Assignment #1 Cryptography

2018008059 김은수

1. Symmetric encryption

(1) DES

input으로 받은 데이터는 original data 변수에 저장되며 cipher type으로 DES를 입력했을 시 DES 암호화가 진행된다.

먼저, input으로 key를 받고 make key 함수를 통해 key를 가공한 뒤 key = key[:8]을 통해 8byte의 key 값으로 만들어준다. 굳이 make key 로 key를 가공해주는 이유는 사용자의 편리를 위해 key값 입력길이에 제안이 없도록 하기 위해서이다. make key 함수는 key 값을 hash함수 SHA256을 통해 hashing 해준 뒤 16진수의 문자열로 key를 반환해준다. 따라서 다음에 사용자가 똑같은 key를 입력해준다면 이 make key 함수를 통해 같은 hash값이 나오기 때문에 같은 암호문자를 낼 수 있다.

```
def makekey(key):
    hash_value = hashlib.sha256(key.encode())
    key = hash_value.digest()
    return key
```

다음으로 DES.new(key, DES.MODE_ECB)로 암호화 객체 des를 만들어 준다. 또한 original data를 pad 함수를 이용해서 8바이트 배수 단위로 만들어준다. 그러기 위해서 인자로 num에 8을 넣어서 호출해준다. pad함수는 모자란 부분을 ' '으로 채워준다.

```
def pad(text,num):
    while len(text) % num != 0:
        text += ' '
    return text
```

Original data를 padding 한 값을 padded_text에 할당해준 뒤 padded_text를 인코딩하여 바이트 코드로 변환해준 뒤 encrypt 해준다. Encrypt된 text를 출력한 뒤 decrypt 해준다. Decrypt된 text를 디코딩하여 출력한다.

```
padded_text = pad(original_data,8)
encrypted_text = des.encrypt(padded_text.encode())
print("encrypted: ", end='')
print(encrypted_text)
decrypted_text = des.decrypt(encrypted_text)
print("decrypted: "+decrypted_text.decode())
```

- DES 전체 코드 및 실행 예시

```
if cipher_type == "DES":
    key = input("key: ")
    key = makekey(key)
    key = key[:8]
    des = DES.new(key, DES.MODE_ECB)
    padded_text = pad(original_data,8)
    encrypted_text = des.encrypt(padded_text.encode())
    print("encrypted: ", end='')
    print(encrypted_text)
    decrypted_text = des.decrypt(encrypted_text)
    print("decrypted: "+decrypted_text.decode())
```

```
C:\Users\김은수\PycharmProjects\happy\venv\Scripts\python.exe
original data: Hello World!
cipher type(DES/DES3/AES/ARC4): DES
key: 12341234
encrypted: b'\xde7\xc3\xd6.\xd9z\xb8i\x14So\xc7\xb4y\xd1'
decrypted: Hello World!
```

(2) DES3

cipher type으로 DES3을 입력했을 시 DES3 암호화가 진행된다.

먼저, input으로 key를 받고 make key 함수를 통해 key를 가공한 뒤 key = key[:24]를 통해 Pycrpto에서 제공하는 3DESkey 크기인 24byte의 크기로 만들어준다. 마찬가지로 make key 로 key를 가공해주는 이유는 key값 입력길이에 제안이 없도록 하기 위해서이다. 또한, 초기화 벡터 IV를 선언해준다. IV는 "0123456789"를 make key 함수를 통해 hashing한 다음 8byte로 슬라이싱하여 할당한 값이다. 3DES는 8바이트 암호화 블록 값을 가지므로 초기화 벡터 또한 8바이트가 되어야하기 때문에 슬라이싱을 하였다.

```
key = input("key: ")
key = makekey(key)
key = key[:24]
IV = "0123456789"
IV = makekey(IV)
IV = IV[:8]
```

DES3.new(key, DES3.MODE_CBC, IV)로 DES3 객체를 만들어준다. 운영모드는 수업시간에 배운 CBC, cipher block chaining mode를 선택하였다. 다음으로 original data를 8바이트 배수로 만들기위해 padding을 하였고 그 값을 padded_text에 할당해준 뒤 padded_text를 인코딩하여 바이트 코드로 변환해준 뒤 encrypt 해준다. Encrypt된 text를 출력한 뒤 decrypt해준다. Decrypt된 text를 디코딩하여 출력한다.

- DES3 전체 코드 및 실행 예시

```
elif cipher_type == "DES3":
    key = input("key: ")
    key = makekey(key)
    key = key[:24]
    IV = "0123456789"
    IV = makekey(IV)
    IV = IV[:8]
    des3 = DES3.new(key, DES3.MODE_CBC, IV)
    padded_text = pad(original_data, 8)
    encrypted_text = des3.encrypt(padded_text.encode())
    print("encrypted: ", end='')
    print(encrypted_text)
    des3 = DES3.new(key, DES3.MODE_CBC, IV)
    decrypted_text = des3.decrypt(encrypted_text)
    print("decrypted: " + decrypted_text.decode())
```

```
original data: Hello World!
cipher type(DES/DES3/AES/ARC4): DES3
key: 123412341234
encrypted: b'C\xafM\x8f\xe1k\xc4\xc1_\xd6 \xd2\xe2\xf2\x9f~'
decrypted: Hello World!
```

(3) AES

cipher type으로 AES을 입력했을 시 AES 암호화가 진행된다.

먼저, input으로 key를 받는다. AES는 16byte, 24byte, 32byte의 키 사이즈를 가질 수 있다. 사용자가 보안상 이유로 key길이를 선택할 수 있으므로 keylen 변수로 사용자가 입력한 byte를 저장한다. make key 함수를 통해 key를 가공한 뒤 입력한 key가 16,24,32 바이트로 AES로 사용할 수 있는 key size면 그 만큼 슬라이싱을 해준다. 만약 아니라면 key = key[:16]를 통해 임의로 16byte로 만들어준다. 또한, 초기화 벡터

IV를 선언해준다. IV는 "1234"를 make key 함수를 통해 hashing한 다음 16byte로 슬라이싱하여 할당한 값이다. AES는 16바이트 암호화 블록 값을 가지므로 초기화 벡터 또한 16바이트가 되어야하기 때문에 슬라이싱을 하였다.

```
elif cipher_type == "AES":
    key = input("key(16/24/32): ")
    keylen = len(key)
    key = makekey(key)
    if keylen ==16 or keylen==24 or keylen==32:
        key=key[:keylen]
    else:
        key = key[:16]

    IV = "1234"
    IV = makekey(IV)
    IV = IV[:16]
```

AES.new(key, AES.MODE_CBC, IV)로 AES 객체를 만들어준다. 다음으로 original data를 16바이트 배수로 만들기위해 padding을 하였다. pad함수에 num인자값으로 16을 넣어주었다. 그 값을 padded_text에 할당해준 뒤 padded_text를 인코딩하여 바이트 코드로 변환해준 뒤 encrypt 해준다. Encrypt된 text를 출력한 뒤 decrypt해준다. Decrypt된 text를 디코딩하여 출력한다.

- AES 전체 코드 및 실행 예시

```
elif cipher_type == "AES":
    key = input("key(16/24/32): ")
    keylen = len(key)
    key = makekey(key)
    if keylen ==16 or keylen==24 or keylen==32:
        key=key[:keylen]
    else:
        key = key[:16]

    IV = "1234"
    IV = makekey(IV)
    IV = IV[:16]

    aes = AES.new(key, AES.MODE_CBC, IV)
    padded_text = pad(original_data, 16)
    encrypted_text = aes.encrypt(padded_text.encode())
    print("encrypted: ", end='')
    print(encrypted_text)

    aes = AES.new(key, AES.MODE_CBC, IV)
    decrypted_text = aes.decrypt(encrypted_text)
    print("decrypted: " + decrypted_text.decode())
```

```
original data: Hello World!
cipher type(DES/DES3/AES/ARC4): AES
key(16/24/32): 1234123412341234
encrypted: b'\xe4\x04\xb9U\x16:\xfc\x01f\xed\xd8\x95\xf2\xfc\x1d\xf5'
decrypted: Hello World!
```

(4) ARC4

cipher type으로 AES을 입력했을 시 AES 암호화가 진행된다.

마찬가지로 key를 입력 받는다. ARC4에서는 40에서 2048bit까지 key 길이로 사용할 수 있다. 따라서 key 값을 5글자 이상 256글자 이하로 입력을 받고 이는 위에 함수들과 다르게 makekey 함수로 hash하지 않고 바로 encode해서 넣어주었다. (위에 다른 암호화 방법도 이와 같이 makekey 함수를 지우고 바로 key 자체에 key = key.encode()를 해서 문제없이 사용할 수 있다.) 그 다음 ARC4.new(key) 로 arc4 객체를 만들어준 뒤 encrypt(original_data.encode())를 통해 암호화해서 출력하고 decrypt(암호화된 문장) 을 통해 복호화 했다. 패딩을 하지 않은 이유는 ARC4는 암호 블록 크기가 1바이트 이기 때문에 1문자 이상이면 암호화, 복호화가 가능하기 때문이다.

- ARC4 전체 코드 및 실행 예시

```
elif cipher_type == "ARC4":
    key = input("key(more than 5 letter, less than 256 letter): ")
    key= key.encode()
    arc4=ARC4.new(key)
    encrypted_text = arc4.encrypt(original_data.encode())
    print("encrypted: ", end='')
    print(encrypted_text)
    arc4 = ARC4.new(key)
    decrypted_text = arc4.decrypt(encrypted_text)
    print("decrypted: " + decrypted_text.decode())
```

```
original data: Hello World!
cipher type(DES/DES3/AES/ARC4): ARC4
key(more than 5 letter, less than 256 letter): 1234512345
encrypted: b'\x18PT\x0eC>[\x00\xe9O\xb0\x91'
decrypted: Hello World!
```

2. Hash 함수

(1) SHA, SHA256, SHA384, SHA512, HMAC

입력으로 hash_type을 받고 이에 따라서 hash를 수행한다. SHA는 hashlib 모듈을 통해 Hash 유형마다 이름이 지정된 생성자 메서드를 호출하여 hash값을 만들고 그 결과를 hesdigest()를 통해 출력한다. HMAC은 hamc 모듈을 통해 hashing 해준다.

- 전체 코드 및 실행 예시 (original data는 Hello World!)

```
def hash_value(hash_type):
    if hash_type == "SHA":
        hash_result = hashlib.sha1(original_data.encode())
        print(hash_result.hexdigest())
    elif hash_type == "SHA256":
        hash_result = hashlib.sha256(original_data.encode())
        print(hash_result.hexdigest())
    elif hash_type == "SHA384":
        hash_result = hashlib.sha384(original_data.encode())
        print(hash_result.hexdigest())
    elif hash_type == "SHA512":
        hash_result = hashlib.sha512(original_data.encode())
        print(hash_result.hexdigest())
    elif hash_type == "HMAC":
        hash_result = hmac.new(original_data.encode())
        print(hash_result.hexdigest())
```

- SHA 예시, SHA256 예시

```
hash type(SHA/SHA256/SHA384/SHA512/HMAC): SHA
2ef7bde608ce5404e97d5f042f95f89f1c232871
```

```
hash type(SHA/SHA256/SHA384/SHA512/HMAC): SHA256
7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069
```

- SHA384 예시, SHA512 예시

```
hash type(SHA/SHA256/SHA384/SHA512/HMAC): SHA384
bfd76c0ebbd006fee583410547c1887b0292be76d582d96c242d2a792723e3fd6fd061f9d5cfd13b8f961358e6adba4a
```

```
hash type(SHA/SHA256/SHA384/SHA512/HMAC): SHA512
861844d6704e8573fec34d967e20bcfef3d424cf48be04e6dc08f2bd58c729743371015ead891cc3cf1c9d34b49264b510751b1ff9e537937bc46b5d6ff4ecc8
```

- HMAC 예시

```
hash type(SHA/SHA256/SHA384/SHA512/HMAC): HMAC
2f5cc07a72c395419b2cac85707ca475
```

3. Asymmetric encryption

RSA 암호화를 진행한다. 우선 key 길이를 입력 받는다. 그리고 RSA.generator를 통해서 입력 받은 만큼의 bit사이즈의 private key를 생성한다. 그리고 privatekey.publickey()를 통해 개인키에서 공개키를 불러온다. 그 다음 암호화 method를 얻기 위해 PKCS1_OAEP.new(public_key)를 실행하고 얻은 return 값에 encrypt를 통해 암호화한 후 출력한다. 공개키로 암호화한 값을 복호화 하기 위해서는 개인키로 해야 한다. 따라서 PKCS1_OAEP.new(private_key)를 통해 얻은 return값에 decrypt를 통해 복호화 한 후 출력한다.

- 전체 코드 및 실행 예시

```
def RSA_func():
    number = input("key length(x256, >=1024): ")
    private_key = RSA.generate(int(number))
    public_key = private_key.publickey()
    encryptor = PKCS1_OAEP.new(public_key)
    encrypted = encryptor.encrypt(original_data.encode())
    print("encrypted:", end='')
    print(encrypted)
    decryptor = PKCS1_OAEP.new(private_key)
    decrypted = decryptor.decrypt(encrypted)
    print("decrypted: ", end='')
    print(decrypted.decode())
```

```
RSA
key length(x256, >=1024): 1024
encrypted:b'#\x03\x1f\xd7\xa2\xaf\xae\xedW\x13\xc0\x16\x83\x03\xcbUlH3,
 \xa8V`\x01\xa8\x9c\xaf\xb0\xd1J\x00\xa48x\x1a\xe0/\xea\x18\xce5IEF\xbe\xaeD2\x9el\xc6\xffd\xc4c\x1bBQ
 (\x1d\xa2-\xe9q\x83\xf8\x8666\xe4+\xd3\x95I\xd2\xea\xd1\xe7&:\x00\x97\xec\xa3\xc3\xf8;
 \x01\x81D\x9cWv\x82\x15\xac\xf5\xb9c\x8b\x1b\xc1?V\x91\xd7$\x8c\xf1\xcf\xfe\xe1[\x8b]\xd3mOe\xb7m.X*\xe7\xc0\xf1\x11'
decrypted: Hello World!
```

4. 컴파일 환경 및 main 함수 진행

- Python 3.5
- Ezhashlib, pycrypto, pycryptodome install 후 진행.

```python
if __name__ == "__main__":
    original_data = input("original data: ")
    cipher_type = input("cipher type(DES/DES3/AES/ARC4): ")
    cipher_func(cipher_type)
    print("\n")
    hash_type = input("hash type(SHA/SHA256/SHA384/SHA512/HMAC): ")
    hash_value(hash_type)
    print("\n")
    print("RSA")
    RSA_func()
```

input으로 original data를 받은 뒤 symmetric encryption type을 입력 받는다. 그 다음 cipher_func이라는 DES, DES3, AES, ARC4를 수행하는 함수에 인자로 넣어준다. Cipher_func에서는 인자로 받은 암호화를 실행한다. 그 다음 input으로 hash type을 입력 받고 hash_value라는 함수에 인자로 넣어준다. Hash_value함수에서는 입력한 hash type에 맞게 hash값을 리턴 해준다. 다음으로 "RSA"를 출력하고 RSA_func을 통해 RSA를 실행해준다.

- 명령 프롬프트 실행 모습