

National Sun Yat-sen University  
Introduction To Blockchain Technology  
Homework 2  
Course Number: CSE222, Chapter: 3 & 4

# Notice :

1. No late homework.
2. Please submit your homework to **Cyber University of National Sun Yat-sen University** (<https://cu.nsysu.edu.tw/mooc/index.php>). It is not allowed to submit assignments to any other location.
3. You only need to submit **Homework 2.docx**, after you paste all screenshot of your code and code execution results of solving following problems.
4. We only accept using **python** to write program files.
5. **Please answer each question according to the requirements below, otherwise no points will be awarded.**

1. Verify whether the signature is valid:

P=(0x801be5a7c4faf73dd1c3f28cebf78d6ba7885ead88879b76ffb815d59056af14,0x826ddfcc38dafa6b8d463b609facc009083c8173e21c5fc45b3424964e85f49e)

Signature:

z=0x90d7aecf3f2855d60026f10faab852562c76e7e043cf243474ba5018447c2c22

r=0xf01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f

s=0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8

# Grading for Problem 1:

- Total: 20%
- Correct code and code execution result: 20%  
(Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code and code execution result of solving problem 1 below. ---

## Code:

FiniteField.py(包含 class FieldElement):

```
1  class FieldElement:
2      def __init__(self, num, prime):
3          if num >= prime or num < 0:
4              error = f"Num {num} not in field range 0 to {prime-1}"
5              raise ValueError(error)
6          self.num = num
7          self.prime = prime
8
9      def __add__(self, other):
10         if self.prime != other.prime:
11             raise TypeError('Cannot add two numbers in different Fields')
12         num = (self.num + other.num) % self.prime
13         return self.__class__(num, self.prime)
14
15     def __sub__(self, other):
16         if self.prime != other.prime:
17             raise TypeError('Cannot subtract two numbers in different Fields')
18         num = (self.num - other.num) % self.prime
19         return self.__class__(num, self.prime)
20
21     def __mul__(self, other):
22         if self.prime != other.prime:
23             raise TypeError('Cannot multiply two numbers in different Fields')
24         num = (self.num * other.num) % self.prime
25         return self.__class__(num, self.prime)
26
27     def __truediv__(self, other):
28         if self.prime != other.prime:
29             raise TypeError('Cannot divide two numbers in different Fields')
30         num = self.num * pow(other.num, self.prime - 2, self.prime) % self.prime
31         return self.__class__(num, self.prime)
32
33     def __pow__(self, exponent):
34         n = exponent % (self.prime - 1)
35         num = pow(self.num, n, self.prime)
36         return self.__class__(num, self.prime)
37
38     def __rmul__(self, coefficient):
39         return self.__class__((self.num * coefficient) % self.prime, self.prime)
40
41
42     def __str__(self):
43         return str(self.num)+ '(' + str(self.prime) + ')'
44
45     def __eq__(self, other):
46         return self.num == other.num and self.prime == other.prime
47
48     def __ne__(self, other):
49         return not (self == other)
```

EllipticCurves.py(包含 class Point):

```
1  from FiniteField import *
2
3  class Point:
4      def __init__(self, x, y, a, b):
5          self.a = a
6          self.b = b
7          self.x = x
8          self.y = y
9          if self.x is None and self.y is None:
10             return
11          if self.y**2 != self.x**3 + a*x + b:
12             raise ValueError(f'({x}, {y}) is not on the curve')
13
14      def __eq__(self, other):
15          return self.x == other.x and self.y == other.y and self.a == other.a and self.b == other.b
16
17      def __ne__(self, other):
18          return not (self == other)
19
20      def __str__(self):
21          if self.x is None:
22              return "Point(infinity)"
23          else:
24              if type(self.x) == int:
25                  return f"Point({self.x}, {self.y})_{self.a}_{self.b}"
26              else:
27                  return f"Point({self.x.num}, {self.y.num})_{self.a.num}_{self.b.num} FieldElement({self.x.prime})"
28
29      def __add__(self, other):
30          if self.a != other.a or self.b != other.b:
31              raise TypeError(f"Points {self} and {other} are not on the same curve")
32
33          if self.x is None:                                # 0 + P_other = P_other
34              return other
35          if other.x is None:                                # P_self + 0 = P_self
36              return self
37          if self.x == other.x and self.y != other.y:        # P + (-P) = 0
38              return self.__class__(None, None, self.a, self.b)
39          if self.x != other.x:                                # P1 != P2
40              s = (other.y - self.y) / (other.x - self.x)
41              x = s**2 - self.x - other.x
42              y = s * (self.x - x) - self.y
43              return self.__class__(x, y, self.a, self.b)
44          if self == other:                                    # P1 == P2
45              if self.y == 0 * self.x:
46                  return self.__class__(None, None, self.a, self.b)
47              s = (3 * self.x**2 + self.a) / (2 * self.y)
48              x = s**2 - 2 * self.x
49              y = s * (self.x - x) - self.y
50              return self.__class__(x, y, self.a, self.b)
51
52      def __rmul__(self, coefficient):
53          result = self.__class__(None, None, self.a, self.b)
54          temp_coefficient = coefficient
55          currDigit = self
56          while temp_coefficient > 0:
57              if temp_coefficient & 1:
58                  result += currDigit
59                  currDigit += currDigit
60                  temp_coefficient >>= 1
61          return result
```

Verify.py(包含 class S256Field, S256Point, Signature, PrivateKey):

```
1  from FiniteField import *
2  from EllipticCurves import *
3
4  gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
5  gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
6  p = 2**256 - 2**32 - 977
7  N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
8  A = 0
9  B = 7
10
11 class S256Field(FieldElement):
12     def __init__(self, num, prime=None):
13         super().__init__(num = num, prime = p)
14
15 class S256Point(Point):
16
17     def __init__(self, x, y, a=None, b=None):
18         a, b = S256Field(A), S256Field(B)
19         if type(x) == int:
20             super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
21         else: # infinity point
22             super().__init__(x = x, y = y, a = a, b = b)
23
24     def __rmul__(self, coefficient):
25         coefficient = coefficient % N
26         return super().__rmul__(coefficient)
27
28     def verify(self, z, sig):
29         s_inv = pow(sig.s, N-2, N)
30         u = (z * s_inv) % N
31         v = (sig.r * s_inv) % N
32         kG = u * G + v * self
33         return kG.x.num == sig.r
34
35 G = S256Point(gx, gy)
36
37
38 class Signature:
39     def __init__(self, r, s):
40         self.r = r
41         self.s = s
42
43 class PrivateKey:
44     def __init__(self, secret):
45         self.secret = secret
46         self.point = secret * G
47
```

```

48     def sign(self, z, k):
49         r = (k * G).x.num
50         k_inv = pow(k, N-2, N)
51         s = (k_inv * (z + self.secret * r)) % N
52         if s > N / 2:
53             s = N - s
54         return Signature(r, s)
55
56
57 if __name__ == '__main__':
58     P = S256Point(0x801be5a7c4faf73dd1c3f28cebf78d6ba7885ead88879b76ffb815d59056af14,
59                  0x826ddfcc38daf6b8d463b609facc009083c8173e21c5fc45b3424964e85f49e)
60     z = 0x90d7aecf3f2855d60026f10faab852562c76e7e043cf243474ba5018447c2c22
61     r = 0xf01d6b9018ab421dd410404cb869072065522bf85734008f105cf385a023a80f
62     s = 0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8
63
64     sig = Signature(r, s)
65
66     if P.verify(z, sig):
67         print("Signature is valid")
68     else:
69         print("Signature is invalid")

```

**Result:**

```

C:\python_file\區塊鏈導論\HW2>python Verify.py
Signature is valid

```

2. Sign the following message with the secret:

Private key  $e=1234567$

$z = \text{int.from\_bytes}(\text{hash256}(\text{b}'\text{Introduction to Bitcoin homework 2.2}'), \text{'big'})$

$k = 1234567$

Print the point,  $z$ ,  $r$  and  $s$  in hexadecimal.

# Grading for Problem 2:

- Total: 20%

- Correct code and code execution result: 20%

(Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code and code execution result of solving problem 2 below. ---

## Code:

Sign.py()(包含 class S256Field, S256Point, Signature, PrivateKey , 及 method hash256):

**Note:** 四個 class 同 problem1

```
1  from FiniteField import *
2  from EllipticCurves import *
3  import hashlib
4
5  gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
6  gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
7  p = 2**256 - 2**32 - 977
8  N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
9  A = 0
10 B = 7
11
12 class S256Field(FieldElement):
13     def __init__(self, num, prime=None):
14         super().__init__(num = num, prime = p)
15
16 class S256Point(Point):
17
18     def __init__(self, x, y, a=None, b=None):
19         a, b = S256Field(A), S256Field(B)
20         if type(x) == int:
21             super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
22         else: # infinity point
23             super().__init__(x = x, y = y, a = a, b = b)
24
25     def __rmul__(self, coefficient):
26         coefficient = coefficient % N
27         return super().__rmul__(coefficient)
28
29     def verify(self, z, sig):
30         s_inv = pow(sig.s, N-2, N)
31         u = (z * s_inv) % N
32         v = (sig.r * s_inv) % N
33         kG = u * G + v * self
34         return kG.x.num == sig.r
35
36 G = S256Point(gx, gy)
37
38
39 class Signature:
40     def __init__(self, r, s):
41         self.r = r
42         self.s = s
43
44 class PrivateKey:
45     def __init__(self, secret):
46         self.secret = secret
47         self.point = secret * G
48
```

```

49     def sign(self, z, k):
50         r = (k * G).x.num
51         k_inv = pow(k, N-2, N)
52         s = (k_inv * (z + self.secret * r)) % N
53         if s > N / 2:
54             s = N - s
55         return Signature(r, s)
56
57     def hash256(s):
58         return hashlib.sha256(hashlib.sha256(s).digest()).digest()
59
60     if __name__ == '__main__':
61         e = 1234567
62         z = int.from_bytes(hash256(b'Introduction to Bitcoin homework 2.2'), 'big')
63         k = 1234567
64
65         private_key = PrivateKey(e)
66         sig = private_key.sign(z, k)
67         print("Point: ", private_key.point)
68         print("r: ", hex(sig.r))
69         print("s: ", hex(sig.s))
70

```

## Result:

```

C:\python_file\區塊鏈導論\HW2>python Sign.py
Point: Point(58816500655650144487794134876851742874492254758156806771173176308747531417647, 8305713
7326221658054703598734219124683682001483192222523127688683857461171418)_0_7 FieldElement(11579208923
7316195423570985008687907853269984665640564039457584007908834671663)
r: 0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f
s: 0x478298f333e732cc80b383708bde0d90d9785d25107e349c570b04337a02681b

```

## 3. Solve these problems:

3-1. Find the uncompressed SEC format for the public key where the private key secrets is: 23396049

3-2. Find the compressed SEC format for the public key where the private key secrets is: 23396050

3-3. Find the DER format for a signature whose r and s values are:

r =

0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f

s =

0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8

# Grading for Problem 3:

- Total: 30%
- Correct code and code execution result: 30%, 10% for each  
(Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code and code execution result of solving problem 3-1 below. ---

### Code:

serialization.py(包含 class S256Field:新增 sqrt(), S256Point:新增 sec(), Signature: 新增 DER(), PrivateKey, 及 method hash256(), parse(), ):

```

1  from FiniteField import *
2  from EllipticCurves import *
3  import hashlib
4
5  gx = 0x79be667ef9dcbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798
6  gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
7  p = 2**256 - 2**32 - 977
8  N = 0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
9  A = 0
10 B = 7
11
12 class S256Field(FieldElement):
13     def __init__(self, num, prime=None):
14         super().__init__(num = num, prime = p)
15     def sqrt(self):
16         return self**((p + 1) // 4)
17
18 class S256Point(Point):
19
20     def __init__(self, x, y, a=None, b=None):
21         a, b = S256Field(A), S256Field(B)
22         if type(x) == int:
23             super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
24         else: # infinity point
25             super().__init__(x = x, y = y, a = a, b = b)
26
27     def __rmul__(self, coefficient):
28         coefficient = coefficient % N
29         return super().__rmul__(coefficient)
30
31     def verify(self, z, sig):
32         s_inv = pow(sig.s, N-2, N)
33         u = (z * s_inv) % N
34         v = (sig.r * s_inv) % N
35         kG = u * G + v * self
36         return kG.x.num == sig.r
37
38     def sec(self, compressed = True):
39         if compressed:
40             if self.y.num % 2 == 0:
41                 return b'\x02' + self.x.num.to_bytes(32, 'big')
42             else:
43                 return b'\x03' + self.x.num.to_bytes(32, 'big')
44         else:
45             return b'\x04' + self.x.num.to_bytes(32, 'big') + self.y.num.to_bytes(32, 'big')
46

```



```

47     @classmethod
48     def parse(self, sec_bin):
49         if sec_bin[0] == 4:
50             x = int.from_bytes(sec_bin[1:33], 'big')
51             y = int.from_bytes(sec_bin[33:65], 'big')
52             return S256Point(x, y)
53         is_even = sec_bin[0] == 2
54         x = int.from_bytes(sec_bin[1:], 'big')
55         alpha = (x**3 + S256Field(B))
56         beta = alpha.sqrt()
57         if beta.num & 2 == 0:
58             even_beta = beta
59             odd_beta = S256Field(p - beta.num)
60         else:
61             odd_beta = beta
62             even_beta = S256Field(p - beta.num)
63         if is_even:
64             return S256Point(x, even_beta)
65         else:
66             return S256Point(x, odd_beta)
67
68     G = S256Point(gx, gy)
69
70
71     class Signature:
72         def __init__(self, r, s):
73             self.r = r
74             self.s = s
75
76         def DER(self):
77             r_bin = self.r.to_bytes(32, byteorder = 'big')
78             r_bin = r_bin.lstrip(b'\x00')
79             if r_bin[0] & 0x80:
80                 r_bin = b'\x00' + r_bin
81             result = bytes([2, len(r_bin)]) + r_bin
82
83             s_bin = self.s.to_bytes(32, byteorder = 'big')
84             s_bin = s_bin.lstrip(b'\x00')
85             if s_bin[0] & 0x80:
86                 s_bin = b'\x00' + s_bin
87             result += bytes([2, len(s_bin)]) + s_bin
88             return bytes([0x30, len(result)]) + result

```

```

89
90 v class PrivateKey:
91 v     def __init__(self, secret):
92         self.secret = secret
93         self.point = secret * G
94
95 v     def sign(self, z, k):
96         r = (k * G).x.num
97         k_inv = pow(k, N-2, N)
98         s = (k_inv * (z + self.secret * r)) % N
99 v         if s > N / 2:
100             s = N - s
101         return Signature(r, s)
102
103 v def hash256(s):
104     return hashlib.sha256(hashlib.sha256(s).digest()).digest()
105
106
107 v if __name__ == '__main__':
108     ### 3-1, 3-2
109     e1 = 23396049
110     e2 = 23396050
111     private_key1 = PrivateKey(e1)
112     private_key2 = PrivateKey(e2)
113     print("uncompressed SEC format (e = 23396049): ", private_key1.point.sec(compressed = False))
114     print("compressed SEC format (e = 23396050): ", private_key2.point.sec())
115
116     ### 3-3
117     r = 0x8208f5abf04066bad1db9d46f8bcf5a6cc11d0558ab523e7bd3c0ec08bdb782f
118     s = 0x22afcd685b7c0c8b525c2a52529423fcdff22f69f3e9c175ac9cb3ec08de87d8
119     sig = Signature(r, s)
120     print("DER format: ", sig.DER())

```

## Result:

```

C:\python_file\區塊鏈導論\HW2>python serialization.py
uncompressed SEC format (e = 23396049): b'\x04\xfb\x2a\x6t0\x1b\x94^$\xa5D.2PF\xfa0{(+\x10\x85\x93
'\x8e?\x82X\x15\x07\xdc\xdc\xea\xfb\xec\x04oH\xdfj'\xc0 \xc2\xd7L\x17n\xb7\xea\x91\xa6\xcf\xcl\xbe<\x
de\x01\x86\xb8\xe9o\xa5\x93\xf0'

```

--- Please paste screenshot of your code and code execution result of solving problem 3-2 below. ---

## Code:

同 problem 3-1

## Result:

```

compressed SEC format (e = 23396050): b'\x03\xea[\x9b\x9f\xf4c\x96\x0b{\x1de\xc5zHm\xe3\xc5\x9dCD\
c1L\xdc\r~?\tgY\xe26\xf'

```

--- Please paste screenshot of your code and code execution result of solving problem 3-3 below. ---

## Code:

同 problem 3-1

## Result:

```
DER format: b'0E\x02!\x00\x82\x08\xf5\xab\xf0@\f\xba\xd1\xdb\x9dF\xf8\xbc\xf5\xa6\xcc\x11\xd0U\x8a\x
b5#\xe7\xbd<\x0e\xc0\x8b\xdb\x/\x02 "\xaf\xcdh[\x0c\x8bR\*\RR\x94#\xfc\xdf\xf2/i\xf3\xe9\xcl\xac\x9
c\xb3\xec\x08\xde\x87\xd8'
```

4. Compute the slope and the sum of the points:

4-1. Find the address corresponding to Public Keys whose Private Key secrets are:

23396051 (use uncompressed SEC, on testnet) (10%)

23396052 (use compressed SEC, on testnet) (10%)

4-2. Find the WIF for Private Key whose the secret is:

23396053 (use compressed SEC, on testnet) (10%)

# Grading for Problem 4:

- Total: 30%

- Correct code and code execution result: 30%, 10% for each

(Only have correct code and correct code execution can get score.)

--- Please paste screenshot of your code and code execution result of solving problem 4-1 below. ---

## Code:

Address\_and\_WIF.py(包含 S256Field, S256Point: 新增 RIPEMD160\_SHA256()及 address(), Signature, PrivateKey: 新增 WIF(), 及 method hash256, parse, encode\_base58(), RIPEMD160\_SHA256()):

```
1  from FiniteField import *
2  from EllipticCurves import *
3  import hashlib
4
5  gx = 0x79be667ef9dcbbac55a06295ce870b07029bfcd2dce28d959f2815b16f81798
6  gy = 0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8
7  p = 2**256 - 2**32 - 977
8  N = 0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141
9  A = 0
10 B = 7
11
12 class S256Field(FieldElement):
13     def __init__(self, num, prime=None):
14         super().__init__(num = num, prime = p)
15     def sqrt(self):
16         return self**((p + 1) // 4)
17
18 class S256Point(Point):
19
20     def __init__(self, x, y, a=None, b=None):
21         a, b = S256Field(A), S256Field(B)
22         if type(x) == int:
23             super().__init__(x = S256Field(x), y = S256Field(y), a = a, b = b)
24         else: # infinity point
25             super().__init__(x = x, y = y, a = a, b = b)
26
27     def __rmul__(self, coefficient):
28         coefficient = coefficient % N
29         return super().__rmul__(coefficient)
30
31     def verify(self, z, sig):
32         s_inv = pow(sig.s, N-2, N)
33         u = (z * s_inv) % N
34         v = (sig.r * s_inv) % N
35         kG = u * G + v * self
36         return kG.x.num == sig.r
37
38     def sec(self, compressed = True):
39         if compressed:
40             if self.y.num % 2 == 0:
41                 return b'\x02' + self.x.num.to_bytes(32, 'big')
42             else:
43                 return b'\x03' + self.x.num.to_bytes(32, 'big')
44         else:
45             return b'\x04' + self.x.num.to_bytes(32, 'big') + self.y.num.to_bytes(32, 'big')
```

```

46     @classmethod
47     def parse(self, sec_bin):
48         if sec_bin[0] == 4:
49             x = int.from_bytes(sec_bin[1:33], 'big')
50             y = int.from_bytes(sec_bin[33:65], 'big')
51             return S256Point(x, y)
52         is_even = sec_bin[0] == 2
53         x = int.from_bytes(sec_bin[1:], 'big')
54         alpha = (x**3 + S256Field(B))
55         beta = alpha.sqrt()
56         if beta.num & 2 == 0:
57             even_beta = beta
58             odd_beta = S256Field(p - beta.num)
59         else:
60             odd_beta = beta
61             even_beta = S256Field(p - beta.num)
62         if is_even:
63             return S256Point(x, even_beta)
64         else:
65             return S256Point(x, odd_beta)
66
67     def RIPEMD160_SHA256(self, compressed = True):
68         return RIPEMD160_SHA256(self.sec(compressed))
69
70     def address(self, compressed = True, testnet = False):
71         h160 = self.RIPEMD160_SHA256(compressed)
72         if testnet:
73             prefix = b'\x6f'
74         else:
75             prefix = b'\x00'
76         return encode_base58(prefix + h160 + hash256(prefix + h160)[0:4])
77         # (version + hashed public key + checksum) encoded in base58
78
79     G = S256Point(gx, gy)
80
81
82     class Signature:
83     def __init__(self, r, s):
84         self.r = r
85         self.s = s

```

```

87     def DER(self):
88         r_bin = self.r.to_bytes(32, byteorder = 'big')
89         r_bin = r_bin.lstrip(b'\x00')
90         if r_bin[0] & 0x80:
91             r_bin = b'\x00' + r_bin
92         result = bytes([2, len(r_bin)]) + r_bin
93
94         s_bin = self.s.to_bytes(32, byteorder = 'big')
95         s_bin = s_bin.lstrip(b'\x00')
96         if s_bin[0] & 0x80:
97             s_bin = b'\x00' + s_bin
98         result += bytes([2, len(s_bin)]) + s_bin
99         return bytes([0x30, len(result)]) + result
100
101
102     class PrivateKey:
103         def __init__(self, secret):
104             self.secret = secret
105             self.point = secret * G
106
107         def sign(self, z, k):
108             r = (k * G).x.num
109             k_inv = pow(k, N-2, N)
110             s = (k_inv * (z + self.secret * r)) % N
111             if s > N / 2:
112                 s = N - s
113             return Signature(r, s)
114
115         def WIF(self, compressed = True, testnet = False):
116             secret_bytes = self.secret.to_bytes(32, 'big')
117             if testnet:
118                 prefix = b'\xef'
119             else:
120                 prefix = b'\x80'
121             if compressed:
122                 suffix = b'\x01'
123             else:
124                 suffix = b''
125
126             return encode_base58(prefix + secret_bytes + suffix + hash256(prefix + secret_bytes + suffix)[0:4])
127             # (prefix + secret + suffix + checksum) encoded in base58
128
129         def hash256(s):
130             return hashlib.sha256(hashlib.sha256(s).digest()).digest()

```

```

131
132 def encode_base58(s):
133     BASE58_AKPHABET = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'
134
135     count = 0
136     for c in s:
137         if c == 0:
138             count += 1
139         else:
140             break
141
142     num = int.from_bytes(s, 'big')
143     prefix = '1' * count
144     result = ''
145
146     while num > 0:
147         num, mod = divmod(num, 58)
148         result = BASE58_AKPHABET[mod] + result
149     return prefix + result
150
151 def RIPEMD160_SHA256(s):
152     return hashlib.new('ripemd160', hashlib.sha256(s).digest()).digest()
153
154
155 if __name__ == '__main__':
156     ### 4-1
157     e1 = 23396051
158     e2 = 23396052
159     private_key1 = PrivateKey(e1)
160     private_key2 = PrivateKey(e2)
161     print("address (e=23396051, uncompressed SEC, testnet): ", private_key1.point.address(compressed = False, testnet = True))
162     print("address (e=23396052, compressed SEC, testnet): ", private_key2.point.address(compressed = True, testnet = True))
163
164     ### 4-2
165     e3 = 23396053
166     private_key3 = PrivateKey(e3)
167     print("WIF (e=23396053, compressed SEC, testnet): ", private_key3.WIF(compressed = True, testnet = True))
168

```

## Result:

```

C:\python_file\區塊鏈導論\HW2>python Address_and_WIF.py
address (e=23396051, uncompressed SEC, testnet):  mrmyTYBRrqajL6bbfjdPkVPF6uSzBLRT9r
address (e=23396052, compressed SEC, testnet):  mqs96pmCwhKpBq64mShWxMeyDhYnrC4hUi
WIF (e=23396053, compressed SEC, testnet):  cMahea7zqjxrtgAbB7LSGbcQUrluX1ojuat9jZodNPqQsC5vocuj

```

--- Please paste screenshot of your code and code execution result of solving problem 4-2 below. ---

## Code:

同 problem 4-1

## Result:

```

WIF (e=23396053, compressed SEC, testnet):  cMahea7zqjxrtgAbB7LSGbcQUrluX1ojuat9jZodNPqQsC5vocuj

```