Assignment 5 WRITEUP.pdf

Roman Luo

February 26, 2023

1 numtheory.c

1.1 What I have learned:

- **GMP Library:** GMP, GNU Multiple Precision Arithmetic Library, provides a wide range of functions for performing arithmetic operations on large numbers, such as addition, subtraction, multiplication, division, modular arithmetic, prime number testing, etc. GMP is widely used in applications that require high-precision arithmetic, such as cryptography.
- **Initializing/Clearing GMP:** Initializing GMP is done using a GMP function like `mpz_init()` for a single variable or `mpz_inits()` for multiple variables. Clearing GMP is typically done using a GMP function like `mpz_clear()` for a single variable or `mpz_clears()` for multiple variables. Note that the NULL pointer is required for `mpz_inits()` and `mpz_clears()` and serves as a marker to indicate the end of the list of variables, ensuring that the GMP library can safely initialize or clear the variables without causing memory errors.
- **Initialization and Assignment Functions:**
  - `void mpz_set(mpz_t rop, const mpz_t op)`: sets the value of `rop` to the value of `op`.
  - `void mpz_set_ui(mpz_t rop, unsigned long int op)`: sets the value of `rop` to the unsigned long integer `op`.
- **Addition and Subtraction Functions:**
  - `void mpz_add(mpz_t rop, const mpz_t op1, const mpz_t op2)`: sets `rop` to the sum of `op1` and `op2`.
  - `void mpz_sub(mpz_t rop, const mpz_t op1, const mpz_t op2)`: sets `rop` to the difference of `op1` and `op2`.
  - `void mpz_add_ui(mpz_t rop, const mpz_t op1, unsigned long int op2)`: sets `rop` to the sum of `op1` and the unsigned long integer `op2`.
  - `void mpz_sub_ui(mpz_t rop, const mpz_t op1, unsigned long int op2)`: sets `rop` to the difference of `op1` and the unsigned long integer `op2`.
- **Modular Arithmetic Functions:**
  - `void mpz_mod(mpz_t rop, const mpz_t op1, const mpz_t op2)`: sets `rop` to the remainder of dividing `op1` by `op2`.
- **Multiplication and Division Functions:**
  - `void mpz_mul(mpz_t rop, const mpz_t op1, const mpz_t op2)`: sets `rop` to the product of `op1` and `op2`.
  - `void mpz_fdiv_q(mpz_t q, const mpz_t n, const mpz_t d)`: sets `q` to the quotient of dividing `n` by `d`, using floor division.
  - `void mpz_mul_ui(mpz_t rop, const mpz_t op1, unsigned long int op2)`: sets `rop` to the product of `op1` and the unsigned long integer `op2`.

- `void mpz_fdiv_q_ui(mpz_t q, const mpz_t n, unsigned long int d)`: sets `q` to the quotient of dividing `n` by the unsigned long integer `d`, using floor division.

- **Random Number Functions:**
  - `void mpz_urandomm(mpz_t rop, gmp_randstate_t state, const mpz_t n)`: Sets the value of the GMP integer variable `rop` to a random integer less than `n`, using the GMP random state `state`.
  - `void mpz_urandomb(mpz_t rop, gmp_randstate_t state, mp_bitcnt_t n)`: Sets the value of the GMP integer variable `rop` to a random integer with `n` bits, using the GMP random state `state`.

- **Comparison Functions:**
  - `int mpz_cmp(const mpz_t op1, const mpz_t op2)`: compares the values of `op1` and `op2`. Returns a positive value if `op1` is greater, a negative value if `op2` is greater, and 0 if they are equal.
  - `int mpz_cmp_ui(const mpz_t op1, unsigned long int op2)`: compares the value of `op1` to the unsigned long

- **Miscellaneous Functions:**
  - `size_t mpz_sizeinbase(const mpz_t op, int base)`: Returns the size (in bytes) of a string representation of the GMP integer variable `op` in the given `base` (i.e., the number of digits required to represent the number in that base). This function can be used to determine the size of a buffer needed to hold the string representation of an integer in a particular base before calling `mpz_get_str()`.

2 ss.c

2.1 What I have learned:

- **Random Number in Range:** The function `random_number_btw()` calculates a random number in the range between `lower` and `upper` by first generating a random number using the `random()` function, then scaling it to the appropriate range by computing the remainder of the random number divided by the range (calculated by `upper - lower`) and adding the result to `lower`.

- **Reading/Writing from/to Files:** `int gmp_fprintf(FILE *stream, const char *format, ...);`

  `gmp_fprintf` writes formatted output to a stream specified by `stream` argument. The `format` argument specifies the format of the output, just like in `fprintf()`.

  `int gmp_fscanf(FILE *stream, const char *format, ...);`

  `gmp_fscanf` reads formatted input from a stream specified by `stream` argument. The `format` argument specifies the format of the input, just like in `fscanf()`. The difference is that the `...` argument list may contain one or more `mpz_t`, `mpq_t`, or `mpf_t` GMP data types instead of normal C types.

  For example, `gmp_fprintf(pvfile, "%ZX\n%ZX\n", pq, d)` is used in my `ss_write_priv()`, `%ZX` is a specifier for converting into hex string.

- **Checking File:** The feof function tests whether the end-of-file (EOF) indicator has been set for a given stream. The function takes a pointer to a FILE object as its argument and returns a non-zero value if the EOF indicator has been set, and zero otherwise. The syntax for feof is: `int feof(FILE *stream);` This function returns 0 if the end of file is not reached. Hence, the condition !feof(infile) is used for a while loop that keeps reading and converting data while there are still unprocessed bytes in infile.

- **Reading/Writing Bytes from/to Files:** fread and fwrite are functions in the C standard library that allow you to read from and write to files, respectively. Where ptr is a pointer to the array of elements to be read, size is the size in bytes of each element to be read, count is the number of elements to be read, and stream is the file pointer to the input stream.

```
The syntax for fread is:
size_t fread(void *ptr, size_t size, size_t count, FILE *stream);

The following read at most k1 bytes in from infile,
and let j be the number of bytes actually read.
Place the read bytes into the allocated block
starting from index 1 so as to not overwrite the 0xFF:
uint64_t j = fread(&block[1], sizeof(uint8_t), k - 1, infile);

The syntax for fwrite is:
size_t fwrite(void *ptr, size_t size, size_t count, FILE *stream);

The following Write out j - 1 bytes starting from index 1 of the block to outfile:
fwrite(&block[1], sizeof(uint8_t), j-1, outfile);
```

- **Converting Bytes into mpz:** We can use mpz import()/export(), convert the read bytes into mpz t, or the other way around. Both functions can take up to 7 arguments, including rop, count, order, size, endian, nails, limbs (optional).

```
Using mpz_import() convert the read bytes into an mpz_t m.
Store in block:
mpz_import(m, j + 1, 1, sizeof(uint8_t), 1, 0, block);

Write out j - 1 bytes starting from index 1 of the block to outfile.
fwrite(&block[1], sizeof(uint8_t), j-1, outfile);
```

3 randstate.c

3.1 What I have learned:

- **Initializing/Clear State:** We can use gmp-randinit-mt to initialize a state, then set it with a seed using gmp-randseed-ui(state, seed). To clear state, simply use gmp-randclear(state);

4 keygen, encrypt, decrypt

4.1 What I have learned:

- **File Permissions:** To set a file with 0600 permissions, first get a file descriptor by calling fileno(File Steam here); Then, set the permission by calling fchmod(file-descriptor, 0600);

5 Makefile

5.1 What I have learned:

- **Makefile that compiles multiple C files with main():** Specify all the object files that each main C files needs to run. Specify each other them with corresponding names to their C files.

```
SOURCES  = $(wildcard *.c) // all the c files
OBJECTS  = numtheory.o ss.o randstate.o

CC       = clang
CFLAGS   = -Wall -Wpedantic -Werror -Wextra -gdwarf-4
LIBFLAGS = `pkg-config --libs gmp` //gmp lib

.PHONY: all clean format

all: keygen encrypt decrypt // specify names

keygen: $(OBJECTS) keygen.o
 $(CC) -o $@ $^ $(LIBFLAGS)

encrypt: $(OBJECTS) encrypt.o
 $(CC) -o $@ $^ $(LIBFLAGS)

decrypt: $(OBJECTS) decrypt.o
 $(CC) -o $@ $^ $(LIBFLAGS)
```