Zheng Ma *February 21, 2005*

Solutions to Problem Set 2

Note: Each small question is worth 5 points.

Problem 5: Number theory review

Do the following number theory problems (a little bit mathematical):

(a) Express 1 as a linear combination of 2058 and 1019. What is 1019^{-1} modulo 2058?

Solution: Let, $x < y$. From the Euclidean algorithm of GCD and EGCD, we know that $GCD(X, Y) = GCD(X, Y\%X) = GCD(Y\%X, X)$. From the procedure for calculating GCD, you can get an expression of $ax + by = GCD(x, y)$. If x and N are mutually prime, then we have $ax + bN = 1$. Taking the equation modulo N, you get $ax \equiv 1 \pmod{N}$. Therefore $x^{-1} \equiv a \pmod{N}$. Let's do the calculation of the $\gcd(2058, 1019)$:

$$2058 = 1019 \times 2 + 20 \quad (1)$$
$$1019 = 20 \times 50 + 19 \quad (2)$$
$$20 = 19 \times 1 + 1 \quad (3)$$
$$(4)$$

Proceed backwards, we get the equation:

$$1 = 20 - 19 = 20 - (1019 - 20 \times 50) = 20 \times 51 - 1019$$
$$= (2058 - 1019 \times 2) \times 51 - 1019 = 2058 \times 51 - 1019 \times 103$$

$1019^{-1} \equiv -103 \equiv 1955 \pmod{2058}$

(b) Calculate $2^{5^{49}} \bmod 29$.

Solution: (Note $2^{5^{49}} = 2^{(5^{49})}$) This question is to test your knowledge of Euler function and Euler's theorem. Notice that 29 is a prime, $\phi(29) = 28$ and $2 \in \mathbf{Z}^*_{29}$. By Euler's theorem, $2^{28} \equiv 1 \pmod{29}$. So let's calculate $5^{49} \bmod 28$. Note that $\phi(28) = 2 \times (2-1) \times (7-1) = 12$ and $5 \in \mathbf{Z}^*_{28}$. So $5^{12} \equiv 1 \pmod{28}$. So $5^{49} \equiv 5 \pmod{28}$. So $2^{5^{49}} \equiv 2^5 = 3 \pmod{29}$.

(c) Compute $\gcd(6188, 4709)$.

Solution: $\gcd(6188, 4709) = \gcd(1479, 4709) = \gcd(1479, 272) = \gcd(119, 272) = \gcd(119, 34) = \gcd(17, 34) = \gcd(17, 0) = 17$.

(d) Show that 1234 and 357 are relatively prime. Find the multiplicative inverse of 357 in $\mathbf{Z}_{1234}$.

Solution: Let's do the calculation for $\gcd(1234, 357)$ using extended Euclid algorithm.

$$\begin{array}{rrr}
(1234, & 1, & 0) \\
(\ 357, & 0, & 1) \\
(\ 163, & 1, & -3) \\
(\ 31, & -2, & 7) \\
(\ 8, & 11, & -38) \\
(\ 7, & -35, & 121) \\
(\ 1, & 46, & -159)
\end{array}$$

So 1234 and 357 are relatively prime, and $357^{-1} \equiv -159 \equiv 1075 \pmod{1234}$

Problem 6: Number theory on computer

(a) Use Erathostenes's Sieve to count the exact number of primes less than one million. What is the estimate given by the Prime Number Theorem for this number? What is the relative error of the estimate?

Solutions: I use the following simple code to calculate the primes less than one million, it is pretty quick and the result is 78498. According to the Prime Number Theorem, $\pi(10^6) = 10^6/\ln(10^6) = 72382$. So the relative error is $(78498 - 72382)/78498 = 0.0779$

```
#include <iostream.h>
#include <bitset>
using namespace std;
main(int argc,char** argv) {
        unsigned long max=1000000;
        unsigned long k,i;
        bitset<10000000> bitmap;
        int count=0;
        max = ((argc==2)? atoi(argv[1]):max);

        for(i=2;i<=max;i++){
          if(!bitmap.test(i)){
             for(k=i+i;k<=max;k+=i)bitmap.set(k);
        }
        }
        for(i=2;i<=max;i++){
           if(!bitmap.test(i))count++;
        }
        cout << count << '\n';
}
```

(b) Get a ball-park figure of the time it takes to generate 1024-bit (308 decimal digits) and 2048-bit RSA keys on a modern PC by implementing RSA key generation using ln3 or any other big number package. Submit both the answer to this question and also the documented code that you wrote.

Solution:

I use the following code to implement RSA. For 1024 bits and 2048 bits, the total run time of the RSA is attached. (Each program was run 10 times.) The average time of 1024 bits is 11.0576s and the average time of 2048 bits is 158.033s

```
#include <lnv3/lnv3.h>
#include <nttl/gcd.h>
#include <nttl/randomPrime.h>
#include <nttl/inverse.h>
#include <sys/time.h> // for timeval and gettimeofday
#include <stdio.h>
```

```
#define TIME_DIF_TO_NS(s,f) \
    ((f.tv_sec-s.tv_sec)*1000000000.0 + (f.tv_usec-s.tv_usec)*1000.0)

main(int argc, char** argv){
        ln N,phiN,p,q,e,d,temp;
        double sample_s,sample_ns;
        struct timeval start,finish;
        int i,bitlen=154,times=1;
        bitlen = ((argc > 1) ? atoi(argv[1]):bitlen);
        times = ((argc > 2) ? atoi(argv[2]):times);

        for(i=0;i<times;i++){
         gettimeofday(&start,NULL);
         RandomPrime(&p,bitlen,20);
         RandomPrime(&q,bitlen,20);
         N=p*q;
         phiN = (p-1)*(q-1);
         temp = 2;
         while(temp > 1){
                printf("bitlen is %d\n",2*bitlen);
                RandomPrime(&e,bitlen*2,20);
                temp = GCD(e,phiN);
         }
         d = Inverse(e,phiN);
         gettimeofday(&finish,NULL);
         sample_ns = TIME_DIF_TO_NS(start,finish);
         sample_s = sample_ns / 1000000000.0;
         printf("%f\n",sample_s);
        }
}
```

Problem 7: RSA: Theory and Practice

Both mathematical and practical:

(a) (With pen) My toy RSA key is $N = 187$, $e = 107$. You observe a ciphertext $c = 5$. What is the plaintext?

Solution: Notice that $N = 187 = 11 \times 17$, so $\phi(N) = (11-1) \times (17-1) = 160$. By using the GCD algorithm we can get $d = 3 \equiv 107^{-1} \pmod{160}$. So $D(c, N, d) = c^d \bmod N = 5^3 \bmod 187 = 125$.

(b) (With computer) Consider this RSA key

$$N : \begin{array}{l} 120457322460183418712065226069810172366048205604752299621042894 \\ 397706979004293595855337673954215374262637794524272045818217908 \\ 229478526584500478161639581465312338385782996541477205027111304 \end{array}$$

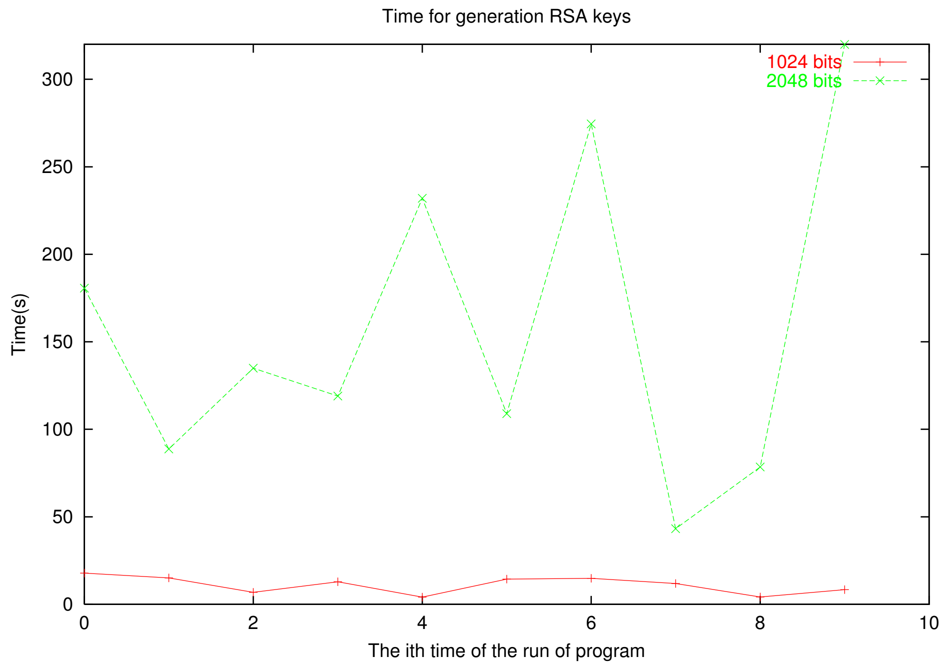

Figure 1: Ballpark graph of RSA calculation

$$
\begin{aligned}
& 594623121868092711384962797824273219760781441239953781771300913 \\
& 27992893347504900757047650844010384728900224247825638890 9.
\end{aligned}
$$

$$
\begin{aligned}
e \quad : \quad & 108320892662862955596302357470782070713553067937889499942447147 \\
& 355968656694359903462867764882012512073853365701549521606511348 \\
& 714104172770325051335157387268754815037274565436944534867153132 \\
& 292684133742464029214648540750182171588972138378036750055446438 \\
& 6980762970780272482741827118729743260764294385077942709 51.
\end{aligned}
$$

Suppose you manage to obtain the decryption key

$$
\begin{aligned}
d \quad : \quad & 870778041505896784929057064247111975431088041103712624121674096 \\
& 250991494449208495397081617394569729997785285908978755792242047 \\
& 951504608606471891232362973145723219036260087876400092012690445 \\
& 506827289066083911176383869641400399764707038140971639114758576 \\
& 803956361759865393827611426273085021285958953198368839 91.
\end{aligned}
$$

Factor N. (These numbers will be placed in the file `/c/cs467/assignments/ps2/rsakeys.txt` on the Zoo so that you don't need to copy them by hand.)

Solution: (Thanks for Melody Chan to let me use part of her solution here.)

The following programs just follows the pseudo-code provided in our lectures, Week 5, Figure 1.

```
#include <lnv3/lnv3.h>
#include <nttl/gcd.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <fstream>
#include <ctype.h>

int
main( int argc, char* argv[] )
{

  ln s, t, a, b, p, q, n, e, d;

  /* These values for n, e, and d are given by
  in /c/ccs467/assignments/ps2/rsakeys.txt. */

  n = "1204573224601834187120652260698101723660482056
  0475229962104289439770697900429359585533767395421 53
  7426263779452427204581821790822947852658450047816 16
  3958146531233838578299654147720502711130459462312 18
  6809271138496279782427321976078144123995378177130 09
  1327992893347504900757047650844010384728900224247 82
  56388909";

  e = "1083208926628629555963023574707820707135530679
  3788949999424471473559686566943599034628677648820125
  1207385336570154952160651134871410417277032505133 51
  5738726875481503727456543694453486715313229268413 37
  4246402921464854075018217158897213837803675005544 64
  3869807629707802724827418271187297432607642943850 77
  94270951";

  d = "8707780415058967849290570642471119754310880411
  0371262412167409625099149444920849539708161739456 97
  2999778528590897875579224204795150460860647189123 23
  6297314572321903626008787640009201269044550682728 90
  6608391117638386964140039976470703814097163911475 85
  7680395636175986539382761142627308502128595895319 83
  6883991";

  /* Find s and t satisfying ed - 1 = (2^s) * t,
  where s is the highest power of 2 dividing ed-1 */

  s = 0; t = e*d - 1;

  while ((t % 2) ==0) {
    s++;
```

```
    t /= 2;
  }

  /* Search for nontrivial square root of 1 */

  do {

    while (GCD((a = ln().Random(n.GetSize()) % n), n) != 1);
    b = a.FastExp(t, n);
    while ((b * b - 1) % n != 0) {b = b.FastExp(2,n) % n;}

  /* We need only check whether b is congruent to -1 mod n */
  } while ((b + 1) % n == 0);

  /* p and q are the factors of n */

  p = GCD(b-1, n);
  q = n/p;

  cout<<p<<endl<<q<<endl;

  return 0;

}
```

The result is

```
p=89889346615430307833703531491368016351546
1245785775971241172207399930747228455371345
6086186589457066485259896373658921765496420
1755944158734989493887824117

q=13400622765179366124022968566392587747304
9260004479416277756494679806323152803850944
4724172534755335820918890908406623138835495
798093288574998983673650777
```