پروژه درس رمزنگاری و امنیت شبکه دکتر ملا

مریم امیرشاه کرمی
4003613008

Shamir's Secret Sharing Scheme:

یک الگوریتم رمزنگاری است که برای تقسیم یک راز به چندین بخش یا سهام طراحی شده است، به طوری که برای بازسازی راز اصلی فقط به تعداد آستانه مشخصی از سهام نیاز است. این طرح توسط Adi Shamir در سال 1979 پیشنهاد شد و به طور گسترده ای برای محاسبات امن چند جانبه و مدیریت کلید در سیستم های توزیع شده استفاده می شود.

Secret S:

اطلاعاتی که باید محافظت شوند: راز

Threshold t:

حداقل تعداد سهام مورد نیاز برای بازسازی راز.

Total number of shares n:

تعداد کل قسمت هایی که راز به آنها تقسیم می شود

Prime number ppp:

یک عدد اول بزرگتر از رمز و سایر ضرایب استفاده شده در چند جمله ای

Lagrange Interpolation

To reconstruct the secret from the shares, we use Lagrange interpolation. Given t shares $(x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t)$, the polynomial can be reconstructed as:

$$f(x) = \sum_{j=1}^{t} y_j \cdot \ell_j(x) \mod p$$

where $\ell_j(x)$ are the Lagrange basis polynomials, defined as:

$$\ell_j(x) = \prod_{1 \le m \le t, m \neq j} \frac{x - x_m}{x_j - x_m} \mod p$$

Evaluating the polynomial at $x = 0$ gives the secret:

$$S = f(0) = \sum_{j=1}^{t} y_j \cdot \ell_j(0) \mod p$$

```
import random
import numpy as np
```

- random: برای تولید اعداد تصادفی استفاده می‌شود.
-numpy: برای عملیات آرایه‌ها و توابع ریاضی مانند جذر استفاده می‌شود.

```
def is_prime(x: int):
    if x < 2:
        return False
    elif x == 2:
        return True
    else:
        for i in range(2, int(np.sqrt(x)), 2):
            if x % i == 0:
                return False
    return True
```

این تابع بررسی می‌کند که آیا عدد `x` اول است یا خیر:
- اگر `x` کمتر از ۲ باشد، `False` برمی‌گرداند.
- اگر `x` برابر ۲ باشد، `True` برمی‌گرداند.
- در غیر این صورت، بررسی می‌کند که آیا `x` بر عددی از ۲ تا جذر `x` بخش‌پذیر است یا خیر.

```
def share_secret():
    print("----SECRET SHARING----")
    n = 0
    t = 1
    while n < t:
```

```
    print("n= ", end="")
    n = int(input())
    print("t= ", end="")
    t = int(input())
    if n <= t:
        print("t should be less or equal than n")
S = 1
p = 0
while p < S:
    print("S= ", end="")
    S = int(input())
    while not is_prime(p):
        print("p= ", end="")
        p = int(input())
        if not is_prime(p):
            print("please enter a prime number for p")
    if S >= p:
        print("please choose a bigger p than S")
```

این بلوک ورودی‌های کاربر را برای تعداد سهام `n`، حداقل تعداد سهام `t`، راز `S` و یک عدد اول `p` که بزرگتر از `S` است جمع‌آوری می‌کند.

```
a = np.zeros(t)
a[0] = S
for i in range(1, t):
    a[i] = random.randint(-10, 10)

x = np.zeros(n)
for i in range(0, n):
    x[i] = i+1
```

```
    y = np.zeros(n)
    for i in range(0, n):
        tmp = 0
        for j in range(0, t):
            tmp += a[j] * pow(x[i], j)

        y[i] = tmp.__mod__(p)

    for i in range(0, n):
        print(f'({x[i]}, {y[i]})')
```

- `a` آرایه‌ای از ضرایب چندجمله‌ای است. راز `S` ضریب ثابت است و سایر ضرایب به صورت تصادفی انتخاب می‌شوند.
- مقادیر `x` از ۱ تا `n` هستند.
- مقادیر `y` با استفاده از چندجمله‌ای و سپس گرفتن باقی‌مانده بر `p` محاسبه می‌شوند.
- زوج‌های `(x, y)` نمایانگر سهام هستند.

```
def mod_inverse(a: int, p: int):
    for i in range(1, p):
        x = a*i
        if x.__mod__(p) == 1:
            return i
```

این تابع معکوس پیمانه‌ای `a` نسبت به `p` را محاسبه می‌کند.

```
def get_secret():
    print("----SECRET REVEALING----")
    print("t= ", end="")
    t = int(input())
    p = 0
```

```
    while not is_prime(p):
        print("p= ", end="")
        p = int(input())
        if not is_prime(p):
            print("please enter a prime number for p")

    y = np.zeros(t)
    x = np.zeros(t)
    for i in range(0, t):
        print(f'x[{i+1}]= ', end="")
        x[i] = int(input())
        print(f'y[{i+1}]= ', end="")
        y[i] = int(input())

    S = 0
    for i in range(0, t):
        tmp = 1
        for j in range(0, t):
            if i != j:
                tmp1 = x[j] - x[i]
                tmp1 = mod_inverse(tmp1, p)
                tmp2 = x[j] * tmp1
                tmp2 = tmp2.__mod__(p)
                tmp *= tmp2
        S += (y[i]*tmp).__mod__(p)

    print(f'S= {S.__mod__(p)}')
```

- ورودی‌های `t` و `p` و `t` زوج `(x, y)` جمع‌آوری می‌شوند.
- راز `S` با استفاده از تداخل لاگرانژ در میدان متناهی تعریف‌شده توسط `p` بازسازی می‌شود.

python

share_secret()
get_secret()

- تابع `share_secret` برای توزیع راز فراخوانی می‌شود.
- تابع `get_secret` برای بازسازی راز از سهام‌ها فراخوانی می‌شود.

نتیجه
این کد یک پیاده‌سازی ساده از طرح تقسیم راز شمر است. راز را به `n` سهام تقسیم می‌کند، به طوری که هر `t` سهام می‌توانند راز را بازسازی کنند، اما کمتر از `t` سهام هیچ اطلاعاتی درباره راز فاش نمی‌کنند. کد از تداخل چندجمله‌ای در میدان متناهی تعریف‌شده توسط یک عدد اول `p` استفاده می‌کند.