Secret Sharing report

Secret sharing works by splitting confidential information into smaller pieces or shares and then distributing those shares amongst a group or network. In secret sharing, each individual share is useless on its own(In other word, No individual holds any intelligible information about the secret), but when all the shares are together, they reconstruct an original secret.

The secret sharing is there is 1 dealer and n players. The dealer distributes shares of the secret to the players, but players can only reconstruct the secret from their shares if certain conditions are met. The dealer achieves this by giving each player a share, such that any group of t (for a threshold) or more players can reconstruct the secret together, but fewer than t players cannot. Such a system is called a (t, n)-threshold scheme. For ease of understanding, let's take an example. Imagine that you are the boss of the bank. You need to keep one million dollars in the bank account so that make bank run successfully and in order to access this bank account you used the password: secret. Besides, you need to let shareholders share this password so you should split the password up and distribute a message to six shareholders. All of them have one of the different letters that together can consist of the real password. The only information that each shareholder would have is the letter that they hold, making their individual shares useless. By the way, the secret sharing scheme can be distributed shares based on how much the shareholders are trusted.

But what about when there is zero trust between secret owners and shareholders? The privacy and security of secret sharing must be mentioned. Secret sharing schemes are ideal for storing highly sensitive and very important information. On the face of it, each shareholder only holds seemingly random numbers. When all the shareholders put all the letters together, they still can't get the real code. Secret sharing greatly solves this problem. Improving the reliability of keys by storing multiple copies reduces confidentiality by creating additional attack vectors; secret sharing also allows the distributor of the secret to trusting a group "in general". Giving a secret to a group for safekeeping requires the distributor to fully trust all members of the group. A secret sharing scheme allows a distributor to securely store a secret with the group, even if not all members can be trusted all the time. As long as the number of traitors does not exceed the critical number needed to reconstruct the secret, the secret is safe. The security in such environments can be made greater by continuous changing in the way the shares are constructed.

The secret has many uses and applications. A secret sharing scheme can secure a secret over multiple servers and remain recoverable despite multiple server failures. The dealer may act as several distinct participants, distributing the shares among the participants. Each share may be stored on a different server, but the dealer can recover the secret even if several servers break down as long as they can recover at least t shares;

There are three kinds of efficient secret sharing, namely Shamir scheme, Blakley scheme and Using the Chinese remainder theorem. We focus on the Shamir scheme. Our code used the Shamir scheme as well. One of the benefits of Shamir's algorithm is that it is flexible and extensible. This means that the secret owner can add, modify, or remove shares at any time they want without modifying the original secret.

Shamir's secret sharing scheme is an algorithm first proposed by the famous Israeli cryptographer Adi Shamir in 1979. It allows information to be split into many parts, while only a fraction of those parts is needed to reconstruct the original secret. This means that, instead of requiring all shares to reconstruct the original secret, Shamir’s scheme requires a minimum number of shares. As we mentioned before, this minimum is referred to as the threshold.

This is my group's main code and there is some code explanation on it.

```
import random

UPPER_LIMIT = 10^6

def create_shares(T, N, secret):
    if T > N:
        raise ValueError("invalid argument. The value of T must be larger than the value of N")

    """ Firstly, we need to create a random polynomial of degree N-1 """
    coefficients = [random.randrange(UPPER_LIMIT) for _ in range(T-1)]
    coefficients.append(secret)

    """ Then we construct a polynomial according to the coefficients """
    x = QQ['x'].0
    f = 0
    exp = len(coefficients) - 1
    for i in coefficients:
        f += x^exp*i
        exp -= 1

    """ Then we generate N random points on the polynomial """
    shares = []
    for _ in range(N):
        x = random.randrange(UPPER_LIMIT)
        shares.append((x, f(x)))
    return shares

def decrypt_secret(shares):
    """ The function calculates the secret value based on the shares given """
    R = PolynomialRing(QQ, 'x')
    f = R.lagrange_polynomial(shares)
    # print(f)
    return f(0) # The constant term of the generated polynomial is the secret we would like to compute
```

Explanations of the written code:

We define two functions to achieve the aim.
create_shares function takes T, N, S(secret) as parameter:
Firstly, we need to create a random polynomial of degree N-1
Secondly, Then we construct a polynomial according to the coefficients
Then, We generate N random points on the polynomial

decrypt_secret function takes shares as parameter:
The function calculates the secret value based on the shares given

The approach is find a polynomial equation with the degree (K – 1) for the given K points.

Let’s say S is the secret that we wish to encode. It is divided into N parts: S1, S2, S3, …., Sn. After dividing it, the number K is chosen by the user in order to decrypt the parts and find the original secret. It is chosen in such a way that if we know less than K parts, then we will not be able to find the secret S (i.e.) the secret S can not be reconstructed with (K – 1) parts or fewer. If we know K or more parts from S1, S2, S3, …., and Sn, then we can compute/reconstruct our secret code S easily. This is conventionally called the (K, N) threshold scheme.

The idea is to build a polynomial with the degree (K – 1) such that the constant term is the secret code and the remaining numbers are random and

this constant term can be found by using any K points out of N points generated from this polynomial by using Lagrange's Basis Polynomial.
Firstly, we build a polynomial of degree (K – 1) in order to encrypt the secret code.
Secondly, for the given two points, (x1, y1) and (x2, y2) we can find a linear polynomial ax + by = c. Let the polynomial be y = a + bx and the constant part 'a' is our secret code and let b be any random number.
Finally, we generate N points and we can generate the initial polynomial from any two of these N points in the resulting polynomial, the constant term a is the required secret code. When reconstructing the given polynomial back, we use Lagrange basis Polynomial. We mainly use the Lagrange's identities to calculate them and the summation of these identities gives us the required function which we need to find from the given points.

So above the thinking of making an approach, implement the code:

```
secret = int(input("Please enter the secret number you would like to have: "))
N = int(input("Please enter the number of shares you would like to have: "))
T = int(input("Please enter the number of shares required to get the secret: "))

shares = create_shares(T, N, secret)
print("The shares generated are: ")
print(shares)

random_shares = random.sample(shares, T)
answer = decrypt_secret(random_shares)
print(f"After combining the shares, the secret is: {answer}")
```

```
Please enter the secret number you would like to have: 12
Please enter the number of shares you would like to have: 5
Please enter the number of shares required to get the secret: 4
The shares generated are:
[(291559, 22618426630555228978238), (222189, 10010387908066492604538), (276965, 1938910413746004070724
2), (117198, 1469077238606408109984), (
810372344748524291064 60)]
After combining the shares, the secret is: 12
```

Test1: Tests in this case mean to generate secret values that are equal to the original secret value

```
def normal_test(T, N ,secret):
    """ Tests in this case mean to generate secret values that are equal to the original secret value """
    shares = create_shares(T, N, secret)

    random_shares = random.sample(shares, T)
    decrypted_secret = decrypt_secret(random_shares)

    return secret == decrypted_secret

print("##### TEST WITH NORMAL DATA #####")

if normal_test(1, 2, 15):
    print("Test passed")
else:
    print("Test failed")

if normal_test(3, 5, 18):
    print("Test passed")
else:
    print("Test failed")

if normal_test(5, 5, 123):
    print("Test passed")
else:
    print("Test failed")

if normal_test(5, 6, 1556):
    print("Test passed")
else:
    print("Test failed")

if normal_test(5, 7, 10000):
    print("Test passed")
else:
    print("Test failed")
```

```
##### TEST WITH NORMAL DATA #####
15
Test passed
263648*x^2 + 386507*x + 18
Test passed
865036*x^4 + 132680*x^3 + 577792*x^2 + 489016*x + 123
Test passed
87979*x^4 + 820470*x^3 + 521655*x^2 + 448024*x + 1556
Test passed
900206*x^4 + 232599*x^3 + 410607*x^2 + 212321*x + 10000
Test passed
```

Test2: Tests in this case mean to generate a secret value which is different from the original secret

```
def failed_test(T, N, secret):
    """ Tests in this case mean to generate a secret value which is different from the orginal secret """
    shares = create_shares(T, N, secret)

    random_shares = random.sample(shares, random.randrange(1, T-1)) # Generate m shares where m < t, meaning that the secret cannot k
    decrypted_secret = decrypt_secret(random_shares)

    return secret != decrypted_secret

print("##### TEST WITH FAILED DATA #####")

if failed_test(3, 5, 12):
    print("Test passed")
else:
    print("Test failed")

if failed_test(3, 6, 18):
    print("Test passed")
else:
    print("Test failed")

if failed_test(5, 5, 123):
    print("Test passed")
else:
    print("Test failed")

if failed_test(5, 6, 122):
    print("Test passed")
else:
    print("Test failed")

if failed_test(6, 8, 12345):
    print("Test passed")
else:
    print("Test failed")
```

```
##### TEST WITH FAILED DATA #####
Test passed
Test passed
Test passed
Test passed
Test passed
```

Test3: All tests in this case mean to raise error during the execution of the program

```
def error_test(T, N, secret):
    """All tests in this case mean to raise error during the execution of the program """
    shares = create_shares(T, N, secret)

    random_shares = random.sample(shares, T)
    decrypted_secret = decrypt_secret(random_shares)

    return secret == decrypted_secret

try:
    error_test(6, 5, 12)
    print("Test failed")
except ValueError as e:
    print("Test passed")

try:
    error_test(7, 3, 1232)
    print("Test failed")
except ValueError as e:
    print("Test passed")

try:
    error_test(9, 4, 12132)
    print("Test failed")
except ValueError as e:
    print("Test passed")
```

```
Test passed
Test passed
Test passed
```