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 \(gcd (2058, 1019)\):

\[
\begin{align*}
2058 &= 1019 \times 2 + 20 \\
1019 &= 20 \times 50 + 19 \\
20 &= 19 \times 1 + 1
\end{align*}
\]

Proceed backwards, we get the equation:

\[
1 = 20 - 19 = 20 - (1019 - 20 \times 50) = 20 \times 51 - 1019
\]

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

(b) Calculate \(2^{549} \mod 29\).

Solution: (Note \(2^{549} = 2^{(5\times 549)}\)) This question is to test your knowledge of Euler function and Euler’s theorem. Notice that 29 is a prime, \(\phi(29) = 28\) and \(2 \in \mathbb{Z}_{29}^*\). By Euler’s theorem, \(2^{28} \equiv 1 \pmod{29}\). So let’s calculate \(5^{49} \mod 28\). Note that \(\phi(28) = 2 \times (2-1) \times (7-1) = 12\) and \(5 \in \mathbb{Z}_{28}^*\). So \(5^{12} \equiv 1 \pmod{28}\). So \(5^{49} \equiv 5 \pmod{28}\). So \(2^{549} \equiv 2^5 = 3 \pmod{29}\).

(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 \(\mathbb{Z}_{1234}\).

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

\[
\begin{align*}
(1234, & \quad 1, \quad 0) \\
(357, & \quad 0, \quad 1) \\
(163, & \quad 1, \quad -3) \\
(31, & \quad -2, \quad 7) \\
(8, & \quad 11, \quad -38) \\
(7, & \quad -35, \quad 121) \\
(1, & \quad 46, \quad -159)
\end{align*}
\]
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 $(78498 - 72382)/78498 = 0.0779$

```cpp
#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 << '
';
}
```

(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

```cpp
#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>
```
```c
#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 $D(c, N, d) = c^d \mod N = 5^3 \mod 187 = 125$.

(b) (With computer) Consider this RSA key

```
N : 120457322460183418712065226069810172366048205604752299621042894397706979004293595853376739542153742626377954272045818217908229478526584500478161639581465312338385782996541477205027111304
```
Figure 1: Ballpark graph of RSA calculation

Suppose you manage to obtain the decryption key

\[ d = 870778041505896784929057064247111975431088041103712624121674096 \\
25099149449208495397081617394569729997785285908978755792242047 \\
951504608606471891232362973145723219036260087876400092012690445 \\
5068272890660839111763836869641400399764707038140971639114758576 \\
8039563617598653938276114262730850212859589531983685991. \]

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
04752299621042894397706979004293595855337639542153
74262637794524272045818217908229478526585004781616
39581465312338385782996541477205027113045946231218
680927113849627978242732197607814412399537817713009
132799289334750490075704765084401038472890022424782
56388909";

    e = "1083208926628629555963023574707820707135530679
37889499942447147355596856694359903462867648820125
120738533657015495216065113487141041727703250513351
573872687548150372745654369445348671531322926841337
424640292146485407501821715889721383780367500554464
3869807629707802724827418271187297432600764294385077
94270951";

    d = "87077804150508678492920570642471119754310880411
037126241216740962509914944492084953970816173945697
29997858258908978755792242047951504608064718912323
62973145723219036260878764000920126904455068272890
660839111763838696414003997647070381409716391147585
76803956361759865393827611426273085021285958931983
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 60861865894570666485259896373658921765496420 1755944158734989493887824117

q=13400622765179366124022968566392587747304 926000447941627756494679806323152803850944 4724172534755335820918890908406623138835495 798093288574998983673650777