236237 - Practical assignment 2

In this assignment, you will perform a simple reverse engineering task using the HAL infrastructure.

Preparations

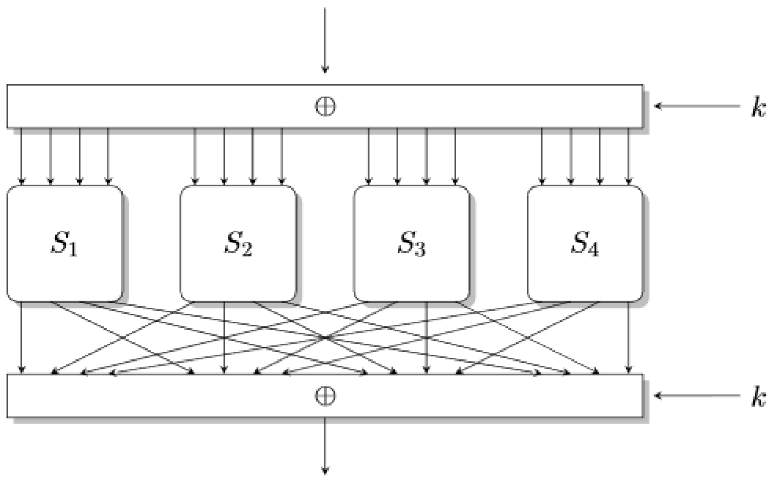
Install HAL -- follow instructions here: https://github.com/emsec/hal

HAL Python documentation: https://emsec.github.io/hal/pydoc/index.html

Introduction

In the following you are provided the netlist of an unknown cryptographic implementation by *Yet Another Company*. Their *ToyCipher* disobeys Kerckhoffs' principle by using its own proprietary algorithms. A colleague of yours has already analyzed the datapath of the implementation and created a block diagram showing one round of the ToyCipher (see below).

To gain a better understanding of the entire chip functionality, your task is to have a look at the control path, i.e., the FSM controlling the ToyCipher.



You will analyze two versions of the chip. First, you will look at an older version of the chip, that has no additional security features. In a more recent version, which you will analyze in the second part, the company implemented an FSM obfuscation scheme, but left the datapath unchanged. Your goal is to break the FSM obfuscation scheme and extract the enabling key.

FSM Detection and Reverse Engineering

In this task, you will reverse engineer the netlist *project2_cipher_v1.v*. Create a Python script, which you will extend in a step-by-step manner throughout the following subtasks.

This script should be generic enough to work with both this netlist and the obfuscated FSM netlist in the next section.

1. Determine possible FSM candidates using the methods introduced in the lecture. You will find candidates for the FSM using the `strongly connected component` algorithm in the `graph_algorithm` plugin. In case you retrieve more than one FSM candidate, your script should automatically decide which candidate is the actual FSM. However, it suffices if your implementation works for both netlists in this project.
2. Create a module that includes all (combinational and sequential) gates which realize the FSM (output logic is not required).
3. Now create two submodules — one that contains the sequential gates and another one, which contains all combinational gates of the FSM.
4. Determine the Boolean function for each bit of the state transition logic, i.e., the Boolean functions of the inputs for each flip-flop of the state register. Include the Boolean functions in your report. Implement this as generically as possible. The flip-flops implementing the state register may have several logic gates in front of them. Therefore, use the `get_subgraph_function()` function from the `netlist_utils` class that can combine several logic gates.
5. You now have to brute-force all reachable states **starting from the initial state (f**or simplicity, you can assume that all flip-flops are initialized with 0). Print the resulting state transition table and include it in your report. **Hint**: Use the Boolean equations and the `evaluate` function.
6. Within your script automatically create a `.dot` file that can be used to visualize the state graph. `.dot` files are often used for depicting graphs. Use Graphviz to create the graphical representation of your state transition graph. For more information on GraphViz and `.dot` files, see *https://en.wikipedia.org/wiki/DOT_(graph_description_language)*. GraphViz is already installed in your Virtual Machine. To generate a `.png` from your `.dot` file, use the following command: `dot -Tpng graph.dot > graph.png` . Insert the resulting figure in your report.
7. What, in your opinion, is the functionality of this FSM?

Analysis of Obfuscated FSM

In this task, you will deal with a newer version of the ToyCipher circuit whose Finite State Machine (FSM) has been obfuscated with the HARPOON scheme presented in the lecture. For this project, you need the netlist `project2_cipher_v2_obfuscated.v`, which contains the obfuscated FSM.

Yet Another Company improved and modified their chip design to protect against overproduction of their innovative proprietary cipher. With the FSM obfuscation in place, a certain enabling key has to be entered in order to reach the initial state of the original FSM every time the chip is powered on. You will find additional input signals that control the FSM in the netlist.

1. Execute the script you created in the previous task with the new netlist. Analyze the generated state transition graph (which you create with GraphViz from the `.dot` file output by your script). How many states does the FSM have in total? Which states

belong to the obfuscated part of the FSM and which belong to the original FSM? Mark them in the state transition graph and include the complete state transition graph of the FSM in your report. **Note:** The script you have created for the original FSM has to work for both netlists.
2. Determine a correct enabling key that has to be entered to reach the initial state of the original FSM. In your report, provide the enabling key.

Submission Guidelines

Submission deadline: as appears at the course site

Submission in couples.

Submission contents:

1. Report
2. Python script(s)

`Questions:`

In the class or by email: `leonida AT technion.ac.il`

Bibliography

- Fyrbiak et al. HAL — The Missing Piece of the Puzzle for Hardware Reverse Engineering, Trojan Detection and Insertion. IEEE Transactions on Dependable and Secure Computing (2019). URL: `https://eprint.iacr.org/2017/783.pdf`
- Shi et al.A highly efficient method for extracting FSMs from flattened gate-level netlist. 2010IEEE International Symposium on Circuits and Systems (ISCAS). URL: `https://ieeexplore.ieee.org/document/5537093`1
- Meade et al. Gate-level netlist reverse engineering for hardware security: Control logic register identification. 2016 IEEE International Symposium on Circuits and Systems (ISCAS).URL: `https://ieeexplore.ieee.org/document/7527495`
- Chakraborty et al. HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2009). URL: `https://ieeexplore.ieee.org/document/5247148`
- Fyrbiak et al. On the Difficulty of FSM-based Hardware Obfuscation. Conference on Cryptographic Hardware and Embedded Systems (CHES) 2018. URL: `https://tches.iacr.org/index.php/TCHES/article/view/7277`