CAA 24-25

Exercise Sheet on Randomness

1 Code Review: Randomness Generation

The file `prng.py` contains an implementation of the Blum-Micali PRNG.

1. Execute it few times. You should notice at least two weird behaviours.
2. Blum-Micali is provably secure. What is the mistake in the code (warning: some maths might be required here).
3. Fix the code.

2 RDSEED

Write a C program that makes a call to RDSEED in order to obtain a TRNG-generated random 32-bit value. Be extra cautious in case RDSEED fails.[1] You can also implement this in another programing language if you want.

3 Implement Diffie-Hellman using OpenSSL (optional)

The goal of this exercise is to implement fully, in C, the Diffie-Hellman protocol using the OpenSSL library. For this, you need to install the openssl library (libssl-dev package). You will find a template of the code in the file `dh.c`. To simulate the network communication we will simply store the sent message (called a "public key" in the OpenSSL language) into a file using the PEM format.

Few useful points:

- You will almost only use the "high-level" function of openSSL which are called "envelop" functions. They start with `EVP`.
- Don't forget to check the error status of every command. You can use the provided `handleErrors()` function to print error message.
- The documentation is not great but exists online and in `man`.
- For key generation, look at `EVP_PKEY_keygen()`.

[1] You will find useful documentation here: `https://www.intel.com/content/www/us/en/developer/articles/guide/intel-digital-random-number-generator-drng-software-implementation-guide.html`

- For key derivation, look at `EVP_PKEY_derive()`.
- For handling PEM formats, look at `PEM_read_PUBKEY()` and `PEM_write_PUBKEY()`.
- To compile: `gcc dh.c -lcrypto -o dh`