Experiment 16
Oracle-Based Attacks on Logic Locking
on HaHa v3.0 Board

We describe an experiment to attack some logic locking techniques using oracle-based attacks, such as Brute Force and SAT solvers.

Instructor: Dr. Swarup Bhunia
Co-Instructors/TAs: Reiner Dizon-Paradis, Pravin Gaikwad, & Shuo Yang

Theory Background

Logic locking helps hide the original logic of the IP as seen in previous experiments. It does this by locking the correct functionality of a chip, which can be unlocked using a key or sequence of keys. In order to perform this countermeasure, additional logic is usually added to the original, unlocked design. However, this protection is also prone to attacks, which is aimed to retrieve the key which unlocks the correct functionality of the IP. Consequently, there are a few attacks on hardware obfuscation and logic locking. An attacker can reverse engineer a locked design, so it can be used to compare against a functional chip (called an oracle). This kind of attack on logic locking is called *oracle-guided* or *oracle-based attacks* in the literature. In this experiment, we will explore two kinds of oracle-based attacks: brute force and Boolean satisfiability (SAT).

1. Brute Force Attacks

Brute force attacks on logic locking aims to try all possible key combinations and monitor the output signals. Attackers are checking for the output from the locked design that matches those of the oracle or golden reference. For this attack, the best-case scenario is when the attacker has to apply 2^N key patterns for a N bit lock. This attack might be effective against smaller key sizes, but for a large key size like 128-bit, this attack does not scale appropriately as to be seen in this experiment.

2. SAT-based Attacks

The SAT-based attack models the locked or obfuscated design as a Boolean satisfiability problem, which can use heuristic-based algorithm to solve it, often called SAT solvers. Just like brute force attacks, the SAT attack requires an oracle or unlocked chip that can produce correct functional outputs. The aim of this attack is to shrink the search space for the key by using randomly selected distinguishing input patterns (DIPs). A DIP is pattern of input signals where at least two key values produce different output values. The best scenario occurs when a DIP can identify multiple incorrect key values. A poorly designed logic locking scheme can be unlocked by SAT attack using few DIPs.

Experiment Set-up: Configuration

The hardware and software needed for this experiment include:

1. The HaHa v3.0 Board.
2. ECE Linux Server
   a) Design Compiler
3. Ubuntu OS (virtual machine, dual boot, or primary OS)
   a) PSAT
   b) Yosys
   c) ABC Compiler
   d) Python 3

Instructions and Questions

PART I: Brute Force Attack on HaHa v3.0 Board

1. Download the zip file marked as your group number. You will find a FS file which is designated only for your group.
2. Load the FS file to your HaHa V3.0 board with GOWIN Programmer.
3. Connect the Slide Switch to the 5-bit key inputs so that MSB of key is assigned to Slide Switch [4] and LSB is assigned to Slide Switch [0]. Connect reset to Slide Switch [9], which resets the value displayed in the 7-segment display and LEDs.
4. Manually apply all key patterns. With each pattern, reset and see if the circuit is unlocked or not. For correct key, the 7-segment display and LEDs will show a hex counter. Note: You don't need to apply reset every time you change the switch values.
5. Demonstrate the working key and record a short video of the demonstration. Submit the detected key in report and the demonstration video. See the instruction for demonstration part for further information.
6. Take in image of the board showing the key pattern in switch box and any working digit on display. Submit the image in report.
7. Observe how long it took (approximately) to find a 5-bit key. Calculate how long will you take to unlock a 256-bit key (x 2^{256} / 2^{6}). Is it possible to break a 256-bit obfuscation manually with brute force attack? Submit your answer in report.

PART II: Automating Brute-force attack with testbench:

You will need to design a testbench that can apply all possible key patterns on an obfuscated circuit and compares the outputs with the original circuit to check if both matches implying the obfuscation is broken.

Design a testbench like Figure 1 –

- with the combinational obfuscated c499 you generated in Part I of Experiment 6, and
- the provided unlocked c499 circuit in Part I of Experiment 6.

Figure 6 shows only one output. You will need to XOR each output of original circuit with corresponding output of obfuscated circuit. Whenever the output pair of an XOR matches, it would produce a 0. For correct key, all XOR output will be 0. You need to detect this condition. You can do that with a NOR gate with all these comparison XOR outputs. You can try a few inputs to compare the outputs to make sure obfuscation is broken.

In your testbench, using $fwrite, write the keys and the output of that NOR on a text file. Analyzing this file, you can see which one is the unlocking key for your obfuscation.

1. How long does it take to run the simulation?
2. Is there more than one key that could work as unlocking key?
3. Submit the testbench along with the report.

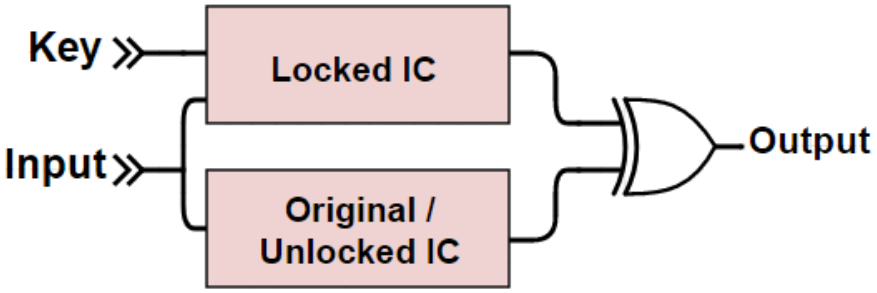

Figure 1: Automating Brute-force attack

PART III: Set up and launch SAT attack

In this part of the experiment, you'll need an Ubuntu OS (virtual machine, dual boot, or primary OS) as well as access to the Linux ECE server. You'll also need to download provided files from Canvas and place them in different machines. Also, you'll need the original c499 design and XOR/XNOR-locked design from Experiment 6. The locked design should have 16 key inputs (`keyinput0`, `keyinput1`, … , `keyinput15`). You will compare the performance of the brute-force attack done in the previous steps compared to the SAT attack.

Step I: Setting up Ubuntu OS if you don't already have one

1. If you don't have an Ubuntu operating system, create a virtual machine with at least Ubuntu 20.04 LTS using Virtualbox VM.
    a. Alternatively, you can import the provided OVA file that contains all the necessary software into your Virtualbox VM. When you import, make sure to disable the USB Controller. Otherwise, you will get an error.
        i. Password: **pass123!**

   b. If you choose this method or have Ubuntu installed as primary or dual boot, skip to Step II.
2. Download the provided files: **install_psat.sh**, **get_bench.py**, **sat_run.py**, and **sky130.lib**.
3. Place all these files in the home directory of your Ubuntu OS.
4. Open a Terminal window and run the following commands:

```
chmod +x ./install_psat.sh

./install_psat.sh
```

Note: You will need to type in your password before the programs are installed in your Ubuntu OS.

Step II: Converting your designs to a target library using Design Compiler on the ECE Linux Server

1. Download the provided file: **convertToSkywater.zip**
2. Place this file on your account on the ECE Linux Server and unzip it to a folder **convertToSkywater/**
3. Inside this folder, also place the original and locked c499 designs from Experiment 6.
4. Open the file called **convertToSkywater.tcl** and pay attention to the two "set" commands at top of the tcl script. Make the following changes and save the script.
   a. Set *design_name* to *c499*
   b. Set *locked* to *false*
5. In the terminal, run the following commands:

```
source /apps/settings

dc_shell -f convertToSkywater.tcl
```

Note: This should give you the original design mapped to Skywater technology library called **c499_sky130.v**

6. Open **convertToSkywater.tcl** and change the value of *locked* to *true*.
7. Repeat the whole `dc_shell` command from Step 5 (2nd line). This time, it will provide you the locked design mapped to Skywater technology library called **c499_lock_sky130.v**
8. Download **c499_sky130.v** and **c499_lock_sky130.v** to your local machine and place them in your Ubuntu OS.

Step III: Launching SAT Attack

1. You should have the **get_bench.py**, **sat_run.py**, and **sky130.lib** files on your Ubuntu machine from Step I.
2. In a terminal window, run the following command and keep in mind what parameters it requires:

```
python3 sat_run.py
```

3. If you have followed all the steps previously, you can run the following command (keep in mind the file locations):

```
python3 sat_run.py -opdir ./c499/ -lib_orig ./sky130.lib -lib_locked ./sky130.lib \
    -orig_design ./c499_sky130.v -locked_design ./c499_lock_sky130.v -top c499
```

Note: In this run, all the outputs (designated by the `-opdir` flag) will be inside the `c499/` directory.

4. Navigate to the `c499/` directory. Open `sat.log` file. You should see the key derived from the SAT attack near the bottom of the log file.

```
Test coverage rate: 100 %
Output error rate (inverse of coverage rate): 0 %
Average Hamming distance: 0 %
 These metrics DON'T cover the sampling of I/O patterns to mitigate the
 stochastic behaviour of the circuit, i.e., testing considers any output
 pattern as is, even if they are erroneous.
key=1001
```

Figure 2: SAT Attack Result of an example run (bottom of the log file). In this case, the 4-bit key is 1001.

5. Take a screenshot of your SAT attack log similar to the last step and add it to your report.
6. Compare this key with your expected key bits from Experiment 6.
7. Compare the speed of this approach to the Brute Force attack.

Step IV: Extend your key inputs to 24 bits and run SAT attack

1. Add 8 additional lock gates to your c499 locked design.
2. Run Steps I to III again.
3. Take a screenshot of your original 24-bit key.
4. Take a screenshot of your SAT attack log.

PART IV (Optional): Launch Brute Force and SAT attack on a design with varying key sizes and perform runtime analysis

We have provided the original design for c7552 that is already mapped to Skywater technology library as well as 5 locked designs of varying key sizes from 16 bit to 256 bit.

1. Using instructions from Part II, run the Brute Force attack against each locked design. Make sure to note how long each attack take per key size. Create a graph that shows the trend of key size vs. run time.
2. Using instructions from Part III, run the SAT attack against each locked design with references to the original design and the Skywater technology library. Make sure to note how long each attack take per key size. Create a graph that shows the trend of key size vs. run time.

Note: If some of the attacks are taking a while, you can stop the run time after at least an hour. To get full credit for this part, you must run your attacks for at least 1 hour. If you are only able to do one of the run time analyses, you will get half the extra credit points.

This optional follow up is worth 20%.

☞ Lab Report Guidelines

Deliverables:

1. For Part I:
    a. Report the correct unlocking key (in report)
    b. Image of the board showing the key pattern in switch box and a working digit on display (in report)
    c. Calculation for question 7 (in report)
    d. Demonstration video (see instruction below)
2. For Part II:
    a. How long does it take to run the simulation?
    b. Is there more than one key that could work as unlocking key?
    c. Submit the testbench along with the report.
    d. How can you protect against Brute Force attacks?
3. For Part III:
    a. How long does it take to run the SAT attack with 16 key inputs?
    b. Take a screenshot of your original 16-bit key.
    c. Take a screenshot of your SAT attack log for locked design with 16 key inputs.
    d. How long does it take to run the SAT attack with 24 key inputs?
    e. Take a screenshot of your original 24-bit key.
    f. Take a screenshot of your SAT attack log for locked design with 24 key inputs.
    g. Which oracle-based attack is faster? How so, and why?
    h. How can you protect against SAT attacks?
4. For Part IV (Optional):
    a. Include the graph of Brute force run time of different key sizes.
    b. Include the graph of SAT Attack run time of different key sizes.

Demonstration:

1. For Part I, once your figure out the correct unlocking key, make a short video showing:
    a. A wrong key loaded in switch → reset → the 7-segment display is showing anything except a counter.
    b. The right key is loaded in switch →reset → the 7-segment display is showing a counter.
2. For Part II, demonstrate the automated brute force attacks and explain the results.
3. Part III, demonstrate successful SAT attacks and explain the results.

References and Further Reading

[1] P. Subramanyan, S. Ray and S. Malik, "Evaluating the security of logic encryption algorithms," 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)

[2] Forte, Domenic, Swarup Bhunia, and Mark M. Tehranipoor. "Hardware Protection through Obfuscation." (2017).

[3] Probabilistic SAT (PSAT) Attack, Design-for-Excellence Lab, NYU Abu Dhabi, Available at: https://github.com/DfX-NYUAD/PSAT

[4] ABC: A System for Sequential Synthesis and Verification, Berkeley Logic Synthesis and Verification Group, UC Berkeley, Available at: https://github.com/berkeley-abc/abc

[5] Yosys Open SYnthesis Suite, Available at: https://yosyshq.net/yosys/download.html