IT8761 – Security Laboratory

Reshma Ramesh Babu

312217104129

Exercise 7

Aim: To implement the Diffie-Hellman Key Exchange algorithm.

Code:

```
import java.io.*;
import java.util.*;
import java.security.SecureRandom;
import java.util.Random;
import java.math.BigInteger;
import java.io.IOException;
class PrimitiveRootGenerator {
   long pr, p, phi;
   public PrimitiveRootGenerator(long p) {
      this.p = p;
      this.phi = this.p - 1;
      Vector < Long > primitiveRoots = this.getPrimitiveRoot(this.p, this.phi);
      this.pr = primitiveRoots.get(new Random().nextInt(primitiveRoots.size()));
   }
   public long getPr() {
      return pr;
   }
   private Vector < Long > getPrimitiveRoot(long p, long phi) {
      Vector < Long > primeFactors = this.genPrimesFactorsList(phi);
      Vector < Long > primitiveRoots = new Vector < > ();
```

```
    for (long i = 2; i < p; i++) {
        boolean flg = false;
        for (Long l: primeFactors) {
            BigInteger iBig = BigInteger.valueOf(i);
            BigInteger phiBig = BigInteger.valueOf(phi / l);
            BigInteger pBig = BigInteger.valueOf(p);
            BigInteger pRootBig = iBig.modPow(phiBig, pBig);
            if (pRootBig.compareTo(BigInteger.valueOf(1)) == 0) {
                flg = true;
                break;
            }
        }
        if (!flg) primitiveRoots.add(i);
    }
    return primitiveRoots;
}
private Vector < Long > genPrimesFactorsList(long phi) {
    Vector < Long > primesFactors = new Vector < > ();
    while (phi % 2 == 0) {
        primesFactors.add((long) 2);
        phi /= 2;
    }
    for (long i = 3; i <= Math.sqrt(phi); i += 2) {
        if (phi % i == 0) {
            primesFactors.add(i);
            phi /= i;
```

```
            }
        }
        if (phi > 2) {
            primesFactors.add(phi);
        }
        return primesFactors;
    }
}
class DHKey {
    BigInteger p, g;
    private Random r;
    public DHKey() {}
    public void genPrimeAndPrimitiveRoot() {
        Random rand = new SecureRandom();
        this.p = BigInteger.probablePrime(32 / 2, rand);;
        this.g = BigInteger.valueOf(new
PrimitiveRootGenerator(this.p.intValue()).getPr());
    }
    public BigInteger getP() {
        return p;
    }
    public BigInteger getG() {
        return g;
    }
    public BigInteger getFirstMessage(BigInteger firstSecretNumber) {
        return this.g.modPow(firstSecretNumber, this.p);
    }
```

```
    public BigInteger getSecondMessage(BigInteger secondSecretNumber) {
        return this.g.modPow(secondSecretNumber, this.p);
    }
    public BigInteger firstCalculationOfKey
        (BigInteger secondMessage, BigInteger firstSecretNumber) {
            return secondMessage.modPow(firstSecretNumber, this.p);
        }
    public BigInteger secondCalculationOfKey
        (BigInteger firstMessage, BigInteger secondSecretNumber) {
            return firstMessage.modPow(secondSecretNumber, this.p);
        }
}
public class DH {
    public static void menu() {
        System.out.println("1. Generate prime and primitive root");
        System.out.println("2. Enter Secret Message A");
        System.out.println("3. Enter Secret Message B");
        System.out.println("4. Display Public key A");
        System.out.println("5. Display Public key B");
        System.out.println("6. Display Shared Secret Key");
        System.out.println("7. Exit\n");
    }
    public static String bytesToString(byte[] encrypted) {
        String test = "";
        for (byte b: encrypted) {
            test += Byte.toString(b);
```

```
    }
    return test;
}
public static void main(String[] args) throws IOException {
    int choice = 0;
    Scanner inp = new Scanner(System.in);
    DHKey d = new DHKey();
    String msg1 = "", msg2 = "";
    BigInteger pub1 = BigInteger.valueOf(0), pub2 = BigInteger.valueOf(0);
    do {
        menu();
        choice = inp.nextInt();
        switch (choice) {
            case 1: {
                d.genPrimeAndPrimitiveRoot();
                System.out.println("Prime Number: " + d.getP());
                System.out.println("Primitive Root: " + d.getG());
                break;
            }
            case 2: {
                System.out.println("Enter secret message A: ");
                msg1 = inp.next();
                break;
            }
            case 3: {
                System.out.println("Enter secret message B: ");
```

```
                msg2 = inp.next();
                break;
            }
            case 4: {
                pub1 = d.getFirstMessage(new BigInteger(msg1.getBytes()));
                System.out.println("Public Key A: " + pub1);
                break;
            }
            case 5: {
                pub2 = d.getSecondMessage(new BigInteger(msg2.getBytes()));
                System.out.println("Public Key B: " + pub2);
                break;
            }
            case 6: {
                System.out.println("Shared Key A: " + d.firstCalculationOfKey(pub2,
new BigInteger(msg1.getBytes())));
                System.out.println("Shared Key B: " +
d.secondCalculationOfKey(pub1, new BigInteger(msg2.getBytes())));
                break;
            }
            case 7:     break;
            default:
                System.out.println("Invalid");
        }
    } while  (choice != 7);
  }
}
```

Output:

```
C:\Users\Reshma\Desktop\cnslab\ex7>javac DH.java

C:\Users\Reshma\Desktop\cnslab\ex7>java DH
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

1
Prime Number: 41651
Primitive Root: 5397
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

2
Enter secret message A:
reshma
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

3
Enter secret message B:
rameshbabu
```

```
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

4
Public Key A: 10695
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

5
Public Key B: 37336
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

6
Shared Key A: 38478
Shared Key B: 38478
1. Generate prime and primitive root
2. Enter Secret Message A
3. Enter Secret Message B
4. Display Public key A
5. Display Public key B
6. Display Shared Secret Key
7. Exit

7

C:\Users\Reshma\Desktop\cnslab\ex7>
```