Readme of Assignment 1

3/11/2012 By Gong Li, Li Wanlong and Zhang Jun

1 Description of the program P1

The program P1 is written by following the Continued Fraction Algorithm stated on the paper “Cryptanalysis of Short RSA Secret Exponents” written by Michael J.Wiener. For the source code of P1, please refer to section 3 in this file. The pseudo-code is written as follows:

```
Create an ArrayList q to hold the integer quotient of each step
Create an ArrayList r to hold the remainder of each step
Create an ArrayList n to hold the numerator of fraction
Create an ArrayList d to hold the denominator of fraction

int i = 0
while(private key is not generated and continued fraction is not
finished){
      step(i)
      i++
}

if private key is generated
      write this private key into a binary file
else
      print out unsuccessful message
-------------------------------------------------------------
BigInteger step(int i){
      if (i == 0)
            add the integer quotient of e/N into q
            add the remainder of e/N into r
            add q[0] into n
            add 1 into d

      else if (i == 1)
            add the integer quotient of 1/r[0] into q
            add the remainder of 1/r[0] into r
            add q[0]*q[1] + 1 into n
            add q[1] into d
```

```
        else
              if (no further continued fraction could be generated)
                    continued fraction is finished
                    and no private key is generated
                    return -1

              add the integer quotient of 1/r[i-1] into q
              add the remainder of 1/r[i-1] into r
              add q[i]*n[i-1] + n[i-2] into n
              add q[i]*d[i-1] + d[i-2] into d

        if (i is even)
              assign <q0, q1, q2,...,qi+1> to k/dg
        else
              assign <q0, q1, q2,...,qi> to k/dg

        edg = e * (the denominator of k/dg)
        (p-1)(q-1) = the quotient of e/(k/dg)
        g = (edg) mod k

        (p+q)/2 = ((p-1)(q-1) - N + 1) / 2
        if ( (p+q)/2 is not an integer )
              the guess of k and dg is wrong
              return null

        [(p-q)/2]^2 = [(p+q)/2]^2 - N
        if ( [(p-q)/2]^2 is not a perfect square )
              the guess of k and dg is wrong
              return null

        privateKey = edg /(e*g)
        return privateKey
}
```

2 Private keys for successfully attacked cases

Private keys for case 1, 2, 3, 4, 5, 6, 8 and 12 are generated successfully, and they are stored in “P1_PrivateKey” + respective case number+”.bin”. For instance, the private key for case 8 is stored in “P1_PrivateKey8.bin”. Below are the private keys in decimal of successfully attacked cases:

Case 1:
65539

Case 2:
205367570127735096896293169268017808877278446401942833673150407401968707000 71

Case 3:
265959691767529381826561591235949647390939325476928350865404360583957880157 73

Case 4:
303066619022858127710994508046548676386081607116725579791709803660494214053 39

Case 5:
350107681674380341175756900528877693693865927233993538722472387411860119054 17

Case 6:
639587904767670829734684363050597463327040888898094948821116347431642943150 43

Case 8:
179750879714083278397039421313747654185974073555450058921569218751077423955237

Case 12:
876328174179171135866248420658986716426766752182897111854515267329448953067603310980234302363744525799184009517469467429982267940523814013665092454990 5807

The attacks to case 7, 9, 10, 11, 13 are unsuccessful.

3 Source code of the program P1 and P2

The folder “Cryptography Assignment 2012\Wiener's Attack” is actually an eclipse Java project, and it could be imported as an eclipse project using the eclipse IDE. The locations of P1 and P2 are as follows:

P1: Cryptography Assignment 2012\Wiener's Attack\src\WienerAttack.java
P2: Cryptography Assignment 2012\Wiener's Attack\src\RSAsignature.java

The other two Java classes are helper classes with the aim to ease the load of programming. Files.java is used to read files and write keys and signature into files. Fraction.java helps to calculate the continued fraction.

4 RSA digital signature

The digital signature can be generated by running RSAsignature.java, and then it is printed out in the console as well as stored in a file named “P2_Signature.bin”. The content of the generated digital signature in decimal is as follows:

```
74892770352069036830635183743189544271522080255742846896879427456
07969729747917843102674813741907622111132317760545584478799743320
99261435924643769052954579046625535799003382660808619752854392123
96829245646845783126949519865218303990878877869934330182495939031
51344688289054587024458002430820232870606150146 2
```