Project D: Modular Exponentiation Report

Le Huy Duc

I. Introduction:

In this project, I implemented the 7 modular exponentiation algorithms that were described in the lecture. In the program, I performed accuracy checking, counting number of squares/multiplications and comparing them to theoretical values, as well as checking the special relation of each algorithm.

Since the algorithms are described very carefully in the slides, I only need to do copy them exactly.

The special thing in my project is that I make a C++ wrapper class (bignum.h) around GMP. This class can be used exactly the same as *int*, but can handle arbitrary-sized integer, like GMP's *mpz_t***,** and **has the same speed** as *mpz_t***.** This allows me to program this project in a very fast, simple and readable format.

II. How to run:

First, there are 3 library files: bignum.h, lib.h, rander.h

There are 3 completely separated source code files.

- **Test_accuracy.cpp** is used to check the correctness of the code. For each test, it generates *(m,d,n)* and calculate *m^d mod n* using each algorithm, then compare the result. By default *d* has 1024 bits.
- **Test_properties.cpp** is used to check the special relation in each algorithm. For example, in Joye Ladder algorithm, we have this relation:

  Invariant properties

  At the end of each loop the following holds:

  - $R_0 = m^{(d_i \ldots d_1 d_0)_2} \bmod n$
  - $R_0 \times R_1 \equiv m^{2^{i+1}} \pmod{n}$

- **Benchmark_bignum.cpp** is used to test the speed of my bignum.h class.

All 3 files can be run from command line. The first parameter is *kbit_length* (the length of key value *d*), the second parameter is *ntest* (number of times the test will be run).

If a parameter is missing, the program will use its default value.

For example: "./test_accuracy 1024 100", "./test_accuracy 1024", "./test_accuracy" are all valid.

- To compile, see **Appendix**

III. Technical details:

1. Bignum.h:

- This class wrap GMP *mpz_t* using C++ in an object-oriented manner. The user uses this class the same way *int* is used. Memory allocation/deallocation is done automatically.

- In benchmark, when calculating numbers with 8192 bits, *bignum.h* is **less than 1% slower** than *mpz_t.*

- The performance penalty is tiny compared to the advantage this class brings.

2. lib.h:

- This header file implements multiple useful function such as *sqrt(), powermod(), modularInverse(), …*

- All functions work with type *bignum*

- GMP already implements everything, but for some functions I provide my own implementation as a practice.

3. rander.h:

- This class is a wrapper for *gmp_randstate, mpz_urandomm, mpz_urandomb*.

- Like *bignum.h*, all memory allocation/deallocation is automatic.

4. **Exponentiation algorithms:**

- Thanks to class *bignum.h,* implementing these algorithms become extremely easy. I only need to copy the exact formula as in the slide. I also place comments in the code where it is necessary.

IV. Result:

- Using **test_accuracy.cpp** and **test_properties.cpp,** we can see that the implementation is correct. The square/multiplication counts also match with their theoretical values.

```
98% completed
100% completed

Accuracy is good
Register relations are good

Average number of square/multiply for 1024 bits private key:
Algorithm                                  Square count        Multiplication count

Left-Right Square Multiply                 1024                512.67
Right-left Square Multiply                 1024                512.67
Left-Right Signed Digit Square Multiply    1024.68             343.66
Left-Right Square Multiply Always          1024                1024
Right-Left Square Multiply Always          1024                1024
Montgomery Ladder                          1024                1024
Joye Ladder                                1024                1024
```

- We can also see that the speed difference between *bignum.h* and raw *mpz_t* is very little using **benchmark_bignum.cpp** (*ntest = 100, kbit_length = 8192)*

```
Start Generic calculation test:
10% completed
20% completed
30% completed
40% completed
50% completed
60% completed
70% completed
80% completed
90% completed
100% completed
Test Generic Speed success
Time using my bignum.h          : 41.247
Time using gmp mpz_t            : 40.99
```

V. Conclusion:

In this project, I have successfully implemented all 7 modular exponentiation methods provided in the lecture as well as testing their consistency relations using GMP and my own C++ Bignum library.

Appendix

I. Scripts for testing the programs:

```
g++ benchmark_bignum.cpp -lgmp -std=c++11 -O2 -o benchmark_bignum

g++ test_accuracy.cpp -lgmp -std=c++11 -O2 -o test_accuracy

g++ test_properties.cpp -lgmp -std=c++11 -O2 -o test_properties


./benchmark_bignum

./test_accuracy

./test_properties
```