

SurajSG23 /  
Cryptography-Assignment

&lt;&gt; Code

Issues

Pull requests

Actions

Projects

Security

Insights



main

Cryptography-Assignment / Elgamal



SurajSG23 Create Elgamal

0c50de6 · now



63 lines (50 loc) · 2.2 KB

Code

Blame



Raw



```
1  import java.math.BigInteger;
2  import java.security.SecureRandom;
3  import java.util.Scanner;
4
5  public class ElGamal {
6
7      // Method to compute a^b mod p using BigInteger for large numbers
8      public static BigInteger modExp(BigInteger a, BigInteger b, BigInteger p) {
9          return a.modPow(b, p);
10     }
11
12     // Method to find modular inverse of a mod p
13     public static BigInteger modInverse(BigInteger a, BigInteger p) {
14         return a.modInverse(p);
15     }
16
17     public static void main(String[] args) {
18         // Scanner for user input
19         Scanner scanner = new Scanner(System.in);
20
21         // Large prime p and generator g
22         BigInteger p = new BigInteger("7873"); // A small prime for simplicity
23         BigInteger g = new BigInteger("2");    // A common generator
24         SecureRandom random = new SecureRandom();
25
26         // Key generation
27         BigInteger d = new BigInteger(256, random); // Private key d
28         BigInteger e1 = modExp(g, d, p); // Public key e1 = g^d mod p
29         System.out.println("Public Key e1: " + e1);
30         System.out.println("Private Key d: " + d);
31
32         // Taking plaintext input from the user
33         System.out.print("Enter the message (integer form): ");
34         BigInteger m = scanner.nextBigInteger(); // Message to encrypt
35         scanner.close();
36     }
```

```
37 // Encryption process
38 BigInteger r = new BigInteger(256, random); // Random r for encryption
39
40 BigInteger c1 = modExp(g, r, p); //  $c1 = g^r \bmod p$ 
41 BigInteger c2 = m.multiply(modExp(e1, r, p)).mod(p); //  $c2 = m * e1^r \bmod p$ 
42
43 System.out.println("Encrypted Message:");
44 System.out.println("c1: " + c1);
45 System.out.println("c2: " + c2);
46
47 // Decryption process
48 BigInteger c1_d = modExp(c1, d, p); //  $c1^d \bmod p$ 
49 BigInteger c1_d_inv = modInverse(c1_d, p); //  $(c1^d)^{-1} \bmod p$ 
50 BigInteger decryptedMessage = c2.multiply(c1_d_inv).mod(p); //  $m = c2 * (c1^d)^{-1} \bmod p$ 
51
52 System.out.println("Decrypted Message: " + decryptedMessage);
53 }
54 }
55
56 //Output
57 Public Key e1: 4363
58 Private Key d: 682026697195199560005988524776753066527424869990706653183792440895
59 Enter the message (integer form): 123456789
60 Encrypted Message:
61 c1: 808
62 c2: 1136
63 Decrypted Message: 276
```