RSA.java

```
import java.util.*;
import java.math.*;

class RSA
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int p,q,n,z,d=0,e,i;
        System.out.println("Enter the number to be encrypted and decrypted");
        int msg=sc.nextInt();
        double c;
        BigInteger msgback;
        System.out.println("Enter 1st prime number p");
        p=sc.nextInt();
        System.out.println("Enter 2nd prime number q");
        q=sc.nextInt();

        n=p*q;
        z=(p-1)*(q-1);
        System.out.println("the value of z = "+z);

        for(e=2;e<z;e++)
        {
            if(gcd(e,z)==1)         // e is for public key exponent
            {
                break;
            }
        }
        System.out.println("the value of e = "+e);
        for(i=0;i<=9;i++)
        {
            int x=1+(i*z);
            if(x%e==0)     //d is for private key exponent
            {
                d=x/e;
                break;
            }
        }
        System.out.println("the value of d = "+d);
        c=(Math.pow(msg,e))%n;
        System.out.println("Encrypted message is : -");
        System.out.println(c);
      //converting int value of n to BigInteger
        BigInteger N = BigInteger.valueOf(n);
        //converting float value of c to BigInteger
        BigInteger C = BigDecimal.valueOf(c).toBigInteger();
        msgback = (C.pow(d)).mod(N);
        System.out.println("Derypted message is : -");
        System.out.println(msgback);
        }

        static int gcd(int e, int z)
        {
                if(e==0)
                        return z;
                else
                        return gcd(z%e,e);
        }
}
```

OUTPUT-

```
$ javac RSA.java
$ java RSA


Enter the number to be encrypted and decrypted
963
Enter 1st prime number p
103
Enter 2nd prime number q
107
the value of z = 10812
the value of e = 5
the value of d = 4325
Encrypted message is : -
6206.0
Derypted message is : -
963
```

DIFFIE_HELLMAN.java

```
import java.io.*;
import java.math.BigInteger;


public class DEFFIE_HELLMAN {

 public static void main(String[]args)throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter prime number:");
        BigInteger p=new BigInteger(br.readLine());
        System.out.print("Enter primitive root of "+p+":");
        BigInteger g=new BigInteger(br.readLine());
        System.out.println("Enter value for x less than "+p+":");
        BigInteger x=new BigInteger(br.readLine());
        BigInteger R1=g.modPow(x,p);
        System.out.println("R1="+R1);
        System.out.print("Enter value for y less than "+p+":");
        BigInteger y=new BigInteger(br.readLine());
        BigInteger R2=g.modPow(y,p);
        System.out.println("R2="+R2);
        BigInteger k1=R2.modPow(x,p);
        System.out.println("Key calculated at Alice's side:"+k1);
        BigInteger k2=R1.modPow(y,p);
        System.out.println("Key calculated at Bob's side:"+k2);
        System.out.println("deffie hellman secret key Encryption has Taken");
    }

}
```

OUTPUT-

```
$ javac DEFFIE_HELLMAN.java
$ java DEFFIE_HELLMAN
Enter prime number:
29
Enter primitive root of 29:7
Enter value for x less than 29:
15
R1=7
Enter value for y less than 29:19
R2=16
Key calculated at Alice's side:16
Key calculated at Bob's side:16
deffie hellman secret key Encryption has Taken
```

ECCKeyGeneration.java

```
import java.security.*;
import java.security.spec.*;

public class ECCKeyGeneration {
  public static void main(String[] args) throws Exception {
    KeyPairGenerator kpg;
    kpg = KeyPairGenerator.getInstance("EC","SunEC");
    ECGenParameterSpec ecsp;
    ecsp = new ECGenParameterSpec("secp192r1");
    kpg.initialize(ecsp);

    KeyPair kp = kpg.genKeyPair();
    PrivateKey privKey = kp.getPrivate();
    PublicKey pubKey = kp.getPublic();

    System.out.println(privKey.toString());
    System.out.println(pubKey.toString());
  }
}
```

ECCProviderTest.java

```
import java.security.Provider;
import java.security.Provider.Service;
import java.security.Security;
import sun.security.ec.SunEC;

public class ECCProviderTest {
    public static void main(final String[] args) {
        Provider sunEC = new SunEC();
        Security.addProvider(sunEC);
        for(Service service : sunEC.getServices()) {
            System.out.println(service.getType() + ": "
                    + service.getAlgorithm());
        }
    }

}
```

ECCSignature.java

```
import java.math.BigInteger;
import java.security.*;
import java.security.spec.*;

public class ECCSignature {
  public static void main(String[] args) throws Exception {
    KeyPairGenerator kpg;
    kpg = KeyPairGenerator.getInstance("EC","SunEC");

    ECGenParameterSpec ecsp;
    ecsp = new ECGenParameterSpec("sect163k1");
    kpg.initialize(ecsp);

    KeyPair kp = kpg.genKeyPair();
    PrivateKey privKey = kp.getPrivate();
    PublicKey pubKey = kp.getPublic();
    System.out.println(privKey.toString());
    System.out.println(pubKey.toString());

    Signature ecdsa;
    ecdsa = Signature.getInstance("SHA1withECDSA","SunEC");
    ecdsa.initSign(privKey);

    String text = "In teaching others we teach ourselves";
    System.out.println("Text: " + text);
    byte[] baText = text.getBytes("UTF-8");

    ecdsa.update(baText);
    byte[] baSignature = ecdsa.sign();
    System.out.println("Signature: 0x" + (new BigInteger(1,
baSignature).toString(16)).toUpperCase());

    Signature signature;
    signature = Signature.getInstance("SHA1withECDSA","SunEC");
    signature.initVerify(pubKey);
    signature.update(baText);
    boolean result = signature.verify(baSignature);
    System.out.println("Valid: " + result);
  }
}
```

OUTPUT:-

```
$ javac ECCProviderTest.java
$ java ECCProviderTest
KeyFactory: EC
AlgorithmParameters: EC
Signature: NONEwithECDSA
Signature: SHA1withECDSA
Signature: SHA224withECDSA
Signature: SHA256withECDSA
Signature: SHA384withECDSA
Signature: SHA512withECDSA
Signature: NONEwithECDSAinP1363Format
Signature: SHA1withECDSAinP1363Format
Signature: SHA224withECDSAinP1363Format
Signature: SHA256withECDSAinP1363Format
Signature: SHA384withECDSAinP1363Format
Signature: SHA512withECDSAinP1363Format
KeyPairGenerator: EC
KeyAgreement: ECDH
$ javac ECCKeyGeneration.java
$ java ECCKeyGeneration
```

```
sun.security.ec.ECPrivateKeyImpl@ffffd30c
Sun EC public key, 192 bits
  public x coord: 1733923460052962372930193434986726966525151190608328894154
  public y coord: 2292742578308248509261161618133361543167297748428548229882
  parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)
$ javac ECCSignature.java
$ java ECCSignature
sun.security.ec.ECPrivateKeyImpl@7e76
Sun EC public key, 163 bits
  public x coord: 8249576310643032065529587246223198621267354545708
  public y coord: 6082156226444628202450207458872312054470383824821
  parameters: sect163k1 [NIST K-163] (1.3.132.0.1)
Text: In teaching others we teach ourselves
Signature:
0x302D0215011A0619ED15E478824308A610FE738978D7D2E1BC02140FFE0C215A277650CA
0957B12455617EEA356440
Valid: true
```

SDES.java

```
 import java.io.*;
import java.lang.*;

class SDES
       {
        public int K1, K2;
        public static final int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
        public static final int P10max = 10;
        public static final int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9};
        public static final int P8max = 10;
        public static final int P4[] = { 2, 4, 3, 1};
        public static final int P4max = 4;
        public static final int IP[] = { 2, 6, 3, 1, 4, 8, 5, 7};
        public static final int IPmax = 8;
        public static final int IPI[] = { 4, 1, 3, 5, 7, 2, 8, 6};
        public static final int IPImax = 8;
        public static final int EP[] = { 4, 1, 2, 3, 2, 3, 4, 1};
        public static final int EPmax = 4;
        public static final int S0[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,
                                                         3},{ 3, 1, 3, 2}};
        public static final int S1[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,
                                                         2},{ 2, 1, 0, 3}};

       public static int permute( int x, int p[], int pmax)
       {
         int y = 0;
         for( int i = 0; i < p.length; ++i)
          {
            y <<= 1;
            y |= (x >> (pmax - p[i])) & 1;
        }
         return y;
       }

       public static int F( int R, int K)
       {
          int t = permute( R, EP, EPmax) ^ K;
          int t0 = (t >> 4) & 0xF;
          int t1 = t & 0xF;
          t0 = S0[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];
          t1 = S1[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];
           t = permute( (t0 << 2) | t1, P4, P4max);
         return t;

  }

  public static int fK( int m, int K)
        {
           int L = (m >> 4) & 0xF;
           int R = m & 0xF;
           return ((L ^ F(R,K)) << 4) | R;
```

```
      }

public static int SW( int x)
{
 return ((x & 0xF) << 4) | ((x >> 4) & 0xF);
}

      public byte encrypt( int m)

      {
       System.out.println("\nEncryption Process Starts........\n\n");
        m = permute( m, IP, IPmax);
       System.out.print("\nAfter Permutation : ");
        printData( m, 8);
        m = fK( m, K1);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore IP inverse : ");
        printData( m, 8);
        m = permute( m, IPI, IPImax);
        return (byte) m;

      }


      public byte decrypt( int m)

      {
        System.out.println("\nDecryption Process Starts........\n\n");
        printData( m, 8);
        m = permute( m, IP, IPmax);
        System.out.print("\nAfter Permutation : ");
        printData( m, 8);
        m = fK( m, K2);
        System.out.print("\nbefore Swap : ");
        printData( m, 8);
        m = SW( m);
        System.out.print("\nAfter Swap : ");
        printData( m, 8);
        m = fK( m, K1);
        System.out.print("\nBefore Extraction Permutation : ");
        printData( m, 4);
        m = permute( m, IPI, IPImax);
        System.out.print("\nAfter Extraction Permutation : ");
        printData( m, 8);
        return (byte) m;

      }
```

```
    public static void printData( int x, int n)
     {
       int mask = 1 << (n-1);
       while( mask > 0)
       {
       System.out.print( ((x & mask) == 0) ? '0' : '1');
       mask >>= 1;
       }
    }


    public SDES( int K)
   {
      K = permute( K, P10, P10max);
      int t1 = (K >> 5) & 0x1F;
      int t2 = K & 0x1F;
      t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);
      t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);
      K1 = permute( (t1 << 5)| t2, P8, P8max);
      t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);
      t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);
      K2 = permute( (t1 << 5)| t2, P8, P8max);

    }

   }
```

SimplifiedDES.java

```
   public class SimplifiedDES
   {

    public static void main( String args[]) throws Exception
    {
     DataInputStream inp=new DataInputStream(System.in);
     System.out.println("Enter the 10 Bit Key :");
      int K = Integer.parseInt(inp.readLine(),2);
      SDES A = new SDES( K);
      System.out.println("Enter the 8 Bit message To be Encrypt  : ");
      int m = Integer.parseInt(inp.readLine(),2);
      System.out.print("\nKey K1: ");
      SDES.printData( A.K1, 8);
      System.out.print("\nKey K2: ");
      SDES.printData( A.K2, 8);
      m = A.encrypt( m);
      System.out.print("\nEncrypted Message: ");
      SDES.printData( m, 8);
      m = A.decrypt( m);
      System.out.print("\nDecrypted Message: ");
      SDES.printData( m, 8);

    }
```

```
}
```

Output :

```
java SimplifiedDES
Enter the 10 Bit Key :
1011011010
Enter the 8 Bit message To be Encrypt  :
10110110
Key K1: 11110101
Key K2: 01100011
Encryption Process Starts........
After Permutation : 01111001
before Swap : 00001001
After Swap : 10010000
before IP inverse : 10000000
Encrypted Message: 01000000
Decryption Process Starts........
01000000
After Permutation : 10000000
before Swap : 10010000
After Swap : 00001001
Before Extraction Permutation : 1001
After Extraction Permutation : 10110110
Decrypted Message: 10110110
```

AES.java

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AES {

    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    public static String encrypt(String strToEncrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-
8")));
        }
        catch (Exception e)
        {
            System.out.println("Error while encrypting: " + e.toString());
        }
        return null;
    }

    public static String decrypt(String strToDecrypt, String secret)
    {
        try
        {
            setKey(secret);
            Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
        }
        catch (Exception e)
        {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

public static void main(String[] args)
{
    final String secretKey = "ssshhhhhhhhhhh!!!!";

    String originalString = "howtodoinjava.com";
    String encryptedString = AES.encrypt(originalString, secretKey) ;
    String decryptedString = AES.decrypt(encryptedString, secretKey) ;

    System.out.println(originalString);
    System.out.println(encryptedString);
    System.out.println(decryptedString);
}

}
```

```
Output:

example string
QS1hZ3rV7Zcz3ihoIkq8uw==
example string
```