Simple Documentation for RSA Implementation

Abraham Xiao

December 9, 2013

1 Introduction

For convenience, we cite some facts and description from [1] without much more mentioning. We hope this will not intrigue intelligence property issues.

Definition 1 *The* RSA problem *is the following: given a positive integer n that is a product of two distinct odd primes p and q, a positive integer e such that $gcd(e,(p-1)(q-1)) = 1$, and an integer c, find an integer m such that $m^e \equiv c \mod n$.*

In other words, the RSA problem is that of finding e^{th} roots modulo a composite integer n. The condition imposed on the problem parameters n and e ensure that for each integer $c \in 0, 1, \ldots, n-1$ there is exactly one $m \in 0, 1, \ldots, n-1$ such that $m^e \equiv c \mod n$. Equivalently, the function $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_n$ defined as $f(m) = m^e \mod n$ is a *permutation.*

2 Implementation

2.1 Data Structure

A special data structure containing two primes p and q, the multiplication of $(p-1)(q-1)$ as well as public, private key pairs is defined as follows:

```
typedef struct RSA_PARAM_Tag
{
```

```
unsigned __int64 p, q;  // p and q are two primes
unsigned __int64 f; // f=(p-1)*(q-1)
unsigned __int64 n, e; // n=pq; gcd(e,f)=1 public keys
unsigned __int64 d; // private key, ed=1(mod f), gcd(n,d)=1
}RSA_PARAM;
```

A class containing a private data as well as public data, method is defined as follows:

```
class RandNumber
{
private:
unsigned __int64 randSeed;
public:
RandNumber(unsigned __int64 s = 0);
unsigned __int64 Random(unsigned __int64 n);
};
```

For the rest part we itemize features in our implementation.

- An array of small prime table is created to speed-up the process of identifying if a large number is a prime or composite.
- The seed used to generate large random number is taken from current calendar time to ensure enough randomness.
- A random number is generated in a way of multiplying a large enough number and then add another one.
- Rabin-Miller primality test is implemented. And the testing loop is adjustable.
- Both the Euclidean algorithm and binary algorithm for calculating *greatest common divisor* are implemented.
- The whole RSA algorithm is implemented neatly.

3 Samples

We use a toy sample to conclude this simple documentation. Up to now, the string with spaces is not supported. We are sorry for that, indeed.

```
abrahamx91@debian:~/Professional/Git/CIS612-Composition/Codes$
 ./a.out
p=47911
q=38839
f=(p-1)*(q-1)=1860728580
n=p*q=1860815329
e=46387
d=1574922403

 Please enter your plaintext: Abraham-Xiao-Keep-Moving!

 Ciphertext is: b58c31a 6d4c7761 15dafa09 17a7e101 2c02bb80
 17a7e101 650e1f0c 64dc1f07 2c3b1738 1189bc8c 17a7e101 19873f79
 64dc1f07 5596ced9 38a8ee68 38a8ee68 9bb7fbf 64dc1f07 49bec0cc
 19873f79 52d47daf 1189bc8c 2dd5496b 13442502 2bec903d 0

Decipher:  You plaintext should be: Abraham-Xiao-Keep-Moving!

abrahamx91@debian:~/Professional/Git/CIS612-Composition/Codes$
```

Some parts are manually modifies due to page space issues.

Acknowledgment

I would like to thank Dr. Zeyar for preparing high quality lectures throughout the whole Fall 2013 semester. In addition, I am extremely grateful for being able to carry out *care-free* research at Masdar Institute of Science and Technology, especially as a late applicant last year[1].

References

[1] Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. *Handbook of Applied Cryptography*, 1st ed. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[1] I submitted my full application just 2 weeks before the deadline. But I got the offer pretty fast.