**Second Lab Assignment Report: Finite Fields and Number Theoretic Reference Problems**

*Written By: Benedek Király*

*(2021/11/02)*

1. **Installation of libsnark library**

The task was done on Ubuntu version 20.04.3 via Oracle VMBox solution. The log and output file copied from the terminal can be found under the name *Zsnark\_install\_output.* While I got errors installing libsnark, the assignment didn’t include any application of it, I only used GMP for the solutions. All executions are rerunnable and the configured environment ISO remained available for later clarifications.

1. **Warm up with small integers**

**Exercise 2.1.: Extended Euclidean algorithm**

File can be found in the *2\_1\_algos\_with\_small\_integers* named *egcd.c*

For the first exercise I created a struct class (gcdExtended) that includes the two outputs as well as the result of the GCD. I also separated to the calculation logic to a function so it can be called later on in the next exercise.

**Exercise 2.2.: Computing multiplicative inverse**

File can be found in the *2\_1\_algos\_with\_small\_integers* named *egcd\_multinv.c*

The exercise is based on exercise 2.1, adding the multInv function for calculating multiplicative inverse.

**Exercise 2.3.: Repeated square-and-multiply algorithm for exponentiation**

File can be found in the *2\_1\_algos\_with\_small\_integers* named *rsm.c*

For the RSM algorithm, I fixed the set of {k…i} as an int array and added fix values for representation (see in the Main function). The algorithm is written in the RSM function, getting a, the array, the length of array(for practicality) and n as an input and gives back the calculated result.

1. **Handle big integers using the GMP library**

For the thirds exercise, recreation of the three previous algorithms using GMP mpz\_t I replaced the logic of the exercise with GMP’s own syntax. The basic use of it is first a declaration is needed for all numeric variables needed with the call mpz\_t, then an initialization with mpz\_init or an initialization and assignment with the mpz\_init\_set\_ui call. The GMP librarys own datatypes are not compatible with C’s basic datatypes, so casting all ints to an mpz\_t object was necessary. To do arithmetic operations with mpz\_t types, the fdiv (division), mul(multiplication) and sub(substraction) was needed. While currently the solutions are tested with smaller numbers, the GMP library promotes the calculation with large numerics.

Files can be found in the folder *algs\_with\_gm:.*

2.1: egcd\_gmp.c

2.2: egcd\_multinv\_gmp.c

2.1: rsm\_gmp.c

1. **The Discrete Logarithm problem**

Files can be found in the folder *4\_th\_exercise:.*

4.1: exhaustive\_dlog.c

4.2:babystep\_giantstep.c

For exhaustive search I used the RSM algorithm I wrote previously and built the logic based on that.

For baby-step gigant-step I used two arrays instead of a table to store the pairs, then searched for values in both based on the index of elements. As in C arrays must be fixed length, I set the value 100000 for representation. This could be further developed. Input variables can be set in the beginning of the Main method. Probably storing J in a table or dictionary would be superior solution, but based on time constraints and lack of experience in C, I rather sticked to the solution with arrays.

1. **The Diffie-Hellman Key Exchange protocol**

The implementation can be found in the folder *5\_th\_exercise* as *diffie\_helman.c.*

For the algorithm I created a struct class that includes the pair of x and gx for easier handling. As stated in the task, the two functions created are KeyGen, responsible for generating the keys for Bob and Alice to share with each other. The function generates a random int and puts g on the power of it too. After changing the generated keys, the KeyAgree function can decode g by multiplying the two messages.

s