ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ 2017-2018

ΟΜΑΔΑ:

ΖΕΡΚΕΛΙΔΗΣ ΔΗΜΗΤΡΗΣ Π14046

BONUS PROJECT:

Αυτό το προτζεκτ το κάναμε επειδή είχαμε πει κάτι στο μάθημα της ασφάλειας για diffie-hellman ανταλλαγή κλειδιού.

Είναι ένας τρόπος να ανταλλάξουν κλειδιά ένας σερβερ και ένας πελάτης χωρίς να ξέρει κάποιος 3ος το κλειδί τους.

Έχουμε το interface:

```java
import java.rmi.Remote;

// Creating Remote interface for our application
public interface Hello extends Remote {
    public int genkey(int g,int p,int a) throws RemoteException;
    public int power(int g,int b,int p) throws RemoteException;
    public int primality(byte[] number) throws RemoteException;
    public long fibonacci(byte[] number) throws RemoteException;
    public long genfib(int number) throws RemoteException;
    public String tolower(byte[] chararr) throws RemoteException;
    public int palindrome(byte[] chararr) throws RemoteException;
}
```

την υλοποίηση του:

```java
import java.io.PrintWriter;

public class ImplExample extends UnicastRemoteObject implements Hello
{

    protected ImplExample() throws RemoteException {
        super();
        // TODO Auto-generated constructor stub
    }

    public int key = 0;
    public long[] arr = new long[100000];
```

Η μέθοδος power υπολογίζει το κλειδί μέσα απο πράξεις κάνει ουσιαστικά g^b mod p

```
public int power(int g,int b,int p)
{
  long res=1;
  long x=Long.valueOf(g);
  while(b>0)
  {
     if(b%2==1)
     {
        res=(res%p*x%p)%p;
     }
     b/=2;
     x=(x%p*x%p)%p;
  }
  res=res%p;
  return (int) res;
}
```

Αυτή η μέθοδος κάνει generate το κλειδί τυχαία χρησιμοποιώντας και την power.
Η συνάρτηση primality τσεκάρει ποιος αριθμός είναι πρώτος.

```
public int genkey(int g,int p,int a)
{
    Random rand = new Random();
    int b = rand.nextInt(10000)+100;
    int val = power(g,b,p);
    key = power(val,a,p);
    return val;
}

public int primality(byte[] number)
{
    byte[] bytes = ByteBuffer.allocate(4).putInt(key).array();
    int i=0;
    int value=0;
    for(byte b: number)
    {
        int shift = (4 - 1 - i) * 8;
        value += ((b^bytes[i%4]) & 0x000000FF) << shift;
        i++;
    }
    System.out.println(i);
    for(i=2;i<=Math.sqrt(value);i++)
    {
        if(value%i==0)
        {
            return 0;
        }
    }
    return 1;
}
```

Αυτή η μέθοδος δημιουργεί μια ακολουθία fibonacci μέχρι τον αριθμό που θα βάλουμε, χρησιμοποιώντας την genfib.

```
public long fibonacci(byte[] number)
{
    byte[] bytes = ByteBuffer.allocate(4).putInt(key).array();
    int i=0;
    int value=0;
    for(byte b: number)
    {
        int shift = (4 - 1 - i) * 8;
        value += ((b^bytes[i%4]) & 0x000000FF) << shift;
        i++;
    }
    System.out.println(value);
    return genfib(value);
}

public long genfib(int number)
{
    if(number<=0)
    {
        return (long) 0;
    }
    if(number==1)
    {
        return (long) 1;
    }
    if(number<100000 && arr[number]!= (long) 0)
    {
        return arr[number];
    }
    if(number<100000)
    {
        arr[number] = genfib(number-1)+genfib(number-2);
        return arr[number];
    }
    return genfib(number-1)+genfib(number-2);
}
```

Μετατρέπει τα γράμματα όλα σε μικρά.

```
public String tolower(byte[] chararr)
{
    byte[] bytes = ByteBuffer.allocate(4).putInt(key).array();
    ByteBuffer bytes1 = ByteBuffer.allocate(chararr.length);
    for(int i=0;i<chararr.length;i++)
    {
        int t = chararr[i]^bytes[i%4];
        bytes1.put((byte) t);
    }
    byte[] bytes2 = bytes1.array();
    String n = new String(bytes2);
    StringBuilder b = new StringBuilder();
    for(int i = 0; i < n.length(); i++)
    {
        char c = n.charAt(i);

        if(Character.isLowerCase(c) == true)
        {
            b.append(String.valueOf(c).toUpperCase());
        }
        else
        {
            b.append(String.valueOf(c).toLowerCase());
        }
    }
    return b.toString();
}
```

Τσεκάρει αν μια λέξη είναι ίδια αν γραφτεί και ανάποδα.

```
public int palindrome(byte[] chararr)
{
    byte[] bytes = ByteBuffer.allocate(4).putInt(key).array();
    ByteBuffer bytes1 = ByteBuffer.allocate(chararr.length);
    for(int i=0;i<chararr.length;i++)
    {
        int t = chararr[i]^bytes[i%4];
        bytes1.put((byte) t);
    }
    byte[] bytes2 = bytes1.array();
    String n = new String(bytes2);
    int i=0;
    int j=n.length()-1;
    while(i<j)
    {
        if(n.charAt(i)!=n.charAt(j))
        {
            return 0;
        }
        i++;
        j--;
    }
    return 1;
}
```

Server program:

```
import java.rmi.*;


public class Server {
   public static void main(String args[]) {


      try
      {
          Registry reg = LocateRegistry.createRegistry(5000);
          reg.bind("hello", new ImplExample());
      }
      catch(Exception ex)
      {
          System.out.println ("server failed with error: " + ex);
      }
   }
}
```

Client:

```
public static void main(String[] args) {
   try {
       String url="rmi://localhost:5000/hello";
       Hello stub=(Hello)Naming.lookup(url);
```

καλούμε το
remote object.

```
Random rand = new Random();
int g = rand.nextInt(10000)+100;
int a = rand.nextInt(10000)+100;
int p = 1000000007;
int val = stub.genkey(g,p,a);
int key = new Client().power(val,a,p);
byte[] bytes = ByteBuffer.allocate(4).putInt(key).array();
// Calling the remote method using the obtained object
```

Όπως βλέπουμε εδώ δημιουργούμε το κλειδί στον πελάτη και τον σώνουμε σε ένα array bytes. Όμως όπως το δημιουργούμε καλούμε τη genkey από το απομακρυσμένο αντικείμενο και μετά καλούμε απο τον πελάτη την συνάρτηση power .

Με αυτόν τον τρόπο ο πελάτης και ο σέρβερ έχουν το ίδιο κλειδί.

Έπειτα στη κονσόλα μπορούμε να πληκτρολογήσουμε τη λειτουργία θέλει να κάνει ο σερβερ για μας όπως θα φανεί παρακάτω.

Θα μπορούσαμε μετά απο αυτήν την ανταλλαγή κλειδιού να κρυπτογραφούμε με όποιον αλγόριθμο θέλουμε και να αποκρυπτογραφούμε και να επικοινωνούν έτσι ο client – server.

```
exchange-master$ java Client
>>primality 4
Not a prime number
>>
```

```
>>fibonacci 6
8
>>
```

```
>>palindrome dog
Not a palindrome
>>
```

```
>>palindrome lol
Palindrome!
>>tolower ASDF
asdf
>>
```

Στην αρχή του προγράμματος εκτυπώνουμε το κλειδί για να δείξουμε ότι είναι ίδιο: