Реализация Схемы Блома распределения ключей

Роман Астраханцев, СКБ-171

1 февраля 2022 г.

1 Написание программы

Была написана программа на языке программирования Python, которая вычисляет ключевые элементы по схеме Блома для n участников, при которой обеспечивается стойкость к m компрометациям, $1 \leq m \leq n-2$. Величины p, n, m задаются пользователем с консоли, $r_1 = 1, \ldots, r_n = n$. Элементы симметричной матрицы коэффициентов A исходного многочлена $f(x,y) = \sum_{i=0}^{m} \sum_{j=0}^{m} a_{ij} x^i y^j$ 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, m=1, n=15

$$A = \begin{pmatrix} 62 & 51 \\ 51 & 27 \end{pmatrix}$$

$g_1(x) = 40 + 5x$	$g_6(x) = 3 + 67x$	$g_{11}(x) = 39 + 56x$
$g_2(x) = 18 + 32x$	$g_7(x) = 54 + 21x$	$g_{12}(x) = 17 + 10x$
$g_3(x) = 69 + 59x$	$g_8(x) = 32 + 48x$	$g_{13}(x) = 68 + 37x$
$g_4(x) = 47 + 13x$	$g_9(x) = 10 + 2x$	$g_{14}(x) = 46 + 64x$
$g_5(x) = 25 + 40x$	$g_{10}(x) = 61 + 29x$	$g_{15}(x) = 24 + 18x$

2.2 p=139, m=2, n=10

$$A = \begin{pmatrix} 116 & 131 & 35 \\ 131 & 84 & 83 \\ 35 & 83 & 26 \end{pmatrix}$$

$$\begin{aligned} g_1(x) &= 4 + 20x + 5x^2 \\ g_2(x) &= 101 + 75x + 27x^2 \\ g_3(x) &= 129 + 18x + 101x^2 \\ g_4(x) &= 88 + 127x + 88x^2 \\ g_5(x) &= 117 + 124x + 127x^2 \\ g_6(x) &= 77 + 9x + 79x^2 \\ g_7(x) &= 107 + 60x + 83x^2 \\ g_8(x) &= 68 + 138x \\ g_9(x) &= 99 + 104x + 108x^2 \\ g_{10}(x) &= 61 + 97x + 129x^2 \end{aligned}$$

ПРИЛОЖЕНИЯ

A Текст программы

```
import numpy as np
import argparse

def generate_matrix(m, p):
    matrix = []
    for i in range(m+1):
        zeros = np.full(i,0)
        values = np.random.randint(p, size=m+1-i)
        string = np.concatenate([zeros, values])
        matrix += [string]
    matrix = np.array(matrix)
    return matrix + np.tril(matrix.T, k=-1)

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

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

        mon = (str(g_i))

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

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

parser = argparse.ArgumentParser(
    description="Blom's key exchange scheme.")
parser.add_argument('p', type=int)
parser.add_argument('m', type=int)
parser.add_argument('n', type=int)

args = parser.parse_args()

p = args.p
m = args.m
```

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

matrix = generate_matrix(m, p)
print(matrix)

for i,r_i in enumerate(r):
    secret = get_secret(matrix, r_i, p)
    to_print = f"g_{i+1}(x) = " + secret_to_string(secret)
    print(to_print)
```