The ISC Int'l Journal of Information Security

July 2021, Volume 13, Number 2 (pp. 157–162)

http://www.isecure-journal.org

An Obfuscation Method Based on CFGLUTs for Security of FPGAs

Mansoureh Labafniya [1,*] and Shahram Etemadi Borujeni [1]

[1] *Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.*

ARTICLE INFO.

Article history:
Received: June 11, 2020
Revised: February 23, 2021
Accepted: May 19, 2021
Published Online: June 19, 2021

Keywords:
Configurable Look Up Tables (CFGLUTs), Hardware Trojan Horses (HTHs), Obfuscation, Hardware Security

Type: Short Paper

doi: 10.22042/isecure.2021.234848.557

dor: 20.1001.1.20082045.2021.13.2.6.1

Abstract

There are many different ways of securing FPGAs to prevent successful reverse engineering. One of the common forms is obfuscation methods. In this paper, we proposed an approach based on obfuscation to prevent FPGAs from successful reverse engineering and, as a result, hardware trojan horses (HTHs) insertion. Our obfuscation method is using configurable look up tables (CFGLUTs). We suggest to insert CFGLUTs randomly or based on some optional parameters in the design. In this way, some parts of the design are on a secure memory, which contains the bitstream of the CFGLUTs so that the attacker does not have any access to it. We program the CFGLUTs in run-time to complete the bitstream of the FPGA and functionality of the design. If an attacker can reverse engineer the bitstream of the FPGA, he cannot detect the design because some part of it is composed of CFGLUTs, which their bitstream is on a secure memory. The first article uses CFGLUTs for securing FPGAs against HTHs insertion, which are results of reverse engineering. Our methods do not have any power and hardware overhead but 32 clock cycles time overhead.

1 Introduction

Reconfigurable hardware platforms consist of an array of distributed logic and interconnection blocks for implementing digital circuits that can be programmed and reprogrammed. Field-programmable gate arrays (FPGAs) are reconfigurable hardware platforms commonly used in all types of applications, including those that deal with secure data. This typical usage results in the security of an FPGA to be noticeable for designers [1]. Security of an FPGA is essential in all stages, from design/fabrication flow until the developer and end-user uses it. Figure 2 shows the malicious alteration of the FPGA during the different phases in the design flow. The untrusted foundry in FPGA's life cycle is a potential place that an attacker can insert its HTH in the design. Different IPs and EDA tools used by the developer of the system can work as an adversary and insert HTHs in the system. The attacker can conflict in internal nodes by inserting malicious code and changing bitstream when used by the end-user [2]. This attack is available by reverse-engineering the bitstream. Reverse engineering is by reading the bitstream from a nonvolatile external memory of the FPGA [2] using related tools that help extract the bitstream and related netlist of the FPGA [3, 4]. The side-channel attack, cloning, authentication attack, and radiation attack are different attack models represented by Drimer, which can be implemented in the untrusted foundry, configured phase, or even when used with end-user [5]. These mentioned attacks in the differ-

* Corresponding author.
Email addresses: mlabaf@eng.ui.ac.ir, etemadi@eng.ui.ac.ir

ISSN: 2008-2045

ent stages can be defined both in FPGAs and ASICs. To have a secure system, it must have a design for

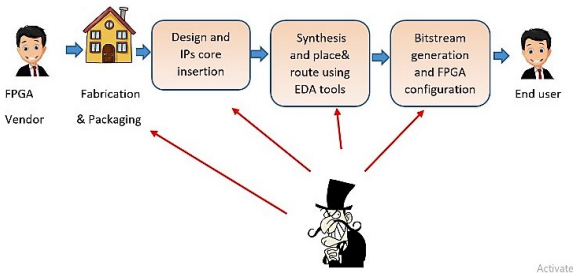


Figure 1. FPGA design flow and potential insertion of HTHs

security (DFS). One of the most parameters in DFS is prevention methods to avoid adversary to insert HTHs in a system. Despite prevention methods, if an adversary attacks the system and inserts HTHs in it, detection methods can detect trojan's presence and retrieve it. There are not many articles that propose protection methods against HTH insertion in FPGA. Mal-Sarkar *et al.* [6] introduced a new TMR structure named adapted triple modular redundancy (ATMR). ATMR uses a different structure for implementing its module. This is highly unlikely that two structure triggers simultaneously by HTH activation. Both conventional TMR and proposed ATMR have high area overhead and power consumption because of its redundant structure. Filling unused space on FPGA is another way to protect it from HTH insertion. In [7], after identifying unused space, filling them with dummy logic cells. This method imposes no performance and power penalty. Besides, it has a detection method if any attacker can insert HTHs in dummy cells. Using physical and logical keys in [8] improves the security of an FPGA system. It has priced a time overhead as specialized FPGA is needed that must be configured differently. Different techniques are introduced to detect HTH insertion on an FPGA. These detection techniques are categorized into two classes: destructive and non-destructive. Destructive methods consist of demetallization of the manufactured circuit and imaging layer by layer. The probable existence of HTH will be revealed by analyzing the images. Logic-based testing like automatic test pattern generation (ATPG) and side-channel analysis like power, temperature, timing, or electromagnetic variation are non-destructive ways to detect HTHs insertion [9]. In this paper, a new prevention method based on obfuscation is presented to secure the FPGA against HTHs insertion and reverse engineering. Our proposed method is based on using CFGLUTs in the design to increase the security of an FPGA. Our proposed method prevents successful reverse engineering of the FPGA bitstream and HTHs insertion on it. Our obfuscated design, is a kind of bitstream concealing or split manufacturing which makes direct wire-tapping ineffective by integrating memory within FPGA. The idea of split manufacturing technique can be used for bitstream concealing. In this way the key configuration bitstreams are stored in the flash memory of the FPGA and other non-critical configuration bitstreams are stored in the external memory. In this way, only partial useless information about FPGA mapping relationship will be leaked to eavesdroppers, which significantly increases the difficulty of bitstream reverse engineering [10]. In this paper, during Section 2, we review different papers related to the obfuscation method. We also describe the application of CFGLUTs in security and related articles. Section 3 describes our approach, which is based on CFGLUTs to obfuscate the design. Implementation results are explained in Section 4. In the last section, we concluded the paper.

2 Previous Work

2.1 Obfuscation Methods

Many articles suggested different approaches to obfuscate the design during various stages of the chip's lifecycle. The majority of them are based on obfuscation by inserting a switch box and LUTs in ASICs chip design to secure them in the fabrication stage [11, 12]. Some of the other articles obfuscate the ASICs by inserting an extra gate in design and stablishing a key to activate the chip before selling them [13, 14]. These methods are named "logic encryption". By inserting extra gates in low controllable points in design, in addition to establishing a key to activate the chip, we decrease the chance of HTHs insertion in design. The other place for inserting the encryption key is the point with positive slack time. At any timing point, the slack time is the difference of its required arrival time minus its arrival time. Inserting extra gates at this point will not change the critical delay of design [13, 15]. Logic encryption prevents both reverse engineering at the fabrication level and cloning/overbuilding. This method also increases the controllability of some points in design in which their activities are low. The other method is based on using the physical unclonable function (PUF) structure for producing obfuscation in design [8, 16, 17]. In this way, the PUF structure can produce a unique key or identify a fake chip from a genuine one. IC camouflage technique is another obfuscation method by inserting different redundant standard logic units in some empty locations of the physical architecture of ASICs to hide the correct circuit. In this technique, even if the attacker can reverse engineer the netlist, the right circuit function cannot be obtained [4]. The mentioned papers are different methods to prevent reverse engineering of ASICs by different obfuscation ways. The different structure of FPGA, in compar-

ison to ASIC, produces other obfuscation methods to protect them. On the otherhand, the existence of bitstream for programming and finalizing FPGAs' design causes various attacks on FPGAs. The obfuscation method on FPGAs tries to protect bitstream from successful reverse engineering. Bitstream concealing makes by integrating memory within FPGA is bitstream protection. Flash memory FPGA and antifuse FPGA do not need external configuration memory to contain bitstream. Therefore, there is not any external memory to be accessible to the attacker. Compared with SRAM FPGAs, these FPGAs need specific equipment for reverse engineering [10]. Despite that, SRAM FPGAs are more usually used. Encrypting the bitstream by different encryption algorithm is a method to protect bitstream in SRAM FPGAs. Obfuscation methods to protect bitstream from successful reverse engineering are presented in [7, 18]. Filling unused space is a kind of obfuscation method to prevent HTHs insertion by misleading the attacker to detect between main gates and redundant ones. This method is introduced for both ASICs and FPGAs [7, 18, 19]. Using evolvable hardware (EH) architectures is another method to change the configuration of FPGAs and its behavior dynamically based on inputs from the environment. In [20], the feasibility of using EH to prevent hardware Trojan horses from being inserted, activated, or propagated in a digital electronic chip is investigated. Paper [20] presents an obfuscation method based on changing the FPGAs' configuration periodically using EH.

2.2 CFGLUT Structure

The CFGLUTs are LUTs that can be configured at run-time from the FPGA. The structure of the CFGLUT5 in Virtex-5 consists of 5-input and 1-output LUT or 4-input and 2-output LUT in addition to a configuration input (CDI) and a configuration output (CDO). This module can be used for partial reconfiguration of FPGAs in run-time instead of manipulating bitstream for reconfiguration. The internal structure of CFGLUT consists of a 16-bit shift configurable memory followed by the subsequent multiplexer stage. Its configuration memory size is 16 bits so that four input is considered for this module. Each RFGLUT is loaded with an INIT value that presents the truth table of the LUT. It is allowable to change LUT's functionality by changing this INIT value in run-time, which gives the user the power of partial reconfiguration of the FPGAs internally. Reconfiguration is done by activating CE port and simultaneously putting 1-bit reconfiguration data on the CDI port. One bit is written in INIT register in each clock cycle. Sixteen clocks are needed to reconfigure the CFGLUT entirely. Figure 2 shows the structure of a CFGLUT. The only paper that uses CFGLUTs for security is [21] in which a method for securing encryption algorithm against side-channel attacks (SCA) is recommended.

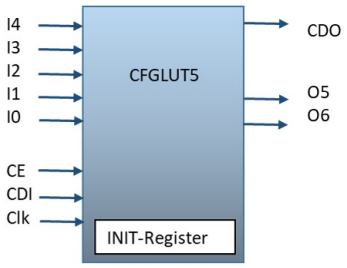


Figure 2. CFGLUT structure

3 The Proposed Obfuscated Design

Our proposed method prevents successful reverse engineering of the FPGA bitstream and HTHs insertion on it. We suggest two different methods to obfuscate the bitstream using CFGLUTs. In the first proposed method, the designer writes the HDL codes and then simulate it to be ensured from its correct functionality. In the next step, we select some part of the HDL code to implement them by CFGLUTs. Choosing the specified portion of the HDL code to substitute them with CFGLUTs can be:

- randomly selected.
- some vulnerable parts of the HDL code, like SBOX in encryption algorithms.
- based on some special parameters like the slack time or middle point of paths [4, 13, 22, 23].

In our implementation, we select some parts of the HDL code randomly to substitute with CFGLUTs. Figure 3 shows our suggested method. We imagine that the code is composed of different modules. We can use CFGLUTs for implementing some parts of the modules, whole the module, or only one gate. The related bitstreams of CFGLUTs are saved in a secure memory loaded dynamically on CFGLUTs in run time. The attacker read the bitstream of the FPGA to reverse engineering and inserting HTHs or some other malicious purpose. But after reverse engineering, he will consider that some CFGLUTs obfuscate the design that their codes do not exist in the primary bitstream of the FPGAs but separated secure memory. In this way, the attack will not be successfully implemented. In this paper, we imagine that the bitstream for the FPGA and the bitstream for programming CFGLUTs are in different memory so that the attacker is unaware of this. Most memories have built-in error detection/correction mechanisms, which is also the case for the block RAM (BRAM) modules that can be used. Our method is a kind of obfuscation method based on CFGLUTs to prevent

HTHs insertion and successful reverse engineering attack on FPGAs. The second proposed method to use

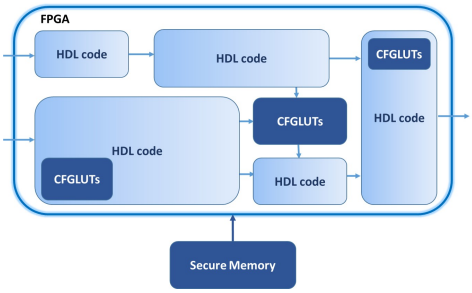


Figure 3. Schematic of our proposed method for securing the FPGA

CFGLUTs is to insert them like a logic encryption method. In this way, besides, to have a key for increasing the security, which is necessary for the circuit's proper working, we have obfuscation in the bitstream that prevents the attacker from reversing engineer the circuit successfully. In this way, we decrease the low controllable points in the design by inserting these CFGLUTs in proper places in the netlist. Figure 4 shows a circuit secured in Figure 5 by inserting CFGLUTs randomly in some nodes. Each CFGLUT can implement one of the AND, OR, NAND, NOR, XOR, XNOR, or NOT gate. It must be selected according to the key and keeping the correct functionality of the final output. For the proper working of the design, first, the right key must be entered. Besides, by saving the bitstream of CFGLUTs in a separated secure memory, we prevent successful reverse engineering of the design's bitstream.

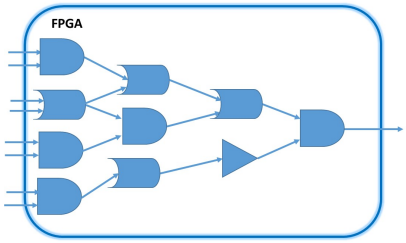


Figure 4. Original circuit

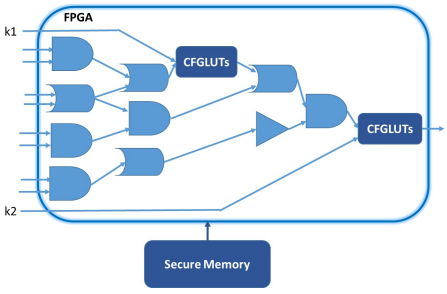


Figure 5. Secured circuit

4 Attack Analysis

The proposed obfuscate structure can protect against HTH insertion by uploading a portion of the circuit configuration at run-time. That means we protect against: (1) When EDA tools manipulate the design: We have two separated bitstream to upload on the FPGAs. One of them is the final bitstream without CFGLUTs related bitstream, and the other one is the bitstream of CFGLUs. EDA tools cannot reverse engineer or successfully insert any HTHs in the FPGAs as they do not have a final bitstream, which consists of the content of CFGLUTs. (2) When a design is in run-time mode: we have two separate memories for saving the two bitsrtreams that complete each other to produce the final bitstream. One of them is more secure so that attackers do not have access or capability to read it. So attackers cannot reverse engineer the bitstream successfully.

4.1 Implementation Results

We use the "mem-ctrl" sample code from the IWLS benchmark to evaluate our proposed work. We use the Vivado2018.2 version for synthesis and implementation. First, we synthesis and implement the code without any CFGLUT insertion, then we randomly select some statements and modules to substitute them with CFGLUTs. Figure 6 shows a portion of the netlist, which is the result of "mem-ctrl" implementation. Figure 7 shows the LUT, which implements the below instruction: "assign init_ack_fe= init_ack_r & !init_ack" (1) Figure 5 shows the CFGLUT5, which implements the below statement: "CFGLUT5(.INIT(32'h00000000))inst0 (.O5(init_ack_fe),.CDI(CDI0),.CE(CE),.CLK(clk), .I0(init_ack),.I1(!init_ack_r),.I2(0),.I3(0),.I4(0))" (2) We write statement2 instead of the statement1 in the HDL code. "init_ack_r", "init_ack" and "init_ack_fe" are some internal signals in the HDL code. The stream of bits equal to "32'h0000100" indicates the functionality of the statement1 is stored in a secure memory, separated from memory, which contains the bitstream of the FPGA. This stream is imposed on CDI0 input pin of CFGLUT to program it in run time. Simulation results show that by substituting one CFGLUT with one LUT, the number of LUTs decrease from 1081 to 1080, and one CFGLUT is added to used resources of the FPGA. Time overhead is equal to 32 clock cycles for programming all CFGLUTs in parallel to each other. Static and dynamic power is equal to 0.006w and 0,242w, respectively, both with and without inserting CFGLUT to code.

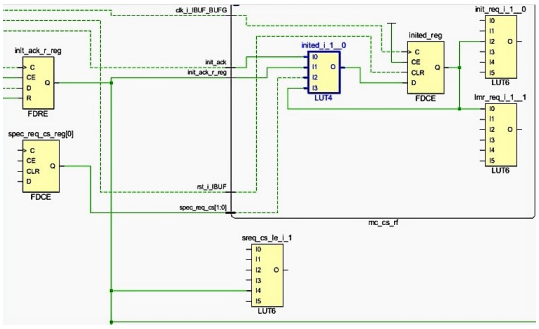


Figure 6. The netlist of the design (the primary netlist)

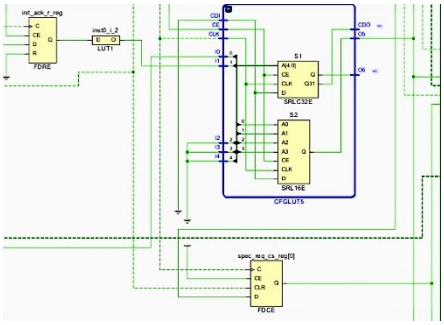


Figure 7. The netlist of the design (the CFGLUT5 which is inserted instead of the LUT)

Table 1. Comparison of different obfuscation method in FPGAs

Paper	Used method	Overhead
[10]	encrypting bitstream	bitstream
[7, 18]	filling space	bitstream
[20]	EA	hardware/time/power
proposed method	using CFGLUTs	memory

4.2 Comparison

This section compares our proposed method with related articles that used the obfuscation method to secure the FPGA in Table 1. Paper [10] encrypt the bitstream to secure the design. This method has an overhead of time to encrypt and decrypt the bitsream, although this method is broken. Paper [7] fill all resources of the FPGA to secure it. This method has overhead in code. Overhead in resource consumption, power and time is the result of using EA in [20]. In this paper's proposed method, we have a time overhead of 16/32 clock cycles to program the CFGLUTs, although if we program them in the spare time of the chip, we can ignore the overhead. Also, we need an extra memory to save the CFGLUTs' bitstream. We do not have any power and hardware overhead. The designer can select each of the mentioned methods to obfuscate the design according to sacrified parameters. The method selection is based on the application.

5 Conclusion

In this paper, we proposed a prevention method for the security of the FPGA based on obfuscation. Our first proposed method is to use CFGLUTs for implementing some parts of modules or the whole of a module to prevent successful reverse engineering attacks. Saving the bitstream of CFGLUTs in separated and secure memory made obfuscation in the bitstream of the design. In this method, we replace the LUTs with CFGLUTs. The second suggestion method is to use the CFGLUT in the logic encryption structure. In this method, we insert extra gates instead of substituting them. Adding additional gates to decrease the potential nodes for HTHs insertion and having a key to secure the design is the second proposed application of CFGLUTs in this paper. It is the first article that is using CFGLUTs for securing FPGAs against HTHs insertion and reverse engineering. Our methods do not have any power and hardware overhead if we substitute each LUT with one CFGLUT but 32 clock cycles time overhead. Despite that, if we insert extra CFGLUTs as a logic encryption method, we have hardware overhead equal to the number of added gates.

References

[1] Steve Trimberger. Trusted design in fpgas. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pages 5–8, New York, NY, USA, 2007. ACM.

[2] Hoyoung Yu, Hansol Lee, Sangil Lee, Youngmin Kim, and Hyung-Min Lee. Recent advances in fpga reverse engineering. *Electronics*, 7(10):246, 2018.

[3] Jean-Baptiste Note and Éric Rannaud. From the bitstream to the netlist. In *FPGA*, volume 8, pages 264–264, 2008.

[4] Jeyavijayan Rajendran, Michael Sam, Ozgur Sinanoglu, and Ramesh Karri. Security analysis of integrated circuit camouflaging. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 709–720, 2013.

[5] Saar Drimer. Volatile fpga design security–a survey. *IEEE Computer Society Annual Volume*, pages 292–297, 2008.

[6] Sanchita Mal-Sarkar, Robert Karam, Seetharam Narasimhan, Anandaroop Ghosh, Aswin Krishna, and Swarup Bhunia. Design and validation for fpga trust under hardware trojan attacks. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):186–198, 2016.

[7] Mansoureh Labbafniya and Roghaye Saeidi. Secure fpga design by filling unused spaces. *ISeCure-The ISC International Journal of Infor-*

mation Security, 11(1):47–56, 2019.

[8] Greg Stitt, Robert Karam, Kai Yang, and Swarup Bhunia. A uniquified virtualization approach to hardware security. *IEEE Embedded Systems Letters*, 9(3):53–56, 2017.

[9] Mohammad Tehranipoor and Cliff Wang. *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[10] Jiliang Zhang and Gang Qu. Recent attacks and defenses on fpga-based systems. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 12(3):1–24, 2019.

[11] Sharareh Zamanzadeh and Ali Jahanian. Automatic netlist scrambling methodology in asic design flow to hinder the reverse engineering. In *2013 IFIP/IEEE 21st International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 52–53. IEEE, 2013.

[12] Soroush Khaleghi, Kai Da Zhao, and Wenjing Rao. Ic piracy prevention via design withholding and entanglement. In *The 20th asia and south pacific design automation conference*, pages 821–826. IEEE, 2015.

[13] Sophie Dupuis, Papa-Sidi Ba, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 49–54. IEEE, 2014.

[14] Jeyavijayan Rajendran, Huan Zhang, Chi Zhang, Garrett S Rose, Youngok Pino, Ozgur Sinanoglu, and Ramesh Karri. Fault analysis-based logic encryption. *IEEE Transactions on computers*, 64(2):410–424, 2015.

[15] Andrea Marcelli, Marco Restifo, Ernesto Sanchez, and Giovanni Squillero. An evolutionary approach to hardware encryption and trojan-horse mitigation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1593–1598. IEEE, 2017.

[16] Sharareh Zamanzadeh and Ali Jahanian. Asic design protection against reverse engineering during the fabrication process using automatic netlist obfuscation design flow. *ISeCure*, 8(2), 2016.

[17] Ghobad Zarrinchian and Morteza Saheb Zamani. Latch-based structure: A high resolution and self-reference technique for hardware trojan detection. *IEEE Transactions on Computers*, 66(1):100–113, 2016.

[18] Behnam Khaleghi, Ali Ahari, Hossein Asadi, and Siavash Bayat-Sarmadi. Fpga-based protection scheme against hardware trojan horse insertion using dummy logic. *IEEE Embedded Systems Letters*, 7(2):46–50, 2015.

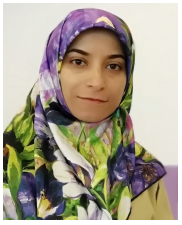
[19] Kan Xiao and Mohammed Tehranipoor. Bisa: Built-in self-authentication for preventing hardware trojan insertion. In *2013 IEEE international symposium on hardware-oriented security and trust (HOST)*, pages 45–50. IEEE, 2013.

[20] Mansoureh Labafniya, Stjepan Picek, Shahram Etemadi Borujeni, and Nele Mentens. On the feasibility of using evolvable hardware for hardware trojan detection and prevention. *Applied Soft Computing*, page 106247, 2020.

[21] Pascal Sasdrich, Amir Moradi, Oliver Mischke, and Tim Güneysu. Achieving side-channel protection with dynamic logic reconfiguration on modern fpgas. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 130–136. IEEE, 2015.

[22] Jie Li and John Lach. At-speed delay characterization for ic authentication and trojan horse detection. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 8–14. IEEE, 2008.

[23] Mohammad Saleh Samimi, Ehsan Aerabi, Zahra Kazemi, Mahdi Fazeli, and Ahmad Patooghy. Hardware enlightening: No where to hide your hardware trojans! In *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 251–256. IEEE, 2016.



Mansoureh Labafniya received her B.S. in hardware computer engineering from Islamic Azad University, South Tehran branch, Tehran, Iran in 2008. Her first M.S. degree is in computer architecture in 2010 and her second M.S. degree is in mechatronic Engineering from Sharif University of Technology, Tehran, Iran in 2012. She was a visiting researcher at KU Leuven for six months in 2018 and 2019. She got his Ph.D. degree in Computer Architecture Engineering at University of Isfahan, Iran in 2020. Her research interests include Hardware security, Digital system design and Residue number system.

Shahram Etemadi Borujeni is born in Borujen, Iran in 1964. He got his B.Sc. in electrical engineering from Iranian University of Science and Technology in 1987, and his M.Tech. degree in Radar and Communication Engineering from Indian Institute of Technology, Delhi in 1992. He got his Ph.D. degree in computer architecture engineering at Shahid Beheshti University, Iran in 2010. He is now with computer engineering faculty at University of Isfahan, Iran. His research interest includes Image encryption, Design for test and Hardware security.