ECE 4156/6156 Hardware-Oriented Security and Trust

Spring 2025

Assoc. Prof. Vincent John Mooney III

Georgia Institute of Technology

Lab 4, 100 pts.

Due Tuesday, April 22 prior to 11:55pm

(please turn in homework electronically on Canvas)

In this lab, you will compile and use the NIST test suite. Then, you will generate 32 supposedly pseudorandom bitstreams using VHDL code for the encryption scheme from the previous lab assignment. Finally, you will carry out cryptanalysis on the 32 bitstreams and answer some questions. **The instructions given here regarding the NIST test suite compilation assume you are using a Linux platform. It is recommended to either use a virtual machine running some kind of Linux distro (e.g., Ubuntu, Debian) or to ensure your system has GCC and make installed to be able to compile the NIST test suite.**

I. Setup

Download the lab4.zip file and place the file in a new directory. Unzip the archive.

1. You should see the directories “VHDL” and “nist_sts.”
2. In “nist_sts,” you should see 6 other directories and a makefile. Navigate to this directory in the terminal and type "make", which will compile the test suite (assuming you have make and GCC installed on your system).
3. You should see an executable file called "assess" created in your current directory. Generally, you can run the file called "assess" by typing "./assess <stream length>"
4. Stream length is the number of bits you want to use per test. **For example**, if your random number generator generates 128 bits as an output, your stream length is 128, so you would type "./assess 128" in the terminal.

II. NIST test suite documentation

The official NIST test suite documentation can be found here (https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf).

III. Generating Test Data

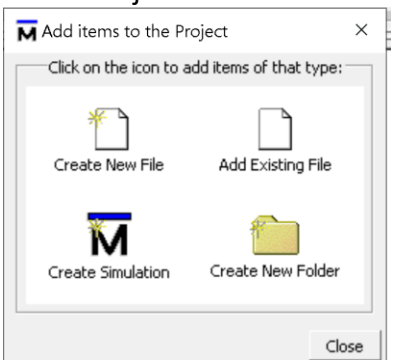
To run the test suite, you need some pseudorandom data to test. In this section, we will use the VHDL for the encryption scheme from the previous lab assignment as the generator of our pseudorandom data.

This encryption scheme is a stream cipher, meaning that it can be used solely for pseudorandom number generation purposes. For this particular design, the inputs are an 84-bit key and an 84-bit IV. The IVs are contained in input.txt, and the key does not change (the key is defined in the testbench).

In particular, we will generate 32 bitstreams, each of length 20,000 bits.

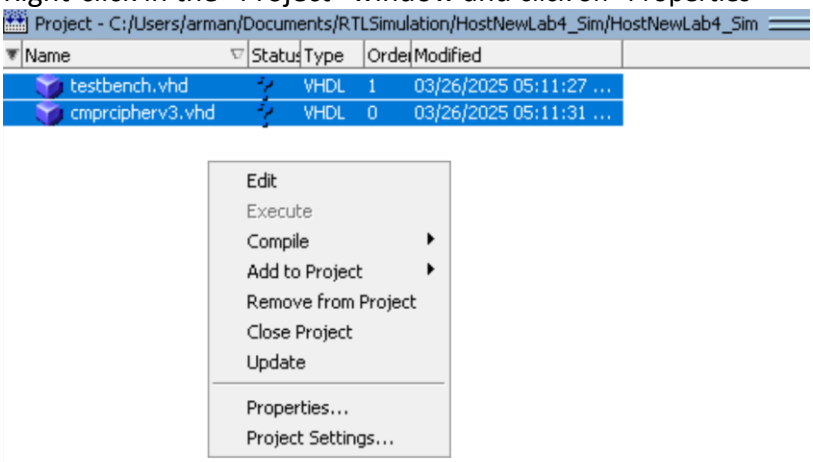
The bitstreams will be generated using ModelSim VHDL simulation.

1. Create a new directory for your ModelSim Project. This directory will initially be empty, but **paste the file "input.txt" file from the "VHDL" directory of lab4.zip into this newly-created directory.**
2. Open up ModelSim and create a new project (File -> New -> Project)
3. Fill the "Project Name" field with a project name of your choice
4. For the "Project Location" field, navigate to the directory you created in step 1 of this section
5. Leave everything else as-is and press "OK." ModelSim will now open up the "Add items to the Project" GUI.

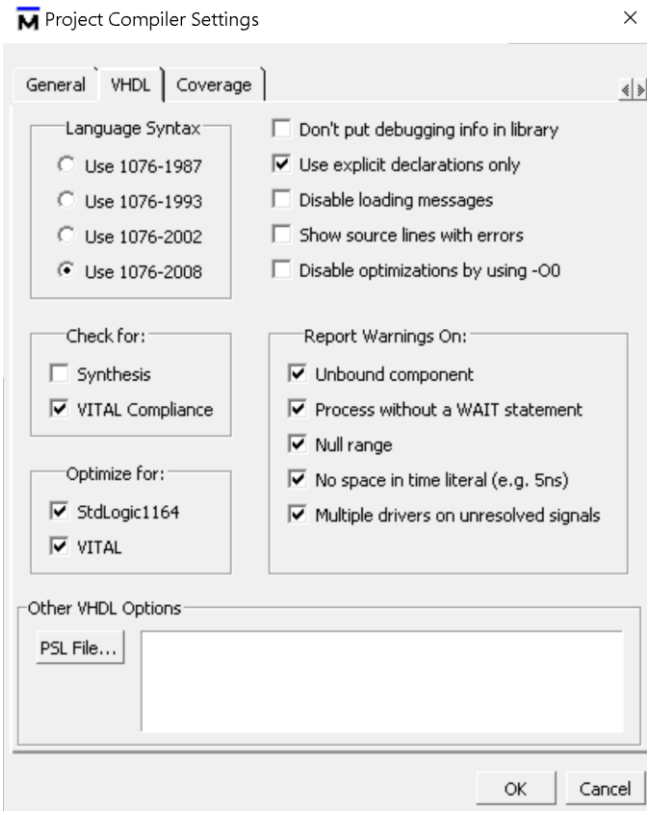


6. Click on "Add Existing File" and add the two VHDL files in the "VHDL" directory of lab4.zip (cmprcipherv3.vhd, testbench.vhd). Once you have added the VHDL files to the ModelSim project, you may click "Close" on the "Add Items to the Project" GUI.

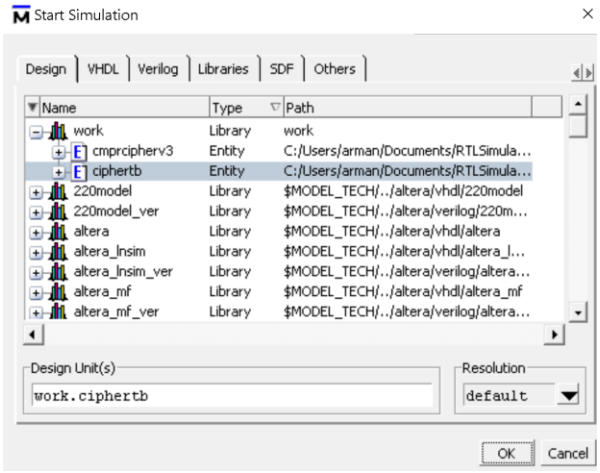
7. Highlight the two VHDL files in the ModelSim “Project” window, either by using ctrl + click on each file or by using ctrl + a.
8. Right-click in the “Project” window and click on “Properties”



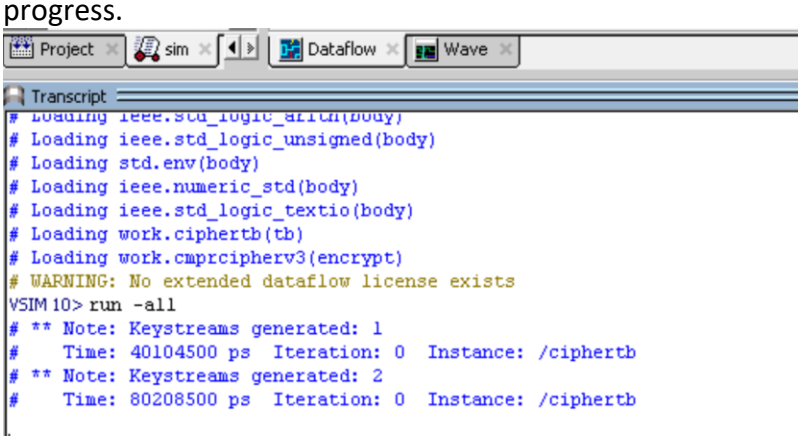
9. You should see the “Project Compiler Settings” GUI. Click on the “VHDL” tab, and under the “Language Syntax” section, select “1076-2008.” Then, click “OK” to close out of the compiler settings GUI.



10. Now, compile the VHDL using “Compile -> Compile All.” The VHDL files that have been provided should compile as-is. **Some users on different versions of ModelSim may need to edit testbench.vhd to include an explicit (complete) path to input.txt.**
11. Start a simulation using “Simulate -> Start Simulation.” This will open up the “Start Simulation” GUI.



12. Click on "+" next to "work." Under "work," select "ciphertb" and then press "OK."
13. You will not need to examine any waveforms for this simulation. In the ModelSim command line, enter the command "run -all."
14. Wait for the data to finish generating. This should take anywhere from one to five minutes depending on your computer. The ModelSim terminal will update you on the progress.



15. Once the data has finished generating, the simulation will terminate and you should see the file "output.txt" in the ModelSim project directory. **Hold on to this file as you will need to use it in the next section and will also need to turn it in.**

IV. Running the NIST Test Suite on Generated Data

The instructions for this section assume that you have completed Section I and have generated the “assess” executable for running the NIST tests.

Before continuing with the steps below, ensure that “output.txt” from the prior section is in the same folder as your “assess” executable.

1. For the following instructions, only type what is within the quotation marks (“”)
2. type "./assess 20000" in the terminal
3. When the "Generator Selection" prompt appears, type "0" and press Enter.
4. When the "User Prescribed Input File:" prompt appears, type "output.txt" in the terminal.
5. When the "Statistical Tests" prompt appears, type “0” and press Enter.
6. When the prompt shows "123456789111111" followed by "012345", type "111111111000011" in the terminal and press Enter. Note that by typing "111111111000011", **we are running all tests except for Universal, Approximate Entropy, Random Excursions, and Random Excursions Variant.** These four tests require significantly more data to run, whereas we only have 640,000 bits.
7. When the "Parameters Adjustments" prompt appears, type "0" and press Enter to continue, leaving the parameters as their default values.
8. When the "How many bitstreams?" prompt appears, type "32" and press Enter to continue
9. When the "Input File Format:" prompt appears, type "0" and press Enter to continue
10. You should see "Statistical Testing Complete" after a few moments.
11. To view the test results, enter the “experiments” directory (in the same directory as the “assess” executable), enter the “AlgorithmTesting” directory, and open the file “finalAnalysisReport.txt.” **Make sure to hold on to this text file as you will need to turn it in and answer some questions based on it.**

V. Reading the NIST test results

Please read Section 1.1.5 (page 14 - 16) and Section 5.7 (page 101) of the official NIST test suite documentation to **answer the following questions for your lab report.**

1. What does it mean when P-value is 1?
2. What does it mean when P-value is 0?
3. What is a significance level?
4. What does the "proportion" column mean in finalAnalysisReport.txt?

VI. Proving the Generated Data is not Pseudorandom

Based on the results of the NIST tests, the generated data is pseudorandom. However, relying only on statistical testing to ascertain the quality of a pseudorandom number generator is a mistake. Further analysis is needed.

1. Observe that “output.txt” has 32 lines, where each line has a bitstream of length 20,000 bits. **Using a programming language of your choice**, write code that computes the bitwise exclusive or (XOR)* of the 32 20,000-bit bitstreams, and determines the number of 0’s and 1’s in the 20,000-bit result. If the 32 bitstreams are pseudorandom, we would expect their bitwise XOR to also be pseudorandom.
2. **Turn in the code from step 1, and in your lab report, include the number of 0’s and 1’s in the result of the bitwise XOR of the 32 bitstreams.**

*By bitwise XOR, we mean that the vectors are XORed bit-by-bit. For example, the bitwise XOR of 110 and 101 is 011.

For those interested, the cryptanalytic attack you just carried out is called a cube tester. It is a distinguishing attack, which *distinguishes* the output of a cryptographic scheme from a truly random sequence.

The punchline of this assignment is that statistical tests can be deceptive. Mathematical vulnerabilities cannot always be detected through statistical analysis.

IX. What to Turn In

Please type all your answers and include a cover sheet with your first and last name, GT ID number, and GT username.

1. Turn in output.txt from Section III
2. Turn in finalAnalysisReport.txt from IV.
3. Answer the questions from Section V.
4. Turn in the code from Section VI, and specify how many 0's and 1's are in the result of the bitwise XOR.

Submit Lab 4 on Canvas.