**Молдавский государственный университет**

**Факультет математики и информатики**

**Департамент Информатики**

**УПРАЖНЕНИЕ**

**Blakley**

**Выполнил**: Гуцу Даниил.

**Группа:** IA2102

**Проверила:** др.,Чербу О.

Кишинев, 2023

**КИШИНЕВ – 2023**

Алгоритм Блейкли (Blakley's algorithm) - это криптографический метод секретного разделения, разработанный Грейс Блейкли (Grace Hopper Blakley) в 1979 году. Этот метод используется для разделения секретной информации между несколькими участниками таким образом, что секрет можно восстановить только при наличии определенного числа этих участников. Алгоритм Блейкли является одним из методов схемы "порогового разделения секрета" (threshold secret sharing).

Основная идея алгоритма Блейкли заключается в следующем:

1. Секретное сообщение разбивается на несколько частей, равное числу участников (порогу).

2. Каждой части секрета присваивается свой набор параметров (коэффициентов), которые являются частью общей криптографической системы.

3. Для восстановления секрета требуется определенное количество участников (порог), которые вместе с их параметрами могут выполнить определенные вычисления и получить исходное секретное сообщение.

Этот метод обеспечивает высокую стойкость к атакам, так как для восстановления секрета злоумышленнику необходимо получить информацию от определенного числа участников, которые могут быть физически разделены и не сотрудничать между собой.

Алгоритм Блейкли и другие схемы порогового разделения секрета часто применяются в криптографии и информационной безопасности для обеспечения надежной защиты секретной информации.

Код программы:

package blakley;  
  
import java.math.BigInteger;  
import java.util.Arrays;  
import java.util.Random;  
  
public class Blackley {  
  
  
 private static final int *BIT\_LENGTH* = 32;  
  
 public static void main(String[] args) {  
  
 final int[] m = {2, 11, 3, 13, 5, 19, 7};  
 System.*out*.println("Key sharing:\nInitial array m = " + Arrays.*toString*(m));  
  
 final int S = *getIntBetweenProductsOfTwoPartsOfArray*(m);  
 System.*out*.println("Secret S between two products = " + S);  
  
 final int M = Arrays.*stream*(m).reduce(1, (a, b) -> a \* b);  
 System.*out*.println("Product M of all elements of array m = " + M);  
  
 final int[] S\_FOOTPRINTS = Arrays  
 .*stream*(m)  
 .map(mi -> S % mi)  
 .toArray();  
 System.*out*.println("Footprints S[i] (S mod m[i]) = " + Arrays.*toString*(S\_FOOTPRINTS));  
  
 System.*out*.println("Parts of key per person: ");  
 for (int i = 0; i < m.length; i++) {  
 System.*out*.printf("P%d = (%d, %d, %d)\n", i, S\_FOOTPRINTS[i], m[i], M);  
 }  
  
  
 final int[] Mi = Arrays  
 .*stream*(m)  
 .map(mi -> M / mi)  
 .toArray();  
 System.*out*.println("\nKey recovery:\nMi (M / m[i]) = " + Arrays.*toString*(Mi));  
  
 final int[] Ni = new int[m.length];  
 for (int i = 0; i < m.length; i++) {  
 Ni[i] = BigInteger  
 .*valueOf*(Mi[i])  
 .modInverse(BigInteger.*valueOf*(m[i]))  
 .intValue();  
 }  
 System.*out*.println("Ni (Mi^(-1) / mi) = " + Arrays.*toString*(Ni));  
  
 final int[] Ii = new int[m.length];  
 for (int i = 0; i < m.length; i++) {  
 Ii[i] = S\_FOOTPRINTS[i] \* Mi[i] \* Ni[i];  
 }  
 System.*out*.println("Ii (Si \* Mi \* Ni) = " + Arrays.*toString*(Ii));  
  
 final int I\_SUM = Arrays.*stream*(Ii).sum();  
 int tempProduct = 1;  
 for (int i = 0; i < m.length; i++) {  
 if (Ii[i] != 0) {  
 tempProduct \*= m[i];  
 }  
 }  
 final int S\_RECOVERED = I\_SUM % tempProduct;  
 System.*out*.println("Recovered Secret S = " + S\_RECOVERED);  
 }  
  
 public static int getIntBetweenProductsOfTwoPartsOfArray(final int[] m) {  
 if (m.length <= 1) {  
 throw new IllegalArgumentException("Invalid vector M size: <= 1");  
 }  
  
 final int half = *getHalf*(m);  
  
 final int firstPartProduct = Arrays.*stream*(Arrays.*copyOfRange*(m, 0, half))  
 .reduce(1, (a, b) -> a \* b);  
  
 final int secondPartProduct = Arrays.*stream*(Arrays.*copyOfRange*(m, half, m.length))  
 .reduce(1, (a, b) -> a \* b);  
  
  
 return (firstPartProduct < secondPartProduct) ?  
 new Random().nextInt(firstPartProduct + 1, secondPartProduct) :  
 new Random().nextInt(secondPartProduct + 1, firstPartProduct);  
 }  
  
 public static int getHalf(final int[] m) {  
 return ((m.length & 2) == 0) ?  
 m.length / 2 :  
 m.length / 2 + 1;  
 }  
  
}

Вывод программы:

Key sharing:

Initial array m = [2, 11, 3, 13, 5, 19, 7]

Secret S between two products = 686

Product M of all elements of array m = 570570

Footprints S[i] (S mod m[i]) = [0, 4, 2, 10, 1, 2, 0]

Parts of key per person:

P0 = (0, 2, 570570)

P1 = (4, 11, 570570)

P2 = (2, 3, 570570)

P3 = (10, 13, 570570)

P4 = (1, 5, 570570)

P5 = (2, 19, 570570)

P6 = (0, 7, 570570)

Key recovery:

Mi (M / m[i]) = [285285, 51870, 190190, 43890, 114114, 30030, 81510]

Ni (Mi^(-1) / mi) = [1, 9, 2, 7, 4, 2, 4]

Ii (Si \* Mi \* Ni) = [0, 1867320, 760760, 3072300, 456456, 120120, 0]

Recovered Secret S = 686