

Introduction

Hardware Security is a big concern nowadays due to the vulnerabilities of ICs. Many attacks threaten IP holders, these attacks are mainly aimed at two purposes:

- IP Piracy
- Information Leakage

Scenario:

A designer has developed and validated an IC and is about to send off to a foundry for production. However, he/she wants the IC to be protected from any spot in the supply chain, such that attacks to the IC will not be successful.

Attacks:

Attackers must perform certain attacks to be able to gain access to whatever they are trying to seek information from. Some types of attacks include:

- Sensitization
- Removal
- SAT

Defenses:

However, those attacks above can be prevented using certain obfuscation techniques such as logic locking. Some obfuscation techniques that are available are:

- Random Logic Locking
- Fault Logic Locking
- Strong Logic Encryption
- SARLock
- ANTISat

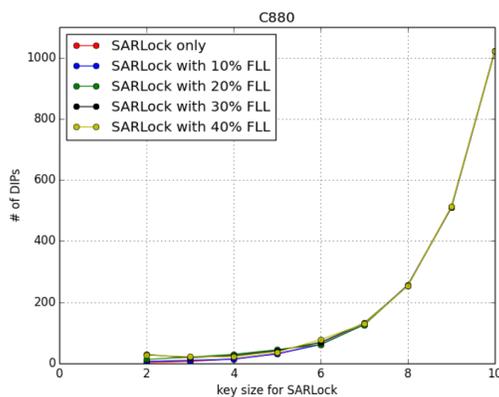


Fig.3 SARLock + FLL Implementation

The number of DIPS is a good parameter to see the SAT resilience of the circuit. It is exponentially proportional to the key size of SARLock.

Attack Techniques & Obfuscation Protection

Breakdown:

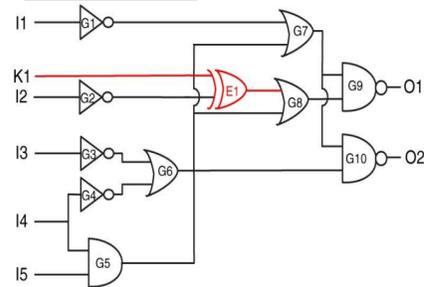


Fig.1 Circuit Encrypted with a key gate Inserted

Fig.1 shows a circuit encrypted with an XOR gate (E1), the XOR gate is what's known as a key gate. The key gate is responsible for giving correct outputs only if the correct key is given. In contrast, a wrong key will give a wrong output, and vice-versa. However, this defense is very vulnerable to an attack known as sensitization.

Sensitization:

The goal of sensitization is to determine the secret key used for the logic encryption. The attacker is assumed to have access to a locked netlist and also have access to a functional IC, which embeds the logic locking key. The functional IC is used as an oracle from which the attacker can get correct I/O pattern pairs. When it tries to know the value of one chosen key bit, the attacker tries to find an input pattern which can mute the propagation of all the unknown key bits to the output, such that the correctness of the output is solely determined by the key bit he/she is concerned about.

Strong Logic Encryption:

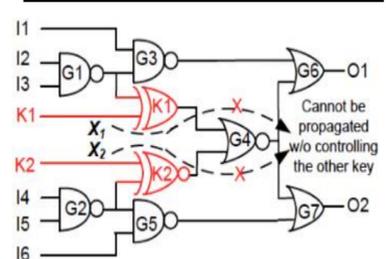


Fig.2 Circuit with two parallel Key-Gates

Fig.2 exhibits a Strong Logic Encrypted circuit. This technique prevents sensitization as both Key-Gates, (K1 & K2), are parallel to each other. This brings difficulty to the attacker because they need to find the value of one key gate (K1), however, they need the value of the other key gate (K2) in order to find the first key gate (K1). But, they can't have the value of (K2) unless they have the value of (K1), and vice-versa. This technique is vulnerable to an attack, known as SAT-Attack.

SAT-Attack:

Boolean satisfiability based attack (SAT Attack), is an attack that can decode most Logic Locking techniques within a reasonable amount of time, and also for a large key size. This attacks works by repeatedly solving formulas that will continuously get rid of incorrect keys until the correct keys are given. However, this can be prevented by a couple of obfuscation techniques known as SARLock and ANTISat.

Skills Learned

Throughout my seven weeks here at SHINE, I have learned many concepts including the basics of:

- Boolean Logic
 - Digital Circuits
- Principles of:
- Circuit/Logic Obfuscation

Coding was key to my research as I was responsible for creating a simulation function for circuits. Therefore, **Python** was the programming language that I learned to be able to create the simulation function.

```

def bool_to_int(x):
    return bool(x)

def bool_to_str(x):
    return '1' if x else '0'

def bool_to_hex(x):
    return hex(x)

def bool_to_bin(x):
    return bin(x)

def bool_to_dec(x):
    return dec(x)

def bool_to_oct(x):
    return oct(x)

def bool_to_base64(x):
    return base64.b64encode(x)

def bool_to_base32(x):
    return base32.b32encode(x)

def bool_to_base16(x):
    return base16.b16encode(x)

def bool_to_base8(x):
    return base8.b8encode(x)

def bool_to_base4(x):
    return base4.b4encode(x)

def bool_to_base2(x):
    return base2.b2encode(x)

def bool_to_base1(x):
    return base1.b1encode(x)

def bool_to_base0(x):
    return base0.b0encode(x)

def bool_to_base(x):
    return base.bencode(x)

def bool_to_base10(x):
    return base10.b10encode(x)

def bool_to_base20(x):
    return base20.b20encode(x)

def bool_to_base30(x):
    return base30.b30encode(x)

def bool_to_base40(x):
    return base40.b40encode(x)

def bool_to_base50(x):
    return base50.b50encode(x)

def bool_to_base60(x):
    return base60.b60encode(x)

def bool_to_base70(x):
    return base70.b70encode(x)

def bool_to_base80(x):
    return base80.b80encode(x)

def bool_to_base90(x):
    return base90.b90encode(x)

def bool_to_base100(x):
    return base100.b100encode(x)

def bool_to_base110(x):
    return base110.b110encode(x)

def bool_to_base120(x):
    return base120.b120encode(x)

def bool_to_base130(x):
    return base130.b130encode(x)

def bool_to_base140(x):
    return base140.b140encode(x)

def bool_to_base150(x):
    return base150.b150encode(x)

def bool_to_base160(x):
    return base160.b160encode(x)

def bool_to_base170(x):
    return base170.b170encode(x)

def bool_to_base180(x):
    return base180.b180encode(x)

def bool_to_base190(x):
    return base190.b190encode(x)

def bool_to_base200(x):
    return base200.b200encode(x)

def bool_to_base210(x):
    return base210.b210encode(x)

def bool_to_base220(x):
    return base220.b220encode(x)

def bool_to_base230(x):
    return base230.b230encode(x)

def bool_to_base240(x):
    return base240.b240encode(x)

def bool_to_base250(x):
    return base250.b250encode(x)

def bool_to_base260(x):
    return base260.b260encode(x)

def bool_to_base270(x):
    return base270.b270encode(x)

def bool_to_base280(x):
    return base280.b280encode(x)

def bool_to_base290(x):
    return base290.b290encode(x)

def bool_to_base300(x):
    return base300.b300encode(x)

def bool_to_base310(x):
    return base310.b310encode(x)

def bool_to_base320(x):
    return base320.b320encode(x)

def bool_to_base330(x):
    return base330.b330encode(x)

def bool_to_base340(x):
    return base340.b340encode(x)

def bool_to_base350(x):
    return base350.b350encode(x)

def bool_to_base360(x):
    return base360.b360encode(x)

def bool_to_base370(x):
    return base370.b370encode(x)

def bool_to_base380(x):
    return base380.b380encode(x)

def bool_to_base390(x):
    return base390.b390encode(x)

def bool_to_base400(x):
    return base400.b400encode(x)

def bool_to_base410(x):
    return base410.b410encode(x)

def bool_to_base420(x):
    return base420.b420encode(x)

def bool_to_base430(x):
    return base430.b430encode(x)

def bool_to_base440(x):
    return base440.b440encode(x)

def bool_to_base450(x):
    return base450.b450encode(x)

def bool_to_base460(x):
    return base460.b460encode(x)

def bool_to_base470(x):
    return base470.b470encode(x)

def bool_to_base480(x):
    return base480.b480encode(x)

def bool_to_base490(x):
    return base490.b490encode(x)

def bool_to_base500(x):
    return base500.b500encode(x)

def bool_to_base510(x):
    return base510.b510encode(x)

def bool_to_base520(x):
    return base520.b520encode(x)

def bool_to_base530(x):
    return base530.b530encode(x)

def bool_to_base540(x):
    return base540.b540encode(x)

def bool_to_base550(x):
    return base550.b550encode(x)

def bool_to_base560(x):
    return base560.b560encode(x)

def bool_to_base570(x):
    return base570.b570encode(x)

def bool_to_base580(x):
    return base580.b580encode(x)

def bool_to_base590(x):
    return base590.b590encode(x)

def bool_to_base600(x):
    return base600.b600encode(x)

def bool_to_base610(x):
    return base610.b610encode(x)

def bool_to_base620(x):
    return base620.b620encode(x)

def bool_to_base630(x):
    return base630.b630encode(x)

def bool_to_base640(x):
    return base640.b640encode(x)

def bool_to_base650(x):
    return base650.b650encode(x)

def bool_to_base660(x):
    return base660.b660encode(x)

def bool_to_base670(x):
    return base670.b670encode(x)

def bool_to_base680(x):
    return base680.b680encode(x)

def bool_to_base690(x):
    return base690.b690encode(x)

def bool_to_base700(x):
    return base700.b700encode(x)

def bool_to_base710(x):
    return base710.b710encode(x)

def bool_to_base720(x):
    return base720.b720encode(x)

def bool_to_base730(x):
    return base730.b730encode(x)

def bool_to_base740(x):
    return base740.b740encode(x)

def bool_to_base750(x):
    return base750.b750encode(x)

def bool_to_base760(x):
    return base760.b760encode(x)

def bool_to_base770(x):
    return base770.b770encode(x)

def bool_to_base780(x):
    return base780.b780encode(x)

def bool_to_base790(x):
    return base790.b790encode(x)

def bool_to_base800(x):
    return base800.b800encode(x)

def bool_to_base810(x):
    return base810.b810encode(x)

def bool_to_base820(x):
    return base820.b820encode(x)

def bool_to_base830(x):
    return base830.b830encode(x)

def bool_to_base840(x):
    return base840.b840encode(x)

def bool_to_base850(x):
    return base850.b850encode(x)

def bool_to_base860(x):
    return base860.b860encode(x)

def bool_to_base870(x):
    return base870.b870encode(x)

def bool_to_base880(x):
    return base880.b880encode(x)

def bool_to_base890(x):
    return base890.b890encode(x)

def bool_to_base900(x):
    return base900.b900encode(x)

def bool_to_base910(x):
    return base910.b910encode(x)

def bool_to_base920(x):
    return base920.b920encode(x)

def bool_to_base930(x):
    return base930.b930encode(x)

def bool_to_base940(x):
    return base940.b940encode(x)

def bool_to_base950(x):
    return base950.b950encode(x)

def bool_to_base960(x):
    return base960.b960encode(x)

def bool_to_base970(x):
    return base970.b970encode(x)

def bool_to_base980(x):
    return base980.b980encode(x)

def bool_to_base990(x):
    return base990.b990encode(x)

def bool_to_base1000(x):
    return base1000.b1000encode(x)
    
```

Code created for the simulation function

- Communication skills was another skill learned as it was key to my research as I was able to express my concerns and share some ideas with the team.

Impact of Professor's Research

Hardware security is as important as software and network security. Many people nowadays are concerned about their privacy and security. Circuits are stored with lots of confidential information that attackers are trying to reveal. Also, attackers can steal the design of a, non-obfuscated, circuit and can illegally reproduce and sell it for profit, which is what is trying to be prevented by hardware obfuscation.

In Professor Pierluigi Nuzzo's lab, we are trying to model and optimize multiple hardware obfuscation techniques for hardware Security. A tool for finding the best obfuscation solution to any IC is their ultimate goal in this research.

Acknowledgements

I would like to thank Professor Pierluigi Nuzzo, my Ph.D. mentors, Yinghua Hu, and Subhajit Dutta Chowdhury, my lab mate, Tristan Brankovic; Dr. Katie Mills, Dr. Megan Herrold, Patrick Valadez and the SHINE team; my teachers, Mrs. Karin Ching and Ms. Rosa Garcia; Cynthia Gonzalez, Lauren Lizarrga, and all my TELACU family; and lastly I would like to thank my parents and my siblings, Luis, Maria, Evelinda, Sam, and Sophia, for supporting and pushing me to achieve my best, it will all pay off soon.

Advice for Future SHINE Students

To all future SHINE students:

- Do not be afraid of the material that will be given to you in your labs, there is a range of profound vocabulary that may seem frightening at first, but with constant effort in it, you will get the hang of it and it will become understandable.
- Do not be discouraged if you do not know the material or the subject at all, talk with your mentors as they are there to fully guide you and they will not let you down.
- Do a thorough background survey of your research as it will give you a better understanding of what you will be working with.
- Make connections and ask questions with the SHINE team, they will be there for you and will also help you to strengthen your academics.
- Make the seven weeks at SHINE last!!! Get to know your fellow SHINE peers, go out and have fun, it will definitely brighten up your experience.

References

- Kirov, D., Nuzzo, P., Passerone, R., & Sangiovanni-Vincentelli, A. (2017, June). ArchEx: An extensible framework for the exploration of cyber-physical system architectures. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE* (pp. 1-6). IEEE.
- Bajaj, N., Nuzzo, P., Masin, M., & Sangiovanni-Vincentelli, A. (2015, March). Optimized selection of reliable and cost-effective cyber-physical system architectures. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* (pp. 561-566). EDA Consortium.
- Rajendran, J., Zhang, H., Zhang, C., Rose, G. S., Pino, Y., Sinanoglu, O., & Karri, R. (2015). Fault analysis-based logic encryption. *IEEE Transactions on computers*, 64(2), 410-400240.
- Rajendran, J., Pino, Y., Sinanoglu, O., & Karri, R. (2012, June). Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference* (pp. 83-89). ACM.
- Yasin, M., Mazumdar, B., Rajendran, J. J., & Sinanoglu, O. (2016, May). Sarlock: Sat attack resistant logic locking. In *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on* (pp. 236-241). IEEE.