Embedded Systems Security'18
Laboratory assignment 2 (must complete)
weeks of Feb. 26 and Mar. 5

During this class you will build a VHDL model of a bit-sequence generator similar to that of A5/1. Using the model you will be able to run some randomness tests to see if you can draw any conclusions.

The VHDL code you will get contains both an exemplary implementation of a LFSR – *linear-feedback shift register* and a testbench, that instatiates two such registers, loads one of them with an IV (*initialization vector*), and runs them both, producing two sequences of bits. This should give you enough insight into the code to allow further extending it into a more useful and closer to A5/1 model.

Assignment 1 Download code available for this class. Read it carefully *before* running any `ghdl` instance! Go through both architectures of *lfsr* entity, sketch them out on paper. Mark other signals from the `lfsr_tb.vhd`: *load, q1, q2, LFSR1, LFSR2.*

Assignment 2 Run code for the testbench a number of times, changing the IV of both UUT's. Observe how (if) the changes you introduce change the output.

Assignment 3 Modify the code so that the LFRS's reflect those found in A5/1 (change the lengths and tap positions). Connect outputs of these LFSR's to a three-input XOR to produce another signal, say *RND*. Using `std.textio` library from previous class print out to console the value of *RND* after each clock cycle. Alterately, you can write this directly to a file, see `https://www.nandland.com/vhdl/examples/example-file-io.html` for an example.

Assignment 4 Using any programming language write a tool to determine the randomness properties of the sequence of *RND* bits you produced. Choose 3-4 tests from *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* by NIST (available online).

Assignment 5 Using the tool, try to find such IV's that result in poorest randomness for the analysed LFSR configuration.

Note. For Assignment 3. you can try to reproduce a real A5/1 generator: you must then introduce the majority voting entity and use it to drive the clock for respective LFSR's. You will also have to modify the LFSR's design to output the clock-decision bits to the voting entity.