과제 수행 내용: 파이썬 언어를 사용하여 AES 암호화/복호화 함수를 구현하고 16바이트 크기의 key를 사용하여 평문 암호화/복호화를 수행하였습니다.

암호화/복호화 코드:

- 평문: “My Name is YoungMin Jeon”
- Key: “WhisperingSecret”

```python
import aes, os

keyword = 'WhisperingSecret'
hex_keyword = keyword.encode('utf-8').hex()
key = bytes.fromhex(hex_keyword)
# key = os.urandom(16)
iv = os.urandom(16)

plain = b"My Name is YoungMin Jeon"

print("Plain Text:", plain)
print("Key:", key)

encrypted = aes.AES(key).encrypt_ctr(plain, iv)
print("Encrypted Text:", encrypted)

restored = aes.AES(key).decrypt_ctr(encrypted, iv)
print("Decrypted Text:", restored)
```

수행 결과:

```
(py310) C:\Users\mkjsy\Desktop\YM\Source Code\VSCode>C:/Users/mkjsy/anaconda3/envs/py310/pyt
Plain Text: b'My Name is YoungMin Jeon'
Key: b'WhisperingSecret'
Encrypted Text: b'\xf8}!\x89\x83\x85\x07\xae\x01\xbaCA+\xd2\x96\xda\xcb\x14\xcc\xd9)\x81*F'
Decrypted Text: b'My Name is YoungMin Jeon'
```

1. S-box 및 역 S-box (s_box, inv_s_box)

```
s_box = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16,
)

inv_s_box = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
)
```

- sub_bytes 및 inv_sub_bytes 변환에 사용되는 조회 테이블입니다.
- sub_bytes는 상태 행렬의 각 바이트를 s_box의 해당 바이트로 대체합니다. 이는 암호화 프로세스에 비선형성을 제공하여 해독을 어렵게 만듭니다.
- inv_sub_bytes는 복호화 중에 inv_s_box를 사용하여 역순으로 수행합니다.

2. shift_rows 및 inv_shift_rows

```
68  def shift_rows(s):
69      s[0][1], s[1][1], s[2][1], s[3][1] = s[1][1], s[2][1], s[3][1], s[0][1]
70      s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
71      s[0][3], s[1][3], s[2][3], s[3][3] = s[3][3], s[0][3], s[1][3], s[2][3]
72
73
74  def inv_shift_rows(s):
75      s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
76      s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
77      s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]
```

- 상태 행렬의 행을 순환적으로 이동시키는 함수입니다.
- shift_rows는 두 번째 행을 1바이트, 세 번째 행을 2바이트, 네 번째 행을 3바이트씩 왼쪽으로 이동시킵니다.
- inv_shift_rows는 복호화 중에 역방향 이동을 수행합니다.

3. add_round_key

```
79  def add_round_key(s, k):
80      for i in range(4):
81          for j in range(4):
82              s[i][j] ^= k[i][j]
```

- 상태 행렬과 라운드 키 사이에 비트별 XOR 연산을 수행합니다.
- 라운드 키는 기본 암호화 키에서 파생되며 암호화 프로세스의 각 라운드에서 변경됩니다.

4. xtime

```
86  xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)
87
88
```

- mix_columns 단계에서 사용되는 유한 필드 GF(2^8)에서 바이트에 2를 곱하는 람다 함수입니다.

5. mix_single_column 및 mix_columns

```
89   def mix_single_column(a):
90       # see Sec 4.1.2 in The Design of Rijndael
91       t = a[0] ^ a[1] ^ a[2] ^ a[3]
92       u = a[0]
93       a[0] ^= t ^ xtime(a[0] ^ a[1])
94       a[1] ^= t ^ xtime(a[1] ^ a[2])
95       a[2] ^= t ^ xtime(a[2] ^ a[3])
96       a[3] ^= t ^ xtime(a[3] ^ u)
97
98
99   def mix_columns(s):
100      for i in range(4):
101          mix_single_column(s[i])
```

- mix_columns는 상태 행렬의 각 열에 선형 변환을 적용합니다.
- mix_single_column을 사용하여 각 열에 대해 GF(2^8)에서 행렬 곱셈을 수행합니다. 이 단계는 상태 행렬 내에서 데이터를 확산시켜 차분 및 선형 암호 분석에 대한 암호화 저항을 증가시킵니다.

6. inv_mix_columns

```
104  def inv_mix_columns(s):
105      # see Sec 4.1.3 in The Design of Rijndael
106      for i in range(4):
107          u = xtime(xtime(s[i][0] ^ s[i][2]))
108          v = xtime(xtime(s[i][1] ^ s[i][3]))
109          s[i][0] ^= u
110          s[i][1] ^= v
111          s[i][2] ^= u
112          s[i][3] ^= v
113
114      mix_columns(s)
```

- 복호화 중에 mix_columns 연산의 역을 수행하는 함수입니다.

7. r_con

```
117    r_con = (
118        0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
119        0x80, 0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A,
120        0x2F, 0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A,
121        0xD4, 0xB3, 0x7D, 0xFA, 0xEF, 0xC5, 0x91, 0x39,
122    )
```

- 키 확장 프로세스에 사용되는 라운드 상수를 보유하는 테이블입니다.

8. 도우미 함수

```python
def bytes2matrix(text):
    """ Converts a 16-byte array into a 4x4 matrix.  """
    return [list(text[i:i+4]) for i in range(0, len(text), 4)]

def matrix2bytes(matrix):
    """ Converts a 4x4 matrix into a 16-byte array.  """
    return bytes(sum(matrix, []))

def xor_bytes(a, b):
    """ Returns a new byte array with the elements xor'ed. """
    return bytes(i^j for i, j in zip(a, b))

def inc_bytes(a):
    """ Returns a new byte array with the value increment by 1 """
    out = list(a)
    for i in reversed(range(len(out))):
        if out[i] == 0xFF:
            out[i] = 0
        else:
            out[i] += 1
            break
    return bytes(out)

def pad(plaintext):
    """
    Pads the given plaintext with PKCS#7 padding to a multiple of 16 bytes.
    Note that if the plaintext size is a multiple of 16,
    a whole block will be added.
    """
    padding_len = 16 - (len(plaintext) % 16)
    padding = bytes([padding_len] * padding_len)
    return plaintext + padding
def unpad(plaintext):
    """
    Removes a PKCS#7 padding, returning the unpadded text and ensuring the
    padding was correct.
    """
    padding_len = plaintext[-1]
    assert padding_len > 0
    message, padding = plaintext[:-padding_len], plaintext[-padding_len:]
    assert all(p == padding_len for p in padding)
    return message

def split_blocks(message, block_size=16, require_padding=True):
        assert len(message) % block_size == 0 or not require_padding
        return [message[i:i+16] for i in range(0, len(message), block_size)]
```

- bytes2matrix: 16바이트 배열을 4x4 행렬로 변환합니다.
- matrix2bytes: 4x4 행렬을 16바이트 배열로 변환합니다.
- xor_bytes: 두 바이트 배열 간에 비트별 XOR을 수행합니다.

- inc_bytes: 바이트 배열을 1씩 증가시킵니다.
- pad: 일반 텍스트에 PKCS#7 패딩을 추가합니다.
- unpad: 일반 텍스트에서 PKCS#7 패딩을 제거합니다.
- split_blocks: 메시지를 지정된 크기의 블록으로 나눕니다.

9. AES 클래스

```
174  class AES:
175      """
176      Class for AES-128 encryption with CBC mode and PKCS#7.
177
178      This is a raw implementation of AES, without key stretching or IV
179      management. Unless you need that, please use `encrypt` and `decrypt`.
180      """
181      rounds_by_key_size = {16: 10, 24: 12, 32: 14}
182      def __init__(self, master_key):
183          """
184          Initializes the object with a given key.
185          """
186          assert len(master_key) in AES.rounds_by_key_size
187          self.n_rounds = AES.rounds_by_key_size[len(master_key)]
188          self._key_matrices = self._expand_key(master_key)
189
190      def _expand_key(self, master_key):
191          """
192          Expands and returns a list of key matrices for the given master_key.
193          """
194          # Initialize round keys with raw key material.
195          key_columns = bytes2matrix(master_key)
196          iteration_size = len(master_key) // 4
```

- __init__: 마스터 키로 AES 객체를 초기화하고 키 크기에 따라 라운드 수를 계산합니다.
- _expand_key: 마스터 키를 라운드 키 목록으로 확장합니다.
- encrypt_block: 16바이트 길이의 일반 텍스트 블록을 암호화합니다.
- decrypt_block: 16바이트 길이의 암호문 블록을 복호화합니다.
- encrypt_cbc, decrypt_cbc: 암호 블록 체이닝(CBC) 모드를 사용하여 암호화 및 복호화합니다.
- encrypt_pcbc, decrypt_pcbc: 전파 암호 블록 체이닝(PCBC) 모드를 사용하여 암호화 및 복호화합니다.
- encrypt_cfb, decrypt_cfb: 암호 피드백(CFB) 모드를 사용하여 암호화 및 복호화합니다.
- encrypt_ofb, decrypt_ofb: 출력 피드백(OFB) 모드를 사용하여 암호화 및 복호화합니다.
- encrypt_ctr, decrypt_ctr: 카운터(CTR) 모드를 사용하여 암호화 및 복호화합니다.

10. 키 파생 및 암호화/복호화 함수

```
465  def encrypt(key, plaintext, workload=100000):
466      """
467      Encrypts `plaintext` with `key` using AES-128, an HMAC to verify integrity,
468      and PBKDF2 to stretch the given key.
469
470      The exact algorithm is specified in the module docstring.
471      """
472      if isinstance(key, str):
473          key = key.encode('utf-8')
474      if isinstance(plaintext, str):
475          plaintext = plaintext.encode('utf-8')
476
477      salt = os.urandom(SALT_SIZE)
478      key, hmac_key, iv = get_key_iv(key, salt, workload)
479      ciphertext = AES(key).encrypt_cbc(plaintext, iv)
480      hmac = new_hmac(hmac_key, salt + ciphertext, 'sha256').digest()
481      assert len(hmac) == HMAC_SIZE
482
483      return hmac + salt + ciphertext
484
485
486  def decrypt(key, ciphertext, workload=100000):
487      """
488      Decrypts `ciphertext` with `key` using AES-128, an HMAC to verify integrity,
489      and PBKDF2 to stretch the given key.
490
491      The exact algorithm is specified in the module docstring.
492      """
493
494      assert len(ciphertext) % 16 == 0, "Ciphertext must be made of full 16-byte blocks."
495
496      assert len(ciphertext) >= 32, """
497      Ciphertext must be at least 32 bytes long (16 byte salt + 16 byte block). To
498      encrypt or decrypt single blocks use `AES(key).decrypt_block(ciphertext)`.
499      """
```

- get_key_iv: PBKDF2-HMAC를 사용하여 암호 및 솔트에서 AES 키, HMAC 키 및 초기화 벡터(IV)를 파생합니다.
- encrypt: AES-128을 CBC 모드에서 사용하여 일반 텍스트를 키로 암호화하고, 무결성을 확인하기 위해 HMAC를 추가하고, 키 스트레칭을 위해 PBKDF2를 사용합니다.
- decrypt: 암호문을 복호화하고 HMAC를 확인하며 동일한 키 파생 프로세스를 사용합니다.