**RSA**

import java.util.List;

import java.util.ArrayList;

import java.util.Scanner;

import java.math.BigInteger;

import java.util.Random;

class Main {

String byteToString(byte [] sample) {

String result = "";

for (byte b : sample) {

result += Byte.toString(b);

}

return result;

}

byte[] encrypt(String plainText, BigInteger E, BigInteger N){

byte [] toEncrypt = plainText.getBytes();

BigInteger m = new BigInteger(toEncrypt);

BigInteger cipher;

cipher = m.modPow(E, N);

System.out.println("The Cipher is: " + cipher);

return cipher.toByteArray();

}

byte[] decrypt (byte[] encrypted, BigInteger D, BigInteger N) {

BigInteger m = new BigInteger(encrypted);

BigInteger plain;

plain = m.modPow(D, N);

return plain.toByteArray();

}

public static void main(String[] args) {

Random random = new Random();

RSA rsa = new RSA();

Scanner sc = new Scanner(System.in);

BigInteger P, Q, N, Phi, E, D;

byte [] encryptedBytes = new byte[10000];

byte [] decryptedBytes = new byte[10000];

P = BigInteger.probablePrime(1024, random);

Q = BigInteger.probablePrime(1024, random);

while (Q.compareTo(P) == 0) {

Q = BigInteger.probablePrime(1024, random);

}

N = P.multiply(Q);

Phi = (P.subtract(BigInteger.ONE)).multiply(Q.subtract(BigInteger.ONE));

E = BigInteger.probablePrime(5, random);

while (Phi.gcd(E).compareTo(BigInteger.ONE) > 0 && E.compareTo(Phi) < 0) {

E.add(BigInteger.ONE);

}

D = E.modInverse(Phi);

while (D.compareTo(Phi) >= 0) {

D = E.modInverse(Phi);

}

String plainText = "";

System.out.println("Random Key Generated!");

System.out.println("Public Key: (\nN: " + N + ", \nE: " + E + "\n)");

System.out.println("Private Key: (\nD: " + D + ",\nP: " + P + ",\nQ: " + Q + "\n)");

System.out.println("\nMENU: ");

System.out.println("1. Get Plain Text");

System.out.println("2. Encrypt");

System.out.println("3. Decrypt");

while (true) {

char ch;

System.out.print("Enter the choice: ");

ch = sc.next().charAt(0);

if (ch == '1') {

// get input

System.out.print("Enter the plain Text: ");

sc.nextLine();

plainText = sc.nextLine();

}

else if ( ch == '2') {

// encrypt

if (plainText.equals("")) {

System.out.println("Enter the plain text first.");

continue;

}

else {

encryptedBytes = rsa.encrypt(plainText, E, N);

}

}

else if (ch == '3') {

decryptedBytes = rsa.decrypt(encryptedBytes, D, N);

System.out.println("The decrypted String is: " + new

String(decryptedBytes));

}

else {

break;

}

}

}

}

**DIFFIE**

import java.math.BigInteger;

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int P = BigInteger.probablePrime(30, new Random()).intValue();

int G = primitiveRoot(P);

System.out.println("P = "+P+" G = "+G);

BigInteger g = new BigInteger(""+G);

BigInteger p = new BigInteger(""+P);

System.out.println("Enter A's secret key ");

BigInteger xa = new BigInteger(in.next());

BigInteger ya = g.modPow(xa, p);

System.out.println("A's public key = " + ya);

System.out.println("Enter B's secret key ");

BigInteger xb = new BigInteger(in.next());

BigInteger yb = g.modPow(xb, p);

System.out.println("B's public key = " + yb);

BigInteger A\_sh = yb.modPow(xa, p);

BigInteger B\_sh = ya.modPow(xb, p);

System.out.println("A's shared secret = "+A\_sh);

System.out.println("B's shared secret = "+B\_sh);

in.close();

}

public static int primitiveRoot(int p)

{

if (!(new BigInteger(""+p).isProbablePrime(1))) return -1;

int phi = p-1;

Set<Integer> factors = primeFactors(phi);

ArrayList<Integer> ans = new ArrayList<>();

for (int i=2;i<=phi;i++)

{

boolean flag = false;

for(int x : factors)

{

if (power(i,phi/x,p) == 1)

{

flag = true;

break;

}

}

if (!flag) return i;

// ans.add(i);

}

if (ans.size() == 0) return -1;

int ind = (int)(Math.random()%ans.size());

return ans.get(ind);

// return -1;

}

public static Set<Integer> primeFactors(int p)

{

Set<Integer> facts = new HashSet<>();

while (p%2==0){

facts.add(2);

p/=2;

}

for(int i=3;i<=(int)Math.sqrt(p);i+=2){

while(p%i==0)

{

p/=i;

facts.add(i);

}

}

if (p>2)

facts.add(p);

return facts;

}

public static int power(int a,int b,int mod)

{

if (b==0)

return 1;

if (b==1)

return a%mod;

int temp = power(a,b/2,mod);

temp = (temp\*temp);

if (b%2!=0)

temp\*= a;

return temp%mod;

}

static boolean millerTest(int d, int n) {

int a = 2 + (int)(Math.random() % (n - 4));

int x = power(a, d, n);

if (x == 1 || x == n - 1)

return true;

while (d != n - 1) {

x = (x \* x) % n;

d \*= 2;

if (x == 1)

return false;

if (x == n - 1)

return true;

}

return false;

}

static boolean isPrime(int n) {

if (n <= 1 || n == 4)

return false;

if (n == 3)

return true;

int d = n - 1;

int k=n-2;

while (d % 2 == 0)

{

d /= 2;

// k++;

}

for (int i = 0; i < k; i++)

if (!millerTest(d, n))

return false;

return true;

}

}

**DSA**

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.Signature;

import java.util.Scanner;

import java.io.UnsupportedEncodingException;

import java.util.\*;

import java.security.\*;

public class Main {

public static void main(String[] args) throws NoSuchAlgorithmException, InvalidKeyException, SignatureException, UnsupportedEncodingException {

Scanner sc = new Scanner(System.in);

System.out.print("Text : ");

String msg = sc.nextLine();

KeyPairGenerator keys = KeyPairGenerator.getInstance("DSA");

keys.initialize(1024);

KeyPair key = keys.generateKeyPair();

PrivateKey pk = key.getPrivate();

PublicKey puk = key.getPublic();

Signature sign = Signature.getInstance("SHA256withDSA");

sign.initSign(pk);

byte[] text = msg.getBytes();

sign.update(text);

byte[] signature = sign.sign();

sign.initVerify(puk);

System.out.print("Data : ");

String data = sc.nextLine();

sign.update(data.getBytes());

if(sign.verify(signature))

System.out.println("Signature Verified !");

else

System.out.println("Signature Invalid !");

}

}