ivtext):
        hash = SHA.new()
        hash.update(keytext.encode('utf-8'))
        key = hash.digest()
        self.key = key[:16]

        hash.update(ivtext.encode('utf-8'))
        iv = hash.digest()
        self.iv = iv[:16]

    def makeEncInfo(self, filename):
        fillersize = 0
        filesize = path.getsize(filename)
        if filesize%16 != 0:
            fillersize = 16-filesize%16

        filler = '0'*fillersize
        header = '%d' %(fillersize)
        gap = 16-len(header)
        header += '#'*gap

        return header, filler
```

```python
def enc(self, filename):
    encfilename = filename + '.enc'
    header, filler = self.makeEncInfo(filename)
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)

    h = open(filename, 'rb')
    hh = open(encfilename, 'wb+')

    enc = header.encode('utf-8')
    content = h.read(KSIZE)
    content = enc + content
    while content:
        if len(content) < KSIZE:
            content += filler.encode('utf-8')

        enc = aes.encrypt(content)
        hh.write(enc)
        content = h.read(KSIZE)

    h.close()
    hh.close()
```

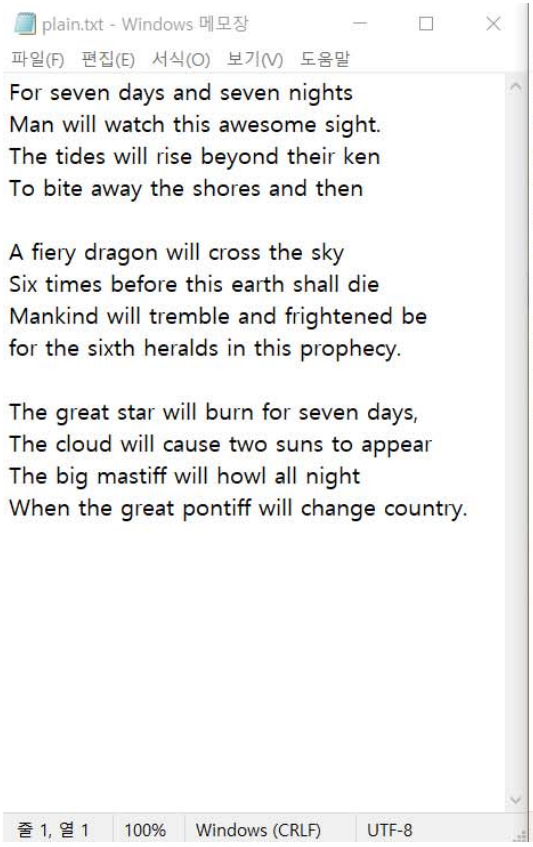
AES 구현

```
def dec(self, encfilename):
    filename = encfilename + '.dec'
    aes = AES.new(self.key, AES.MODE_CBC, self.iv)

    h = open(filename, 'wb+')
    hh = open(encfilename, 'rb')

    content = hh.read(16)
    dec = aes.decrypt(content)
    header = dec.decode()
    fillersize = int(header.split('#')[0])

    content = hh.read(KSIZE)
    while content:
        dec = aes.decrypt(content)
        if len(dec) < KSIZE:
            if fillersize != 0:
                dec = dec[:-fillersize]
        h.write(dec)
        content = hh.read(KSIZE)

    h.close()
    hh.close()
```

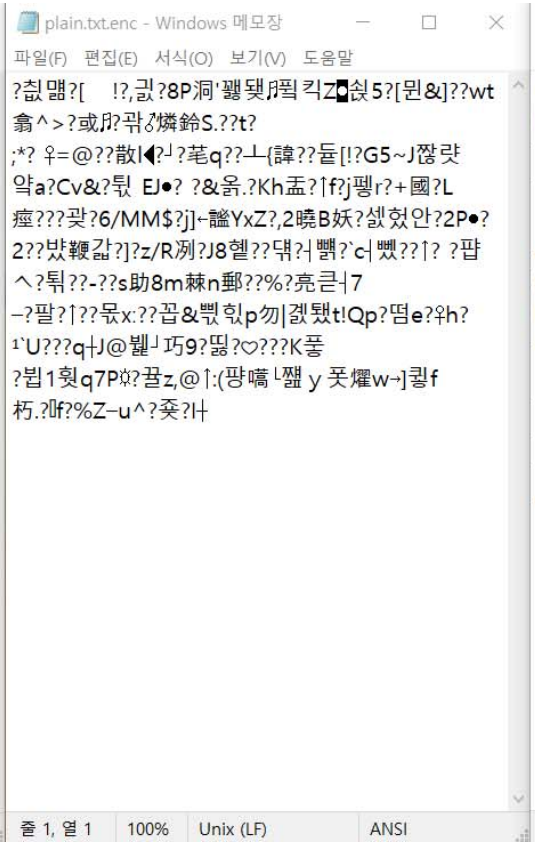
```
def main():
    keytext = 'samsjang'
    ivtext = '1234'
    filename = 'plain.txt'
    encfilename = filename + '.enc'

    myCipher = myAES(keytext, ivtext)
    myCipher.enc(filename)
    myCipher.dec(encfilename)

if __name__ == '__main__':
    main()
```

AES 결과

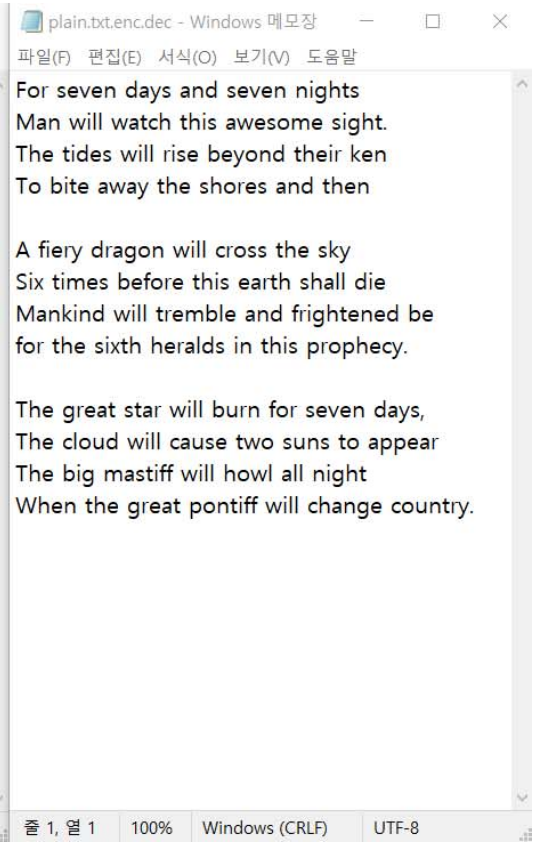
plain.txt (463Byte)



plain.txt.enc (480Byte)



plain.txt.enc.dec (463Byte)



Pycryptodome – 파일암호 3DES

3DES 구현

텍스트 파일을 읽어
모든 내용을 3DES로
암호화

파일 크기를 리턴하는 함수 getsize()
이용하기 위해 os.path 모듈 import

전역변수 KSIZE 정의

❖ 모드 크기의 파일에 대해 3DES로
암호/복호 가능한 코드임

```
from Crypto.Cipher import DES3
from Crypto.Hash import SHA256 as SHA
from os import path

KSIZE = 1024

class myDES():
    def __init__(self, keytext, ivtext):
        hash = SHA.new()
        hash.update(keytext.encode('utf-8'))
        key = hash.digest()
        self.key = key[:24]

        hash.update(ivtext.encode('utf-8'))
        iv = hash.digest()
        self.iv = iv[:8]

    def makeEncInfo(self, filename):
        fillersize = 0
        filesize = path.getsize(filename)
        if filesize%8 != 0:
            fillersize = 8-filesize%8

        filler = '0'*fillersize
        header = '%d' %(fillersize)
        gap = 8-len(header)
        header += '#'*gap

        return header, filler
```

암호 파일은 enc 파일로 저장

```
def enc(self, filename):
    encfilename = filename + '.enc'
    header, filler = self.makeEncInfo(filename)
    des3 = DES3.new(self.key, DES3.MODE_CBC, self.iv)

    h = open(filename, 'rb')
    hh = open(encfilename, 'wb+')

    enc = header.encode('utf-8')
    content = h.read(KSIZE)
    content = enc + content
    while content:
        if len(content) < KSIZE:
            content += filler.encode('utf-8')

        enc = des3.encrypt(content)
        hh.write(enc)
        content = h.read(KSIZE)

    h.close()
    hh.close()
```

4

여기에 슬라이드 노트의 내용을 입력하십시오