2022암호경진대회 4번 문제

[문제] 암호지갑에 포함된 정보가 <wallet>과 같이 주어지고 Mnemonic Code 단어사전이 <wordlist>와 같이 주어졌을 때, 이를 활용하여 <wallet>에 포함된 공개키에 상응하는 개인키 값을 구하여 (개인키 값, 풀이과정 및 구현 소스코드)를 제출하시오.

1. 먼저, 하나의 mnemonic sentence에 몇 개의 단어들이 들어있는지 알아보겠습니다.

   ent = 128 (bits)

   checksum = ent / 32 = 4 (bits)

   ent + checksum = 128 + 4 = 132 (bits)

   mnemonic code = (ent + checksum) / 11 = 12

   => 따라서 15개의 단어들 중 12개의 mnemonic codes들이 선택되어야 합니다.

   => P(15, 12) 개의 경우의 수가 존재합니다.

문제에서 제시된 공개키는 “xpub661”로 시작하는데, 이는 이 공개키가 depth 0에 있음을 의미합니다.

주어진 공개키에 대응하는 개인키를 찾기 위해서는, 문제에 제시된 word list의 15개의 단어 중 12개로 만들 수 있는 모든 mnemonic sentence를 만들고, 그 각각의 sentence에서 개인키를 만든 후, 그 개인키에서 공개키를 만들어서, 문제에서 제시된 공개키와 그것이 동일한지 비교하는 것입니다.

2. test code

find_priv_key()

: 해당 코드의 wordlist를 인수로 받아서 모든 가능한 mnemonic sentence를 만들고, 그 각각을 BIP-0039 기준에 따라 seed로 변환합니다. 그 seed로 BIP-0032 기준에 따라 마스터 개인키와 체인 코드를 생성하고, 이들로 마스터 공개키를 생성합니다. 이번에 만든 공개키가 문제에서 주어진 공개키와 동일하면, 공개키를 만들 때 쓰인 개인키를 출력하고 프로그램을 종료합니다.

```
def find_priv_key(word_list, r=12, passphrase=''):
        global xPubKey
        salt = 'mnemonic' + passphrase
        every_mnemonic_sentence = permutation(word_list, r)
        i = 0
        for sentence in every_mnemonic_sentence:
                mnemonic_code = ' '.join(sentence)
        seed = hashlib.pbkdf2_hmac("sha512", mnemonic_code.encode('utf-8'), salt.encode('utf-8'), 2048) # seed 생성
        master_priv_key, chain_code = master_key(seed)
        master_pub_key = get_xpub_from_master(master_priv_key, chain_code)
        if master_pub_key == xPubKey:
        print(f"private key found: {master_priv_key}")
        break
        print(f"{i} checked")
        i += 1
```

a. every_mnemonic_sentence = permutation(word_list, r)

: 주어진 리스트의 모든 원소들에 대해서 r 개수만큼의 모든 permutation을 반환합니다. find_priv_key()에서 인자로 받은 r값(r=12)을 통해서 P(15,12)의 경우의 수를 모두 만들기 위한 함수입니다.

```
def permutation(word_list, r):
```

```
        wordlist_values = [] # wordlist 를 wordlist_values 라는 이름의 | ist 로 저장
        for v in word_list:
                wordlist_values.append(v)
        result = itertools.permutations(wordlist_values, r) # P(15,12)
        return result
```

b. seed = hashlib.pbkdf2_hmac("sha512", mnemonic_code.encode('utf-8'), salt.encode('utf-8'), 2048)

: BIP-0039 기준에 따라, mnemonic code와 salt 값으로 hmac-sha512 함수를 통해서 seed 값을 반환합니다.

c. master_priv_key, chain_code = master_key(seed)

: BIP-0032 기준에 따라, seed값을 hmac_sha512로 해싱을 합니다. 해싱한 값은 앞의 256비트는 마스터 개인키로, 뒤의 256비트는 체인 코드로 사용됩니다.

```
def master_key(bip39_seed, testnet=False):
        I = hmac_sha512(key=b"Bitcoin seed", msg=bip39_seed) # seed 값 해싱
        i_left = I[:32]  # 256 나누기 8 을 하였을 때 32 가 나오기 때문에 32 를 사용
        int_left_key = big_endian_to_int(i_left)
        if int_left_key == 0:
            raise InvalidKeyError("master key is zero") # 유효하지 않은 key 일 때 예외 발생
        if int_left_key >= CURVE_ORDER: # CURVE_ORDER = CURVE_GENorder()
            raise InvalidKeyError("master key {} is greater/equal to curve order".format(int_left_key))
# 유효하지 않은 key 일 때 예외 발생
```

```
        # chain code
        i_right = I[32:]
        return i_left, i_right  # KEY, chaincode
```

c-1. I = hmac_sha512(key=b"Bitcoin seed", msg=bip39_seed)

: 생성된 seed값을 HMAC-SHA512로 해싱합니다.

```
def hmac_sha512(key: bytes, msg: bytes):
        return hmac.new(key = key, msg = msg, digestmod = hashlib.sha512).digest()
```

c-2. int_left_key = big_endian_to_int(i_left)

: 주어진 바이트 배열에 대한 정수를 반환합니다.

```
def big_endian_to_int(b: bytes) -> int:
        return int.from_bytes(b, "big")
```

c-3. raise InvalidKeyError("master key is zero")

: left_key(= master_priv_key)가 zero일 때 예외를 발생시킵니다.

c-4. raise InvalidKeyError("master key {} is greater/equal to curve order".format(int_left_key))

: left_key(= master_priv_key)가 CURVE_ORDER 일 때예외를 발생시킵니다.

```
class InvalidKeyError(Exception): # Exception 을 상속받아서 InvalidKeyError 라는 새로운 예외를 만듦
        def __init__(self, message):
                super().__init__(message)
```

d. master_pub_key = get_xpub_from_master(master_priv_key, chain_code)

: BIP-0032 기준에 따라, 마스터 개인키와 체인 코드를 이용해서 만든 마스터 공개키를 반환합니다.

```
def get_xpub_from_master(priv_key, chaincode):
        point = CURVE_GEN * int.from_bytes(priv_key, byteorder='big')
            # CURVE_GEN = ecdas.ecdas.generator_secp256k1
        if point.y() & 1:
        pub = b'₩x03'
        else:
        pub = b'₩x02'
          # compressed key
        pub += ecdsa.util.number_to_string(point.x(), CURVE_ORDER).rjust(32, b'₩x00')          #
xpub += ecdsa.util.number_to_string(point.y(), CURVE_ORDER).rjust(32, b'₩x00')


        xpub = b'₩x04₩x88₩xb2₩x1e'
        xpub += b'₩x00' * 9
        xpub += chaincode
        xpub += pub

        checksum = hashlib.sha256(hashlib.sha256(xpub).digest()).digest()[:4]
    return base58.b58encode(xpub + checksum)
```