**Молдавский государственный университет**

**Факультет математики и информатики**

**Департамент Информатики**

**УПРАЖНЕНИЕ**

**Создание ключей с помощью Диффи-Хеллмана**

**Выполнил**: Гуцу Даниил.

**Группа:** IA2102

**Проверила:** др.,Чербу О.

Кишинев, 2023

**КИШИНЕВ – 2023**

Схема подписи El-Gamal:

•Генерируется p = 1013 и генератор a = 2 al gr. Z\*1013.

•Выбирается ключ a = 1009 и производится подсчёт y=a^a mod p=2^1009 mod 1013= 380.

•Публичный ключ A (p=1013, a=2, y=380), а приватный ключ это a = 1009.

•Подписывается сообщение m=1033, выбирается k=1051,

•Считается. r=a^k mod p=2^1051 mod 1013=736 и k^-1 mod (p-1)=421.

•Считается S=421\*(1033-1009\*736) mod 1012=285.

Подпись сообщения m=1033 и пара (r=1490, s=285).

•Для проверки подписи, B считается v1=380^736\*736^285 mod 1013=854. H (m) =1033 si v2=2^1033 mod 1013=242.

•Сущность B не принимает подпись потому что v1!=v2.

Результат:

Main:

public static void main(String[] args) {  
 System.*out*.println("EL-GAMAL SIGNATURE");  
  
 List<List<BigInteger>> pksk = ElGamalSignature.*KeyGen*(200);  
  
 // public key  
  
 BigInteger p = pksk.get(0).get(0);  
  
 BigInteger g = pksk.get(0).get(1);  
  
 BigInteger h = pksk.get(0).get(2);  
  
 // secret key  
  
 BigInteger p\_sk = pksk.get(1).get(0);  
  
 BigInteger x = pksk.get(1).get(1);  
  
 System.*out*.println("Message : 1033");  
  
  
  
 List<BigInteger> encrypt = ElGamalSignature.*Encrypt\_Homomorph*(p, g, h, new BigInteger("1033"));  
  
 System.*out*.println("Decrypted message : " + ElGamalSignature.*Decrypt\_homomorphe*(p\_sk, x, g, encrypt.get(0), encrypt.get(1)));  
}

Код:

import java.math.BigInteger;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Random;  
  
public class ElGamalSignature{  
  
 public static BigInteger *TWO* = new BigInteger("2");  
  
 public static List<List<BigInteger>> KeyGen(int n) {  
 // (a) take a random prime p with getPrime() function. p = 2 \* p' + 1 with prime(p') = true  
 BigInteger p = *getPrime*(n, 40, new Random());  
 // (b) take a random element in [Z/Z[p]]\* (p' order)  
 BigInteger g = *randNum*(p, new Random());  
 BigInteger pPrime = p.subtract(BigInteger.*ONE*).divide(ElGamalSignature.*TWO*);  
  
 while (!g.modPow(pPrime, p).equals(BigInteger.*ONE*)) {  
 if (g.modPow(pPrime.multiply(ElGamalSignature.*TWO*), p).equals(BigInteger.*ONE*))  
 g = g.modPow(*TWO*, p);  
 else  
 g = *randNum*(p, new Random());  
 }  
  
 // (c) take x random in [0, p' - 1]  
 BigInteger x = *randNum*(pPrime.subtract(BigInteger.*ONE*), new Random());  
 BigInteger h = g.modPow(x, p);  
 // secret key is (p, x) and public key is (p, g, h)  
 List<BigInteger> sk = new ArrayList<>(Arrays.*asList*(p, x));  
 List<BigInteger> pk = new ArrayList<>(Arrays.*asList*(p, g, h));  
 // [0] = pk, [1] = sk  
 return new ArrayList<>(Arrays.*asList*(pk, sk));  
 }  
  
 public static List<BigInteger> Encrypt(BigInteger p, BigInteger g, BigInteger h, BigInteger message) {  
 BigInteger pPrime = p.subtract(BigInteger.*ONE*).divide(ElGamalSignature.*TWO*);  
 // *TODO [0, N -1] or [1, N-1] ?* BigInteger r = *randNum*(pPrime, new Random());  
 // encrypt couple (g^r, m \* h^r)  
 return new ArrayList<>(Arrays.*asList*(g.modPow(r, p), message.multiply(h.modPow(r, p))));  
 }  
 public static List<BigInteger> Encrypt\_Homomorph(BigInteger p, BigInteger g, BigInteger h, BigInteger message) {  
 BigInteger pPrime = p.subtract(BigInteger.*ONE*).divide(ElGamalSignature.*TWO*);  
 BigInteger r = *randNum*(pPrime, new Random());  
 // encrypt couple (g^r, h^r \* g^m)  
 BigInteger hr = h.modPow(r, p);  
 BigInteger gm = g.modPow(message, p);  
 return new ArrayList<>(Arrays.*asList*(g.modPow(r, p), hr.multiply(gm)));  
 }  
 public static BigInteger Decrypt(BigInteger p, BigInteger x, BigInteger gr, BigInteger mhr) {  
 BigInteger hr = gr.modPow(x, p);  
 return mhr.multiply(hr.modInverse(p)).mod(p);  
 }  
 public static BigInteger Decrypt\_homomorphe(BigInteger p, BigInteger x, BigInteger g, BigInteger gr, BigInteger hrgm) {  
 BigInteger hr = gr.modPow(x,p);  
 BigInteger gm = hrgm.multiply(hr.modInverse(p)).mod(p);  
  
 BigInteger m = BigInteger.*ONE*;  
 BigInteger gm\_prime = g.modPow(m, p);  
  
 while (!gm\_prime.equals(gm)) {  
 m = m.add(BigInteger.*ONE*);  
 gm\_prime = g.modPow(m, p);  
 }  
  
 return m;  
 }  
 public static BigInteger getPrime(int nb\_bits, int certainty, Random prg) {  
 BigInteger pPrime = new BigInteger(nb\_bits, certainty, prg);  
 // p = 2 \* pPrime + 1  
 BigInteger p = pPrime.multiply(*TWO*).add(BigInteger.*ONE*);  
  
 while (!p.isProbablePrime(certainty)) {  
 pPrime = new BigInteger(nb\_bits, certainty, prg);  
 p = pPrime.multiply(*TWO*).add(BigInteger.*ONE*);  
 }  
 return p;  
 }  
 public static BigInteger randNum(BigInteger N, Random prg) {  
 return new BigInteger(N.bitLength() + 100, prg).mod(N);  
 }

Вывод программы:

