Exercise Sheet Public Key Cryptography

Solve the following exercises and submit them until the communicated date.

LB-PKC 00. (not to be submitted)

(*Code::Blocks* template *GMP project*)

Using the *GMP*, write a program which multiplies two numbers passed as arguments and outputs the result. Verify your program with the example number pairs `2 3` and `x x`, where `x` is the largest possible number which can be stored in a signed 64-bit integer.

LB-PKC 01. (*Code::Blocks* template *GMP project*)

The RSA encryption c of a message m with the public key (e, N) is defined as follows: $c \equiv m^e \pmod{N}$. The decryption with the secret key (d, N) is defined analogously: $m \equiv c^d \pmod{N}$.
Using the *GMP*, write a program which implements an RSA encryption or decryption. The program is supposed to accept three arguments in the following order: the message m to be encrypted or decrypted, respectively, **as a number**, an exponent (e or d, respectively) and the modulus (N). The value to be encrypted or decrypted, respectively, is to be output (**without** additional output) as a number to `std::cout`. Use the `mpz_powm` function for your implementation and test it with $m = 7, e = 29, d = 85$ and $N = 391$.
Hint: Use the `mpz_class::get_mpz_t` method in order to pass the instances of the `mpz_class` class to the `mpz_powm` function. For details on the GMP *functions, refer to the documentation at `https://gmplib.org/manual/Function-Index.html#Function-Index`. `mpz_class` instances can be evaluated during debugging by entering `variable_name.get_str(10)` into the* Watch *window, where `variable_name` must be replaced by the variable name of the instance.*

LB-PKC 02. (*Code::Blocks* template *GMP project*)

An RSA key pair, consisting of a public key (e, N) and a secret key (d, N) can be generated as follows:

1. Choose two mutually distinct primes p and q, i.e., $p, q \in \mathbb{P}$, where $p \neq q$.
2. Compute $N = pq$.
3. Compute $\varphi(N) = (p - 1)(q - 1)$.
4. For the public key, choose an integer e between 1 and $\varphi(N)$ (excluding both limits) which is relatively prime to $\varphi(N)$, i.e., $\gcd(e, \varphi(N)) = 1$.
5. For the secret key, compute d as the inverse of e modulo $\varphi(N)$, i.e., $d \equiv e^{-1} \pmod{\varphi(N)}$.

Write a program which does **not** accept any arguments, creates an RSA key pair as specified above and outputs the public and the secret key in **exactly** the following pattern (upper case and lower case, spaces etc.) to `std::cout`:

```
Public key: (29, 391)
Private key: (85, 391)
```

Verify the keys by using them to encrypt and again decrypt a message with your program from example 01. The key length (bit length of N which is influences by the lengths of p and q) is supposed to be exactly 2,048 bits. If the key length is not exactly 2,048 bits, a new key must be generated. If necessary, generate new keys until the length is achieved exactly.
Hint: Use the two functions `gmp_randinit_default` and `gmp_randseed_ui` to initialize a random number generator at the start of your program. Using this generator and the `mpz_urandomb` function, generate a random number and apply `mpz_nextprime` in order to obtain a prime. For determining the length, use the `mpz_sizeinbase` function.
For the choice of e, iterate through the specified value range until you have found a value which satisfies the specified criterion. For computing the greatest commmon divisor, use the `mpz_gcd` function. For computing the inverse in the modulus, use `mpz_invert`.
Wherever possible, use the overloaded operators, e.g., `+`, `<`, `==` etc. for the corresponding arithmetic and comparison operations.