## Lab 21: Bellcore, common modulus, and small e attacks on RSA, and Wiener's attack

**Group Name: Srikar Maadhu**

**Lahari Karrotu**

1. Simulate Bellcore Attack disturbing mq by 1 in the decryption of Task 1. Send disturbance as additional parameter.

Check

p = gcd(M-M',n)

Implement an additional parameter to RSA\_Fast\_Decryption.

static BigInteger disturbance = BigInteger.ZERO;

...

if (args.length > 4) disturbance = new BigInteger(args[4]);

**Code:**

**import java.math.\*;**

**class RSA\_Fast\_Decryption\_Dist {**

**static BigInteger disturbance = BigInteger.ZERO;**

**public static BigInteger decrypt(BigInteger c, BigInteger dP, BigInteger dQ,**

**BigInteger p, BigInteger q,**

**BigInteger pInv, BigInteger qInv, BigInteger cqp, BigInteger cpq) {**

**BigInteger mp, mq, h, m;**

**mp = c.modPow(dP, p);**

**mq = c.modPow(dQ, q).add(disturbance);**

**System.out.println("c=" + c + "; dP=" + dP + "; p=" + p + "; mp=" + mp + " mq=" + mq);**

**h = mp**

**//.add(cqp)**

**.subtract(mq)**

**.multiply(qInv).mod(p);**

**System.out.println("h=" + h);**

**m = mq.add(q.multiply(h));**

**return m;**

**}**

**public static void main(String args[]) {**

**BigInteger n = new BigInteger(args[0]);**

**BigInteger e = new BigInteger(args[1]);**

**BigInteger c = new BigInteger(args[2]);**

**BigInteger p = new BigInteger(args[3]);**

**if (args.length > 4) disturbance = new BigInteger(args[4]);**

**BigInteger q = n.divide(p);**

**BigInteger O, d, dP, dQ, qInv, pInv;**

**BigInteger[] cqp, cpq;**

**O = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));**

**System.out.println("O = " + O);**

**d = e.modInverse(O);**

**System.out.println("d = " + d + "; p=" + p);**

**dP = d.mod(p.subtract(BigInteger.ONE));**

**if (dP.equals(BigInteger.ZERO)) dP = p.subtract(BigInteger.ONE);**

**dQ = d.mod(q.subtract(BigInteger.ONE));**

**if (dQ.equals(BigInteger.ZERO)) dQ = q.subtract(BigInteger.ONE);**

**qInv = q.modInverse(p);**

**pInv = p.modInverse(q);**

**System.out.println("dP = " + dP + "; dQ = " + dQ + "; pInv = " + pInv + "; qInv=" + qInv);**

**cqp = q.divideAndRemainder(p);**

**if (!cqp[1].equals(BigInteger.ZERO)) cqp[0] = cqp[0].add(BigInteger.ONE);**

**cqp[0] = cqp[0].multiply(p);**

**cpq = p.divideAndRemainder(q);**

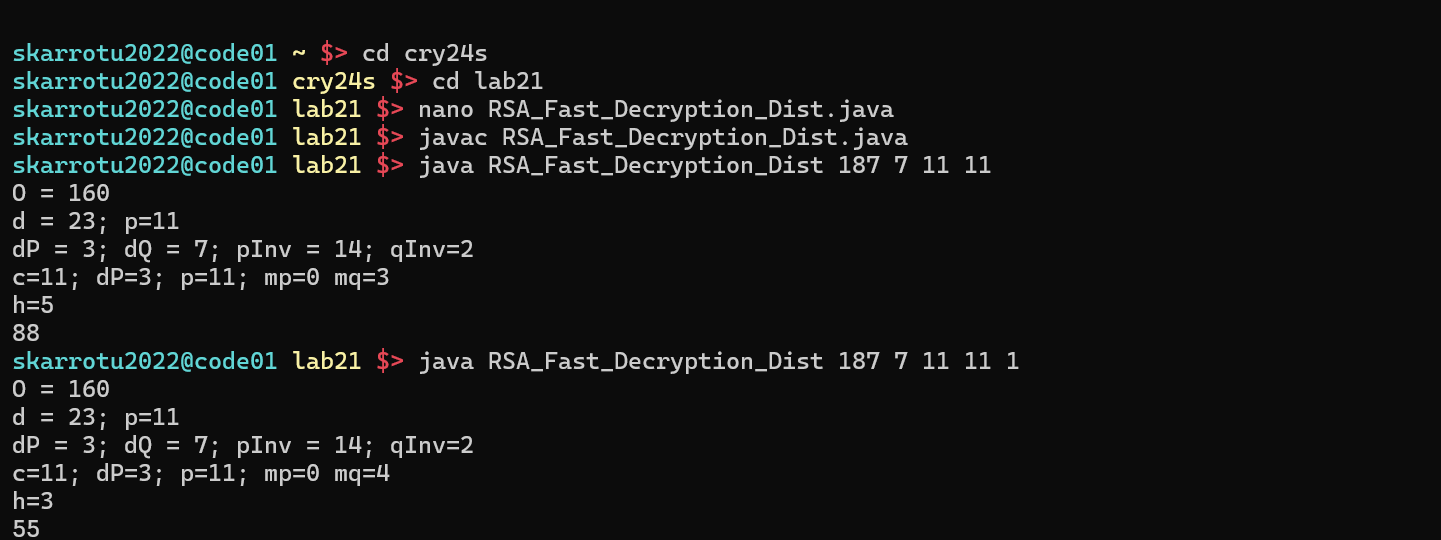
**if (!cpq[1].equals(BigInteger.ZERO)) cpq[0] = cpq[0].add(BigInteger.ONE);**

**cpq[0] = cpq[0].multiply(q);**

**System.out.println(decrypt(c, dP, dQ, p, q, pInv, qInv, cqp[0], cpq[0]));**

**}**

**}**



Then test:

java RSA\_Fast\_Decryption\_Dist 187 7 11 11

java RSA\_Fast\_Decryption\_Dist 187 7 11 11 1

Adjust EEA.java (from Lab Basic RSA) to print gcd (from variable A3).

java EEA 187 `difference of results`

**CODE:**

import java.math.BigInteger;

class EEA {

public static void main(String args[]) {

BigInteger n = new BigInteger(args[0]);

BigInteger e = new BigInteger(args[1]);

BigInteger d = eea(n, e);

System.out.println(d);

}

public static BigInteger eea(BigInteger n, BigInteger e) {

BigInteger A1 = BigInteger.ONE, A2 = BigInteger.ZERO, A3 = n;

BigInteger B1 = BigInteger.ZERO, B2 = BigInteger.ONE, B3 = e;

BigInteger Q, T1, T2, T3;

while (!B3.equals(BigInteger.ZERO)) {

if (B3.equals(BigInteger.ONE))

return (B2.compareTo(BigInteger.ZERO) < 0) ? B2.add(n) : B2; // Ensure the modular inverse is positive

Q = A3.divide(B3);

T1 = A1.subtract(Q.multiply(B1));

T2 = A2.subtract(Q.multiply(B2));

T3 = A3.subtract(Q.multiply(B3));

A1 = B1;

A2 = B2;

A3 = B3;

B1 = T1;

B2 = T2;

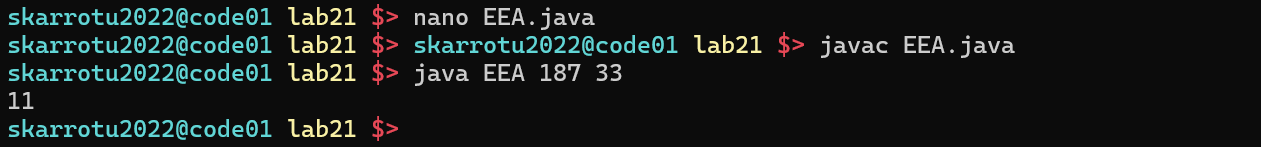
B3 = T3;

}

return A3; // Return A3 if B3 equals zero

}

}



2. Check Common Modulus attack. Implement attacker getting as parameter n, e1, e2, c1, c2.

Use your EEA implementation to get:

m= (c1^{-1})^{-r} c2^s

*where re*1+*se*2=1

Show the execution on:

* c1=185, c2=53,
* n=187, e1=7, e2=9

Using programs from Basic RSA Lab:

java ExpCipher\_Encrypt 187 7 25

java ExpCipher\_Encrypt 187 9 25

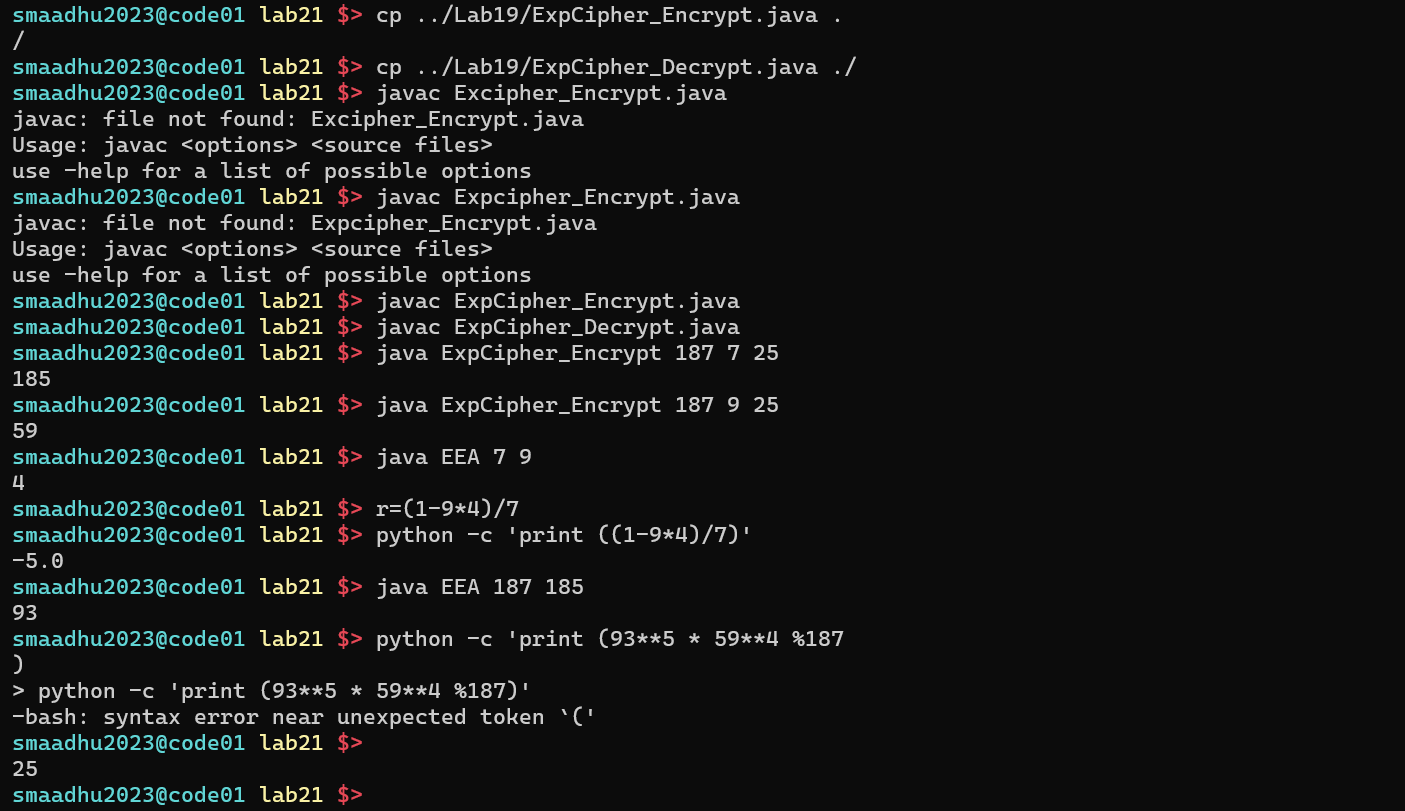
java EEA 7 9

r=(1-9\*4)/7

python -c 'print ((1-9\*4)/7)'

java EEA 187 185

python -c 'print (93\*\*5 \* 59\*\*4 %187)'



3. Implement the small e attack using e-th roots (could be based on binary search), when:

m^e < n1n2

Test with two keys:

* eA=7, NA = 65
* eB=7, NB = 77
* M=3 from cA=31 cB=42

java ExpCipher\_Encrypt 77 7 3

java ExpCipher\_Encrypt 65 7 3

java CRT2 77 65 31 42

python -c 'print (2187\*\*(1./7))'

