Реализация Схемы Блома распределения ключей

Роман Астраханцев, СКБ-171

1 февраля 2022 г.

1 Написание программы

Была написана программа на языке программирования Python, которая реализует схему разделения секрета Щамира для n участников, при которой обеспечивается востановление секрета любыми t участниками, $1 \leq t \leq n$. Величины p, n, t задаются пользователем с консоли, $r_1 = 1, \ldots, r_n = n$. Коэффициенты a_i исходного многочлена $f(x) = \sum_{i=0}^{t-1} a_i x^i$ инициализируются случайно (как элементы $\mathbb{Z}_p$) при запуске программы. Текст программы может быть найден в приложении А.

2 Расчёт ключевых значений

Студент выполнял вариант № k=4.

2.1 p=73, t=7, n=15

$$f(x) = 17 + 54x + 52x^2 + 14x^3 + 13x^4 + 70x^5 + 55x^6$$

$$\begin{array}{ccc}
s_1 = 56 & s_6 = 55 & s_{11} = 24 \\
s_2 = 62 & s_7 = 46 & s_{12} = 58 \\
s_3 = 53 & s_8 = 35 & s_{13} = 6 \\
s_4 = 29 & s_9 = 64 & s_{14} = 28 \\
s_5 = 62 & s_{10} = 39 & s_{15} = 60
\end{array}$$

$$\omega_1 = 7 \quad \omega_2 = 52 \quad \omega_3 = 35 \quad \omega_4 = 38 \quad \omega_5 = 21 \quad \omega_6 = 66 \quad \omega_7 = 1$$

$$s = \sum_{i=1}^{t} s_i \omega_i = 56{\cdot}7{+}62{\cdot}52{+}53{\cdot}35{+}29{\cdot}38{+}62{\cdot}21{+}55{\cdot}66{+}46{\cdot}1 \mod p = 17$$

2.2 p=139, t=5, n=10

$$f(x) = 131 + 6x + 77x^2 + 127x^3 + 63x^4$$

$$\begin{array}{cc} s_1 = 126 & s_6 = 124 \\ s_2 = 112 & s_7 = 0 \\ s_3 = 61 & s_8 = 0 \\ s_4 = 67 & s_9 = 133 \\ s_5 = 68 & s_{10} = 113 \end{array}$$

$$\omega_1 = 5 \quad \omega_2 = 129 \quad \omega_3 = 10 \quad \omega_4 = 134 \quad \omega_5 = 1$$

$$s = \sum_{i=1}^{t} s_i \omega_i = 126 \cdot 5 + 112 \cdot 129 + 61 \cdot 10 + 67 \cdot 134 + 68 \cdot 1 \mod p = 131$$

ПРИЛОЖЕНИЯ

A Текст программы

```
import numpy as np
import argparse

def extendedEuclideanAlgorithm(a, b):
    if abs(b) > abs(a):
        (x,y,d) = extendedEuclideanAlgorithm(b, a)
        return (y,x,d)

    if abs(b) == 0:
        return (1, 0, a)

    x1, x2, y1, y2 = 0, 1, 1, 0
    while abs(b) > 0:
        q, r = divmod(a,b)
        x = x2 - q*x1
        y = y2 - q*y1
        a, b, x2, x1, y2, y1 = b, r, x1, x, y1, y

    return (x2, y2, a)

def inverse(n, p):
    x,y,d = extendedEuclideanAlgorithm(n, p)
    return x % p

def generate_polinomial(t, p):
    secret = np.random.randint(p)
    coefs  = np.random.randint(p, size=t-2)
    last_coef = np.random.randint(1, p)
    polinomial = np.concatenate([[secret], coefs, [last_coef]])
    return polinomial


def print_polinomial(polinomial):
    monoms = []
    for i,g_i  in enumerate(polinomial):
        if g_i == 0:
            continue

        mon = (str(g_i))

        if i > 1:
            mon += (f" x^{i}")
        elif i == 1:
```

```python
            mon += (" x")

        monoms += [mon]
    return " + ".join(monoms)

def get_secret(polinomial, r_i, p):
    x = [(r_i**j % p) for j in range(len(polinomial))]
    secret = np.dot(polinomial, x) % p
    return secret

def get_omega(i, r, t, p):
    prod = 1
    for j in range(t):
        if i != j:
            value = (r[j] * inverse( (r[i] - r[j]) % p ,  p)) % p
            prod *= value % p
    return prod % p

parser = argparse.ArgumentParser(
    description="Shamir's secret sharing sceme.")
parser.add_argument('p', type=int)
parser.add_argument('t', type=int)
parser.add_argument('n', type=int)

args = parser.parse_args()

p = args.p
t = args.t
n = args.n
r = [i for i in range(1,n+1)]

polinomial = generate_polinomial(t, p)
print(print_polinomial(polinomial))

s = []
for i,r_i in enumerate(r):
    secret = get_secret(polinomial, r_i, p)
    to_print = f"s_{i+1} = {secret}"
    print(to_print)
    s+=[secret]


s = s[:t]
omegas = []
for i in range(t):
    omega_i = get_omega(i, r[:t], t, p)
    omegas += [omega_i]
    print (f"\omega_{i+1} = {omega_i}")
```

```


print(np.dot(s, omegas) % p)
```