Hardware Obfuscation and Logic Locking: A Tutorial Introduction

Tamzidul Hoque
University of Florida

Swarup Bhunia
University of Florida

Rajat Subhra Chakraborty
Indian Institute of Technology Kharagpur

Editor's note:
If you are designing or integrating hardware IP blocks into your designs, and you are using common global supply chains, then reading this overview article on how to protect your IP against reverse engineering, piracy, and malicious alteration attacks is a must. The authors give a comprehensive overview of current countermeasures that can be used at RTL, gate-, and layout-level to protect your design with a focus on combinational and sequential logic locking and a discussion on merits, overheads, and shortcomings of such techniques.

—Jürgen Teich, FAU Erlangen

SoCs are central to modern electronic systems, particularly in the Internet of Things (IoT) devices that often support a multitude of functions while maintaining a small form factor. To enable a shorter production cycle and reduced development cost, the SoC integrators routinely purchases hardware intellectual property (IP) cores from different vendors. Due to this reliance on the third-party IP (3PIP)-based ecosystem, the worldwide market of semiconductor IP is expected to reach $8 billion [1]. Meanwhile, most of today's design houses are fabless; therefore, they depend on offshore fabrication facilities for manufacturing the integrated design. Hence, the supply chain of the semiconductor industry has incorporated various foreign entities focusing on individual production steps. However, serious trust issues have emerged in recent times with the increasing globalization of the supply chain which has resulted in the proliferation of theft, reverse engineering (RE), and piracy of the hardware IP [2], [3]. IP protection standards vary significantly across the globe, and in some countries, these laws are not properly enforced. This facilitates a malicious entity to steal and/or clone and modify IPs of its competitors. Hence, traditional countermeasures based on passive methods, such as patenting, copyrighting, and watermarking of IPs, are no longer effective in such a globalized supply chain. Furthermore, unlike the software industry, the semiconductor supply chain cannot benefit from traditional cryptographic solutions due to the requirement of white-box accessibility to the hardware IP by untrusted parties.

To address the emerging needs of IP protection, hardware obfuscation has emerged as a potential countermeasure against major supply chain attacks. Obfuscation aims to hide the design intent, and/or "lock" the actual functionality by introducing functional locking and structural transformations to the IP [4]. Although the locking of the correct functional behavior of the design increases the resistance against unauthorized use, it may not be useful in preventing RE and judicious alteration of the design. Hence, with the aid of structural transformations, design intents are obfuscated to protect against the uncovered attacks. As Figure 1 illustrates, the original design is

Digital Object Identifier 10.1109/MDAT.2020.2984224
Date of publication: 30 March 2020; date of current version: 24 June 2020.

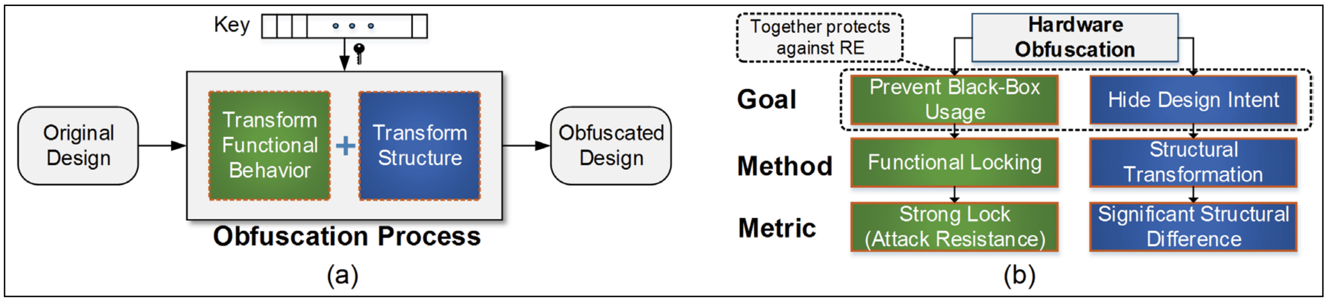


Figure 1. Overview of the obfuscation process: (a) hardware design is transformed through various functional and structural modifications to generate an obfuscated design; (b) each transformation is applied to accomplish specific security goals.

obfuscated (often using a randomly generated key) by introducing various functional and structural transformations to generate an obfuscated design. Without knowing the obfuscation key, the hardware design cannot even be used as a black-box module, as it would not produce correct functional behavior. The design intent also becomes obscure due to structural transformations. These obfuscation methods are evaluated against various attacks that try to leak the obfuscation key or bypass the locking mechanism (e.g., brute-force, removal [5], and Boolean satisfiability (SAT) [6]). The quality of obfuscation is assessed with metrics that quantify its effectiveness with respect to various attacks and resultant changes in the obfuscated design due to transformations [7].

In this article, we begin by discussing the threats in the semiconductor supply chain, present the concept of hardware obfuscation as a viable defense mechanism against these threats, and discuss how this is creating an arms race between the researchers playing the role of attackers and defenders. We provide an overview of various hardware obfuscation methods that exist with respect to their applicability in various stages of the supply chain. We also discuss three different postsynthesis obfuscation methods in detail, while introducing and explaining suitable metrics being used for understanding their effectiveness. Finally, we discuss various reported attacks on these obfuscation techniques and methodologies to inhibit some of these efforts.

Threats in the semiconductor supply chain

Today's semiconductor supply chain is distributed across untrusted entities situated throughout the globe. The primary parties involved in today's SoC supply chain are presented in Figure 2. The design house intending to build an SoC or any form of integrated circuit (IC) first acquires individual IPs (soft, firm, or hard) from multiple third-party vendors and integrates them. The integrated IP is then sent to an untrusted entity called the design-for-test (DFT) vendor for test point insertion to facilitate the impending postsilicon debug. During this stage, the IP is modified with additional logic that improves its controllability and observability. The modified design is forwarded to the foundry for fabrication. Since most of the design houses are fabless, the fabrication process is also performed through a third-party foundry. After fabrication, test, and packaging, the physical chip is integrated in a larger electronic system [i.e., printed circuit board (PCB)]. The assembly of the final product is usually handled by another untrusted company from where it is shipped to individual end users. In the following section, we discuss the different security threats that have emerged due to this globalization of the supply chain.

IP Piracy

A number of entities in the supply chain can illegally use or trade the design with little or no modification. This is referred to as *IP piracy*. Piracy could occur in the following stages of the supply chain.

(i) *Design house*: Even though the design house is considered an authorized purchaser, piracy could occur due to the lack of control over the use of a purchased IP. With trivial or no modification, the design house could market a purchased IP to others under a different brand name, or illegally distribute the IP without the knowledge of the IP vendor.

(ii) *DFT vendor*: Inclusion of debug infrastructures at the DFT stage requires complete access to the gate-level design of the IP. Hence, offshore untrusted DFT facilities have complete flexibility in performing IP piracy through cloning or unauthorized distribution of the design.

(iii) *Foundry*: Foundries receive the functional design in the form of a GDS-II database, which can be used to prepare the fabrication masks. Apart from performing the assigned job of fabricating ICs for the design house, the foundry or some of its unscrupulous employees could illegally sell or distribute the obtained GDS-II files to others under a different, or even the original, vendor name (e.g., in a black market).

(iv) *Assembly and market*: In the traditional IC design flow, the assembly and end users do not receive a reusable version of the IP in the form of an RTL code, a netlist, or a layout. Therefore, the illegal use of IP is difficult. However, if the IP can be extracted from the IC through RE efforts, piracy is possible. For the reconfigurable hardware-based design flow where the hardware IP stays with the programmable device as a configuration file, piracy could occur in the field or untrusted assembly as these files are typically reusable if found unencrypted.

IP Overuse

IP overuse is a scenario where a rightful IP purchaser receives the IP for using a limited number of instances, but instead exceeds the licensed number of uses without letting the IP seller know. The following parties have the flexibility to overuse an IP.

(i) *Design house*: The IP vendor may only wish to sell the IP for fabricating a limited number of ICs or wish to obtain royalties on the total number of ICs being fabricated. However, there is hardly any technique to implement such "metering" that actively tracks the exact use of the IP.

(ii) *Foundry*: Without investing in any of the prior design steps, an untrusted foundry could produce any number of chips using the received GDS-II from the design house. Although the IP owner may intend to only fabricate (say) N SoC instances, the foundry could produce more than N SoCs and market the overproduced ones under the same or a different brand name.

Reverse engineering

RE facilitates the extraction of a more reusable or understandable abstraction of the IP. The IP could be reversed to various forms (e.g., layout, gate-level, RTL, and specification) depending on the original abstraction and RE capabilities available. RE could occur in the following stages.

(i) *Design house and DFT facility*: Since the design house obtains a reusable version of the IP, typically in the form of RTL or gate-level netlist from an IP vendor, the motivation behind RE would be to obtain a more in-depth understanding of the high-level algorithm and implementation details of the design. A dishonest design house could use such information to file a claim on the algorithm if not done so by the IP vendor. However, another motivation could be to perform trust verification of the IP to find potential *hardware Trojans* (malicious, hard-to-detect modifications in a circuit [8]) inserted by a corrupt IP designer. The DFT facility also obtains the gate-level design to improve its testability. A malicious DFT facility may try to extract the high-level algorithm of the design in the form of RTL, control-data flow graph, or state transition graph (STG) from the given netlist.

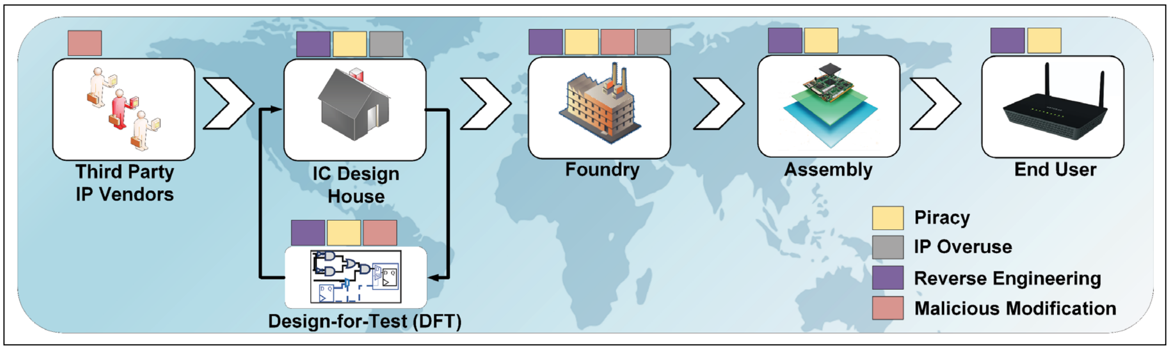


Figure 2. Most likely attacks on hardware IPs in each step of system development flow.

(ii) *Foundry*: If the foundry has any intent of dishonestly claiming IP rights, RE of the GDS-II, followed by a further reversal of the generated gate-level design, is needed. Understanding of the high-level design also facilitates hardware Trojan insertion. If the foundry intends to illicitly use only a part of the design, obtaining the gate-level design from the GDS-II could be helpful. RE of the layout could be done by analyzing the GDS-II files and corresponding physical description of standard cell libraries.

(iii) *Market*: Once fabricated, the IP is no longer available in a reusable form. Hence, attackers intending to perform piracy or searching for security vulnerabilities must reverse engineer the design from a manufactured IC. Destructive RE of the IC is possible through decapsulation of the chip followed by the application of imaging on the exposed die to extract individual layers one after another [9]. As shown in Figure 2, RE is one of the common forms of attacks in the field.

FPGA devices are commonly found in electronic systems today. For FPGAs, the configuration file or bitstream is at risk of RE and piracy since an end user with physical access to the FPGA could obtain the bitstream file from the on-chip configuration flash. If the bitstream is unencrypted, it could be reverse engineered to a netlist using software tools that significantly simplify Trojan insertion [10]. The bitstream could also be used to program any number of FPGAs of identical series. For an encrypted bitstream, various forms of attacks through probing, imaging, and side-channel analysis are viable that helps to recover the unencrypted configuration file [11] on which RE, piracy, and malicious modification become feasible.

Malicious modification

Malicious alteration of an electronic system in the supply chain is more seriously considered than ever; particularly after the recent revelation of malicious microchips embedded in server motherboards manufactured at an untrusted facility. The chip provides a backdoor for outsiders to compromise computer networks of almost 30 U.S. companies [12]. Although the aforementioned modification is done by introducing a new component in the PCB, a more sophisticated alteration made within a chip is significantly harder to find. Such malicious alteration at the chip level could be easily introduced by a 3PIP vendor, a DFT facility, or a foundry. To cause a specific malicious effect, the targeted modification of the design requires a complete or partial understanding of its functionality. For example, if an attacker knows which signals contain specific secret information at runtime, a rarely triggered functional leakage path could be designed that can later be used as a covert channel. Conversely, untargeted alteration could be made without having such an understanding of the design. For instance, a malicious circuit could be implemented using the unused spaces within the IC layout that increases the temperature of the design through excessive switching activity and eventually leads the IC to a parametric failure. The FPGA bitstream could be maliciously tampered by end users after its deployment to incorporate such Trojan circuits. Research demonstrates that such modification of the unencrypted bitstream could lead to a leakage of the cryptographic key [13] and accelerated aging [14] of the FPGA device.

What is hardware obfuscation?

Definition

In general, obfuscation refers to the technique of concealing or obscuring the functional behavior or specific information related to a logical or physical entity. In the field of cryptography and software, an obfuscator O is viewed as a transformer that modifies a program P to generate its obfuscated counterpart $O(P)$. Although P and $O(P)$ are functionally equivalent, an adversary would be unable to formulate P from $O(P)$. If the obfuscation process depends on a key during the generation of $O(P)$, the functional equivalence between P and $O(P)$ is attained only after the application of the secret key to $O(P)$ during operation.

In the context of hardware IPs, the primary goal of obfuscation is similar, i.e., the transformation of the design to introduce ambiguity in functional and structural behavior that protects the IP from piracy, RE, and malicious alteration. Both functional and structural transformations are essential to ensure security against all possible supply chain attacks.

Obfuscation in SoC supply chain

Obfuscation can be introduced in almost all stages of the semiconductor supply chain. Figure 3 shows the integration of obfuscation in a typical SoC design flow. As we will learn in the subsequent section, obfuscation methods are applicable to every possible abstraction of the hardware design. More specifically, the SoC design flow can benefit from obfuscation techniques applicable to RTL,

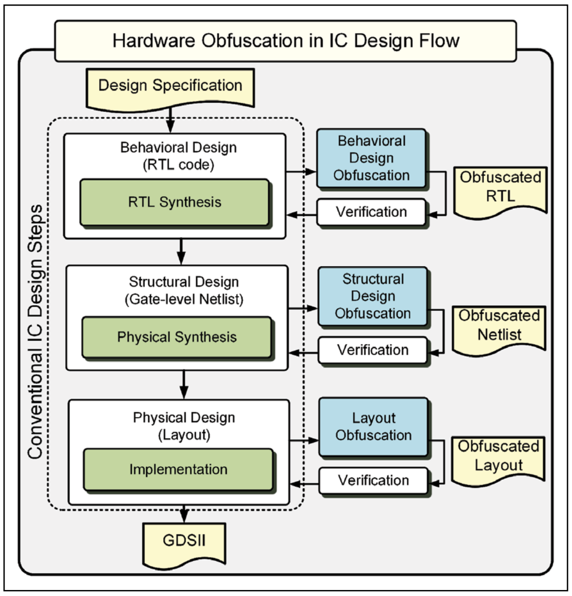


Figure 3. Integration of hardware obfuscation in the standard design flow.

gate-level, and layout representation of the IP. Unlike IP encryption, obfuscation applied at any level of the design stage is preserved even in the physical mask used for fabricating the IC. Hence, the hardware IP stays protected even beyond its digital form.

Apart from the obfuscation step, an additional verification process is required after obfuscation to ensure that the functionality can be retained by applying the correct obfuscation key. Most of the steps related to the obfuscation process can be completely automated using software tools. Besides, using appropriate metrics, the admissible security from obfuscation can be fine-tuned with respect to area, power, and delay constraints at design time.

Inapplicability of encryption in supply chain

Even though encryption using a cryptographically robust algorithm is an effective solution for ensuring the confidentiality of data, it is not applicable to many stages of the supply chain. For instance, soft-IPs purchased from third-party vendors are often received in encrypted forms that are simulated by the purchaser as black box. The computer-aided design (CAD) tools allow encrypted RTL codes to be simulated, given that the encryption key is securely transferred to the software tool for decrypting the IP prior to simulation. However, the gate-level design can still be generated using CAD tools to perform the subsequent design and manufacturing steps. The gate-level design is generated in an unencrypted form where all types of supply chain attacks are possible. Even the encrypted RTL code is not completely secure as the protection relies on the protocol that ensures a secure transfer of the decryption key to the CAD tool. Although the security is mathematically guaranteed, the implementation method of these protocols may have flaws, as found in the IEEE P1735 standard, which has been used for protecting encrypted RTL hardware description in certain CAD tools [15]. As mentioned earlier, most supply chain stages do not support encrypted hardware IPs, as the engineers at the untrusted vendor (e.g., DFT and foundry) often require white-box accessibility of the design to perform designated tasks. Even if all the preceding steps to fabrication are completely automated such that any need of human observation to the unencrypted design is eliminated, the manufacturing process that utilizes physical masks for each layer of the IC layout can no longer be concealed in a software-development process. Therefore, cryptographic solutions are applicable until a specific stage beyond which a solution-like hardware obfuscation is essential.

Difference from software obfuscation

The primary goal of software obfuscation is to hide the details of an algorithm or the design intent from its implementation in a high-level programming language (e.g., C and Python). Even though certain software obfuscation methods may directly apply to hardware IPs, the actual implication may differ when the hardware design is compiled to subsequent abstractions. For instance, a structural transformation of the C code may cause structural obfuscation in the corresponding assembly code, but the same transformation in the RTL code may not provide significant structural change when transformed to the gate-level design due to the logic optimization performed by the synthesis tool. The majority of software obfuscation methods do not require a key for applying the transformations; however, most hardware obfuscation methods use a key to lock the functionality that serves as an input to the obfuscated hardware.

Overview of obfuscation techniques

To protect various abstractions of hardware IP, a range of obfuscation techniques have been developed. Figure 4 illustrates a classification of these techniques based on the major hardware abstractions.

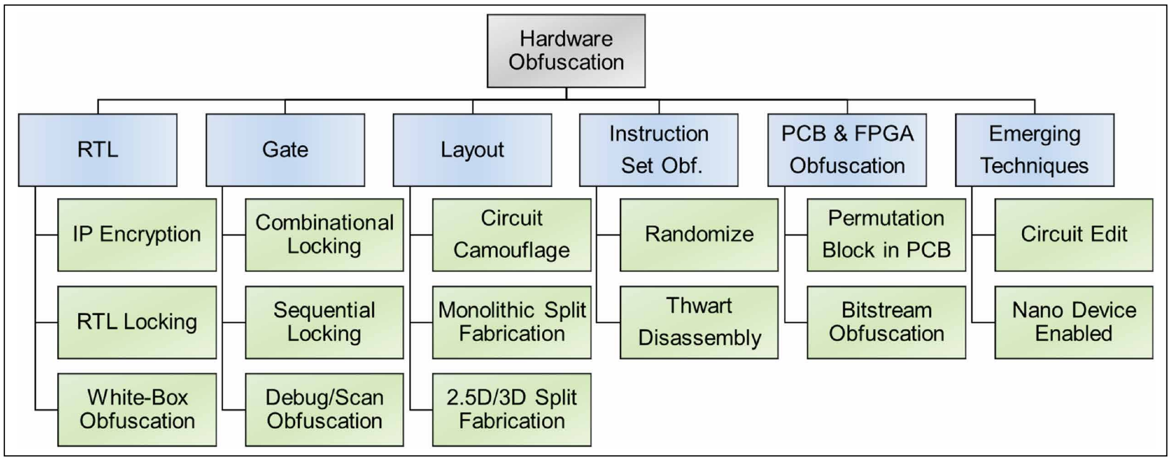


Figure 4. Overview of different obfuscation techniques applicable to various abstractions of hardware IP.

RTL obfuscation

An RTL description represents the hardware design in a very human-readable form. Hence, obfuscation of a high-level construct to hide the design intent is inherently difficult. Furthermore, the code structure of RTL is less regular than gate-level and layout counterparts, which makes automatic parsing and modification of the raw RTL code difficult. Obfuscation of the control and data-flow graph (CDFG) and STG corresponding to the RTL are effective and systematic methods of introducing functional locking. For CDFG transformation, the graph could be modified with dummy branches and nodes, where the correct flow would be dictated by a "key" input [16]. The inclusion of dummy transitions structurally modifies the CDFG and its corresponding RTL while ensuring a key-based locking of the functionality. STG of designs that rely on a finite state machine (FSM)-based controllers can also be locked by introducing dummy state transitions to the original FSM such that the added transitions must be traversed with an application of a correct key sequence to begin normal functional behavior [17]. The design is stuck in nonfunctional "obfuscated" states if a wrong key sequence is applied. The code transformation technique that renders a code intangible is a common software obfuscation method. Such white-box obfuscation techniques could be applied to RTL codes using transformations such as loop unrolling, signal name modification, and shuffling of independent statements [18].

Gate-level obfuscation

Gate-level obfuscation requires the inclusion of additional gates to introduce functional locking and structural modifications. These methods are often termed as logic locking. Combinational logic locking is a well-known gate-level obfuscation technique that requires the addition of "key gates" in the form of XOR, XNOR, and MUX within the netlist [19], [20]. The key gates facilitate the application of key inputs to the design through which normal functionality within the obfuscated design can be retained. Only if the correct values ("0" or "1") are applied to each key input, the design functions properly. The correct key values are assumed to be supplied from a tamper-proof nonvolatile memory.

Whereas the aforementioned obfuscation techniques target the combinational circuit portion of a design, sequential logic locking techniques primarily focus on altering the state space, i.e., control path [21]–[23]. In one of the earliest sequential locking techniques, HARPOON [21], obfuscation of the state transition function was achieved by introducing new states that were previously "unreachable" due to the absence of "transition paths" [21]. The expanded FSM is modified such that the newly reachable states have to be traversed to reach the original state space that provides correct functionality. At startup, the design stays in the "obfuscated mode" where the circuit is nonfunctional and the user applies the correct input sequence to traverse a specific state transition path that leads to the first state of the "normal mode" of operation [21].

Scan chains are DFT structures that could be used to leak secrets stored into scan-chain-enabled flip-flops [24]. Obfuscation techniques have been used to protect assets from getting leaked through such debug infrastructures. A method similar to test compression has been leveraged to make the observation of individual flip-flop responses through scanchain difficult. Locking through the scrambling of scan-chain values based on a key is also possible [25]. Security of scan chains is important for enabling IC metering. By integrating a scrambling block with the scan chain, the design house can ensure that the untrusted foundry can no longer identify a functional IC by observing the scan-chain responses [26], thereby preventing them from producing additional ICs without any knowledge of the design house.

Layout-level obfuscation

To prevent unauthorized use and malicious modification of a GDS-II database at the foundry, various layout-level obfuscation methods exist. Researchers have proposed "split manufacturing" techniques that rely on multiple fabrication facilities, one of which must be trusted. Typically, the expensive active layer and few lower metal layers, also known as the front-end-of-line (FEOL) layers that require advanced process technology, are outsourced to an untrusted foundry. The remaining upper-level metal layers, called back-end-of-line (BEOL) layers that require less advanced process technology, are fabricated at a trusted foundry [27]. By only providing a part of the connectivity information, this technique inherently prevents IP piracy at the untrusted foundry and prevents a judicious insertion of Trojans. The security of this approach could be maximized by routing the security critical wires such that (using upper-level metal layers) their connectivity information is never fully disclosed to an untrusted foundry. However, the aforementioned benefits come at a cost of maintaining a trusted foundry that could complete BEOL with additional challenges of foundry compatibility and wafer alignment issues that could significantly increase the percentage of faulty ICs.

The idea of split manufacturing can be leveraged for protecting ICs fabricated using 3-D and/or 2.5-D technology [28]. The design house could judiciously perform wire-lifting on a given layout to ensure that each gate in the FEOL layer of the design appears similar to several other gates. Since the lifted wires are fabricated as an individual layer in a trusted foundry, the attacker with the FEOL layer would be unable to identify the gates as well as the complete design. Finally, with the use of through-silicon-via (TSV) bond points in a normal 3-D IC design flow, the separately fabricated layers can be assembled to produce the final IC.

Designs fabricated in 2.5-D IC technology can be protected from foundry attacks by partitioning the gate-level representation of the design and connecting them via an interposer layer [29]. Multiple partitions could be fabricated at an untrusted foundry while fabricating the interposer layer at a trusted one. However, the issue of maintaining a trusted foundry with a potential hit on the IC yield remains. Besides, area and delay overhead are impacted significantly due to gateswapping and wire-rerouting steps needed for obfuscation.

Programmable standard cells with a generic structure could be used as "camouflaged" gates. By observing the layout of camouflaged cells, it is hard to deduce their functionality (e.g., NAND, NOR, or XOR) [30]. If such configurable cells are placed in strategic locations, the design intent can no longer be inferred from a layout. Besides, unless the cells are programmed to perform the desired operation, the design stays functionally locked.

Instruction set obfuscation

A range of security benefits could be obtained by obfuscating the instruction set architecture (ISA) of a microprocessor. The ISA could be obfuscated through modifications in both hardware and software. For example, the instruction code could be scrambled during compilation and descrambled prior to execution to produce the original code using a pseudorandom numbers generator in hardware [31]. Such obfuscation prevents the malicious code that is not obfuscated with the appropriate pseudorandom number. However, it may not be suitable for protecting the hardware IP from cloning as the IP can be reused by knowing the hardware-generated pseudorandom number and obfuscating the software codes accordingly. Instruction XOR-ing with a secret key through hardware-level support also provides a similar benefit of malware prevention [32]. Insertion of junk bytes within the code can be used to prevent automatic disassembly of the program [33]. Fyrbiak et al. [34] provide statistical metrics to evaluate the advantages of both hardware- and software-level obfuscation to thwart RE of software code and other software-level attacks. A similar work takes advantage of architectural diversity implemented through hardware-level modification to prevent the propagation of malware by eliminating a break-one-break-all scenario [35].

PCB and FPGA obfuscation

The supply chain of PCB also involves untrusted parties where the PCB layout is prone to piracy and malicious modification. After deployment, the PCB could be reverse engineered or tampered by the end users. Obfuscation of the PCB design with the inclusion of a configurable permutation block could prevent many of the aforementioned attacks. The permutation block could be implemented with a complex programmable logic device (CPLD) or an FPGA that takes a set of critical interconnects (e.g., data bus) and apply a key-based permutation such that only in the presence of a correct key and the correct configuration file for the programmable logic, the block would render the correct functionality in the PCB [36].

To protect the FPGA bitstream from piracy, RE, and in-field alteration, obfuscation techniques can be applied in different ways. Architectural diversity can be introduced by augmenting configurable resources in the FPGA with key inputs [37]. Both one-time programmable physical and timevariant logical keys can be used to prevent a diverse set of attacks. Bitstream configurations for these resources are obfuscated during compilation according to the permanent physical key unique to a particular FPGA and the logical key generated using the EDA tool instantaneously. The bitstream becomes functional after being programmed in the corresponding FPGA as the physical key is always present in the device through e-Fuses and the logical key is transported to the FPGA in a secure manner at startup. Obfuscation of bitstream is also possible for legacy FPGAs [38]. Underused lookup-tables (LUT) in the synthesized FPGA netlist can be modified to have one or more key inputs. If the key input is wrong, the LUT chooses a dummy function integrated in the unused LUT space. Hence, in the absence of the correct key, the bitstream becomes nonfunctional. To protect critical functions of cryptographic algorithms from in-field alteration, a dynamic obfuscation countermeasure can be used that replaces the critical function with random configuration and takes advantage of dynamic reconfiguration to reprogram the correct configuration after the altered algorithm is loaded onto the FPGA [39]. This prevents an attacker from locating and modifying the critical function through static analysis of the bitstream.

Obfuscation process and metrics

Even though the security goals and application abstraction of various obfuscation processes are similar, the corresponding techniques of applying obfuscation vary significantly. We discuss various gate-level and FPGA bitstream obfuscation processes in detail.

Combinational logic locking

Combinational logic-locking techniques try to obscure the actual functional and structural behavior of a gate-level design with the insertion of additional logic gates called "key gates" [20], [40], [41]. These key gates could be XOR/XNOR [20], [40], [42], AND/OR [41], MUXes [40], or certain combination of such gates [43]. Unless the correct key is applied to the added gates, the functionality is locked. Besides, after resynthesis of the design, the obfuscated netlist experiences structural transformations due to the logic optimization performed by the synthesis tool. Oftentimes, the structural transformations are not localized to the key gate area and propagate to a broader part of the original design, which eventually helps to hide the design intent. Since the obfuscated design requires the correct keys for all the key gates, to use the obfuscated IP, one must know the correct key values. The combinational logic-locking technique is illustrated in Figure 5 with an example. The original gate-level design in Figure 5a with inputs A, B, C, and D is obfuscated by introducing three key gates in green color (Figure 5b and c). For both designs, input lines $K1$, $K2$, and $K3$ are *key inputs* that are connected to key gates (XOR gates and MUXes). If the correct key values ($K1 = 0$, $K2 = 1$, and $K3 = 0$) are applied, the obfuscated designs produce correct functional behavior. If any of the key bits are incorrect, the designs produce wrong outputs. For MUX-based logic locking, a second input is integrated from a different part of the design (dotted lines), and the key input is applied to the selected line of the MUX.

Key gates can be inserted into random or strategic locations. Although random insertion of gates [20] ensures structural change across the design

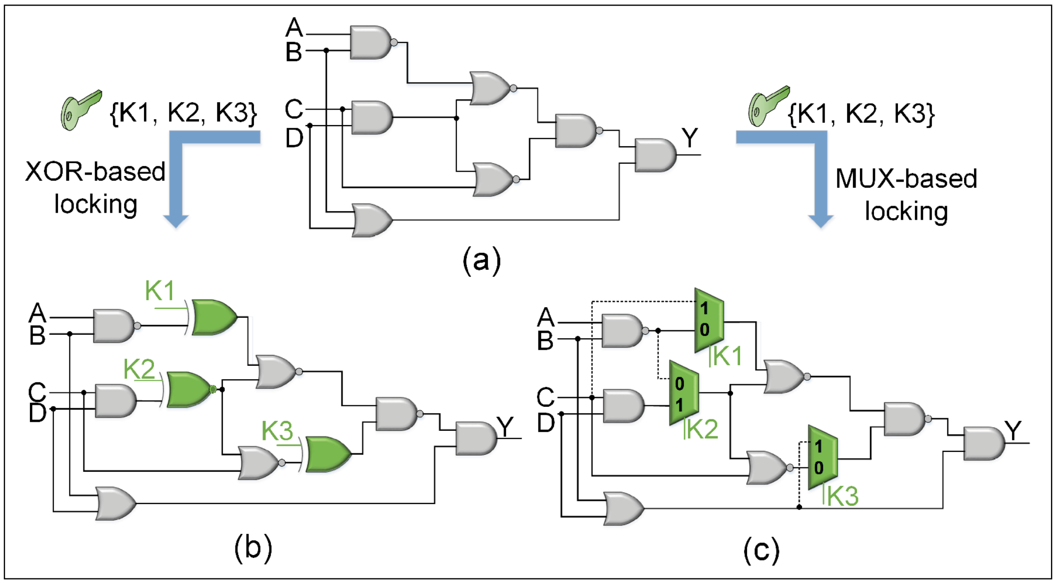


Figure 5. Example of combinational logic locking: (a) original gate-level design; (b) design after applying XOR- and/or XNOR-based logic locking with the insertion of three key gates (in green); (c) design after applying a MUX-based logic locking. The design in (b) and (c) functions as (a) if the correct key (i.e., *K*1 = 0, *K*2 = 1, and *K*3 = 0 for both) is applied.

due to the uniform distribution of key gates, it may not ensure high output corruption for certain combinations of wrong key inputs. This could happen due to the fact that changes in functionality in certain areas in the design may not propagate to the primary output. Therefore, under strict area overhead budget for obfuscation, more judicious insertion of key gates should be followed. Cone-based insertion of key gates accounts for fan-in and/or fan-out cone of a given net while sorting the candidate location for key insertion. Signals with a large number of inputs in their logic cones have the potential of improving the resistance against brute-force or fault-analysis-based attacks as we will learn in the "Attacks on obfuscation" section. Again, signals with large fan-out cones are likely to cause significant output corruption in the presence of a wrong key.

Sequential logic locking

HARPOON [21] is a sequential logic-locking technique that focuses on the modification of both sequential and combinational elements of the gate-level design. Since most practical designs contain a mix of combinational and sequential components, HARPOON provides a comprehensive security for such designs. The FSM of a sequential design can be represented with an STG as shown in Figure 6a. Although the number of all possible states for n state elements (i.e., flip-flops) is 2^n, in most designs, only a handful of the total states are used. Hence, there are significant number of unreachable states. These states are not reachable in the sense that there is no transition path from current valid states of the design to reach them. However, these unreachable states could be utilized for obfuscation purposes without even using additional flip-flops. As shown in Figure 6b, if these unreachable states (i.e., P_1, P_2, and P_3) are made reachable by integrating valid transitions (in red and green), and one of these states (i.e., P_1) is made the initial state of the obfuscated design, only a correct sequence of inputs to the obfuscated FSM (i.e., K_1, K_2, and K_3) would allow one to reach the original initial state S_0. This specific input sequence becomes the obfuscation key to initialize correct functionality.

At startup, the design begins from the new initial state and stays in the "preinitialization" state space (red states). If the correct sequence of input vectors is applied, the FSM reaches the "postinitialization" state space (blue states). No additional input

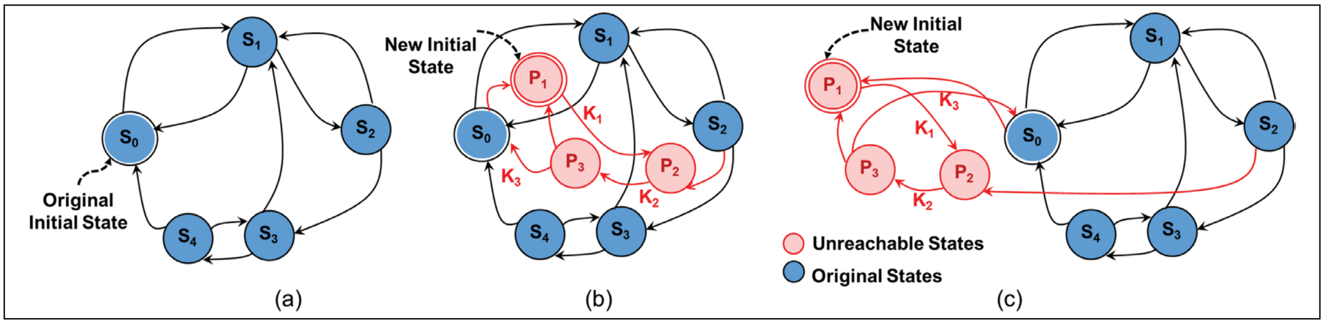


Figure 6. Overview of HARPOON. (a) Original FSM with five states. (b) Three unreachable states (in red) are made reachable by modifying the FSM. Transitions from new states are arranged such that a valid path exists to the original initial state S_0 from the new initial state P_1. (c) For obfuscating large designs the new state space could be integrated as a separate FSM.

pins are required for supplying the key, as only the existing primary input pins to the FSM are used. For a design containing a large and complex FSM with only its gate-level design available, the analysis of available unreachable states and the modification of the design to integrate transition paths to make them reachable could be challenging to automate. Therefore, such key application FSMs could be added by including only one or few state elements that would significantly “blow up” the usable statespace. Besides, with the inclusion of new flip-flop, a new set of states could be easily identified that are not used in the existing design. These states can now be modified to have desired transition paths that eventually lead to the original FSM’s initial state. Integration of the key application FSM as a separate state machine is shown in Figure 6c. This key application FSM is alternatively known as the obfuscation FSM (OFSM). Even though these two FSMs appear disjointed, with the inclusion of several dummy paths from the original FSM to OFSM, these two state machines would appear as the one in Figure 6b. To maximize output corruption under any wrong key and structural obfuscation of purely combinational segments, “modification cells” are introduced as well. These cells are combinational logic units that are kept enabled by the OFSM, unless the correct sequence of keys is applied. When enabled, these cells alter the functionality as done in the combinational logic-locking-based technique. However, instead of using a single key-gate, each modification cell may contain a combination of gates to provide better structural diversity.

Advantage in preventing Trojan attacks: HARPOON provides protection against hardware Trojan insertion at an untrusted foundry and a DFT facility. Hardware Trojan insertion often requires a judicious selection of rarely transitioning signals that could be switched to its rare value at a given instance if specific input patterns are applied to the design. However, in obfuscated state space, understanding the rareness of the internal signals during the normal mode of operation is not possible since the design is functionally locked. Hence, inserted Trojans could get accidentally triggered during logic testing. Besides, as shown in Figure 7, hardware Trojans that only activate during the obfuscated mode of operation are considered invalid. Since the design does not function properly in this mode, any leakage or corruption of assets would not be a valid attack.

FPGA bitstream obfuscation

FPGA configuration file or bitstream is prone to piracy, RE, and tampering. To provide protection against these attacks, various obfuscation-based solutions exist for FPGAs. Even though the FPGA architecture varies across different series, all FPGAs of the

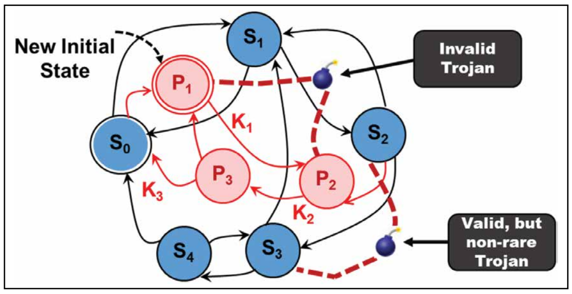


Figure 7. HARPOON improves the resistance against Trojan insertion by hiding the valid states and rareness of the nodes during the normal mode of operation.

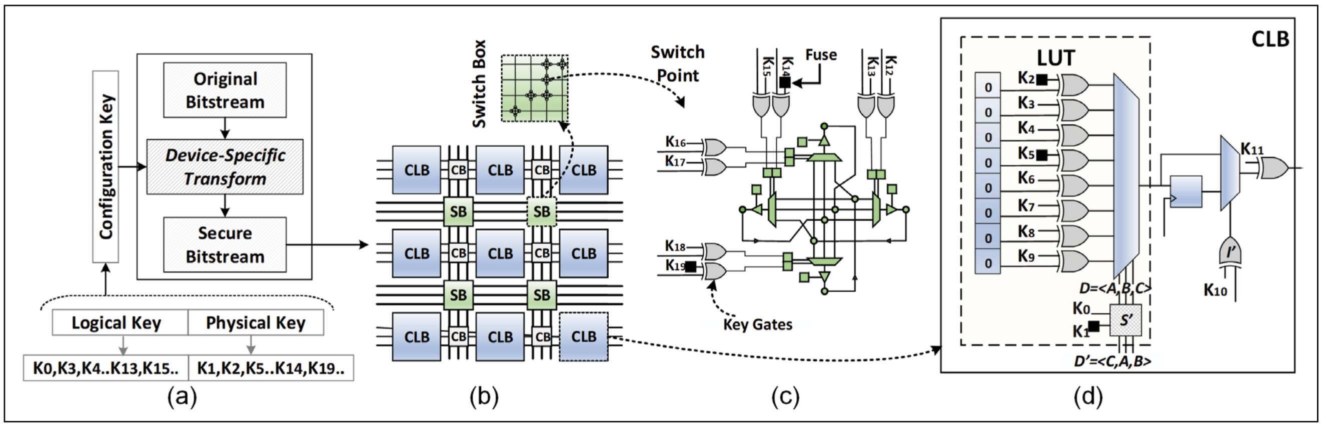


Figure 8. Overview of the bitstream obfuscation method using key augmented FPGA architecture: (a) two-part (logical and physical) key is used to perform a device-specific obfuscation of the bitstream that locks it for that particular device architecture; (b) obfuscated bitstream is mapped to appropriate FPGA device; and (c) and (d) internal FPGA hardware resources such as switchbox and configurable logic blocks (CLBs) are augmented with key gates which implements the inverse transform using logical and physical keys to make the bitstream functional [37].

same series have identical architecture. Architectural diversity in FPGAs can enable security against different attacks on the bitstream [37]. As illustrated in Figure 8a, the configuration file can be obfuscated during/after bitstream generation using a key. To reverse the obfuscation during operation, the FPGA architecture is augmented with key gates as shown in Figure 8c and d. The augmented key gates connect to existing FPGA resources (LUT, switch boxes, connection boxes, etc.) such that the bitstream functions in an unobfuscated form if the correct key is applied. Using such key gates, the FPGA architecture could be diversified per device. The FPGAs of a given series would be fabricated using the same architectural design, but part of the key gates of each device would be permanently programmed with a different "physical key" using the e-Fuses. The presence of a physical key makes the bitstream "node-locked," which prevents an attacker from using a bitstream in any device. Although some of the key gates will obtain their keys from such e-Fuses, the rest would receive "logical keys" during bitstream programming (via JTAG or battery-backed BRAM). For each compilation of the bitstream, a different logical key is used to prevent known design attacks that reveal mapping rules of the configuration file. Incorporation of logical and physical keys together makes it infeasible for an attacker to create a generic bitstream RE tool applicable to any FPGA. Besides, intelligent modification of the contents also becomes extremely difficult due to lack of knowledge regarding the mapping rule. Finally, even if the physical key for one device is compromised through destructive RE, other devices with a different physical key stay secure. Since unique per device identifier (e.g., "Device DNA") is already maintained by FPGA vendors, the logistical challenge of maintaining a per device physical key can be easily met. Besides, the timevariant logical key can be readily integrated by introducing simple modifications to the existing EDA tool. The complete bitstream obfuscation process can simply be integrated as an add-on to the existing synthesis tools as shown in Figure 8a.

Although the previous method of bitstream obfuscation requires modification to the existing FPGA architecture, obfuscation is possible for legacy FPGAs as well without requiring such changes [38]. It can be achieved by taking advantage of the FPGA dark silicon, i.e., unused LUT resources already available in these FPGAs. The majority of the LUTs in FPGA mapped designs do no use all possible inputs available in the physical LUT structure. Although in many FPGAs, an LUT can be configured with up to a six input function, designs mapped for such devices are unable to completely utilize most of them. These unused LUT inputs result in unused memory spaces that are identified as FPGA dark silicon. As illustrated in Figure 9, a two-input function is mapped onto an LUT that can take up to a three-input function. Hence, the empty memory is filled with a dummy function, and a key input dictates if the original or the dummy function would be selected.

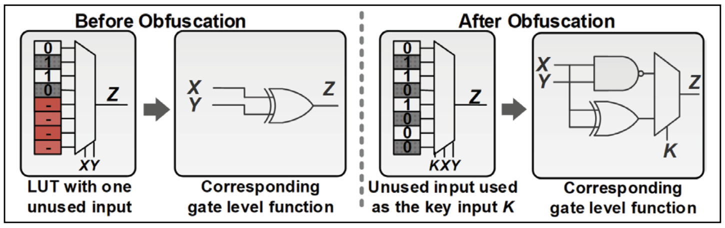

Before Obfuscation
After Obfuscation
LUT with one unused input
Corresponding gate level function
Unused input used as the key input K
Corresponding gate level function
X
Y
Z
K
XY
KXY

Figure 9. Two-input function (XOR gate) mapped onto an LUT that can take up to three inputs is transformed to have an additional input *K*. Depending on the key input the function now works as the original (XOR) or a dummy function (NAND) [38].

Therefore, without using any additional LUT, obfuscation goals can be achieved. Since the majority of LUTs have such unused inputs for most designs, the design can be functionally locked by integrating a key in each of these LUTs. This technique also hides the design intent with the inclusion of a dummy function. Hence, the attacker has to choose between two different functions for each obfuscated LUT. In this approach, the obfuscated design could be node-locked if a device-specific physical unclonable function-generated key is used for obfuscation, making the bitstream functional only when mapped onto a specific FPGA device. The security of this approach has been quantified with the difficulty of bruteforcing the key and the number of trials needed for choosing the correct function for all obfuscated LUTs. The feasibility of hiding the design intent by including dummy functions can be augmented to hide a critical logic in the design [44].

Obfuscation metric

The goal of obfuscation is to improve resistance against piracy, RE, and malicious modification. Therefore, the level of protection against these attacks has to be quantified using an appropriate security metric. Such a metric would allow designers to fine-tune the security versus design and/or test cost, depending on the target application and threat model. Below we present some of the design properties and security metrics used for evaluating various obfuscation techniques.

(i) *Functional correctness*: Upon the application of the correct key, any functional locking technique should provide the original functionality. Retaining correct functionality in an unlocked design having sequential elements with loops may require additional arrangement (e.g., resetting flip-flops) as such loops may retain faulty values after operating in the nonfunctional mode. However, for a purely combinational design obfuscated with combinational locking, original functionality is obtained instantaneously after applying the correct key.

(ii) *Entropy and Hamming distance*: Certain attacks try to take advantage of weak obfuscation that does not corrupt the functionality significantly between the unlocked and a locked states. Such a weakness could be leveraged by an attacker utilizing previous input–output combinations to guess the correct output. To prevent such attacks, the entropy of the output response of a locked design should be maximized. To achieve high output entropy, the Hamming distance between output responses of the same obfuscated design under correct and incorrect key for various inputs should ideally be 50% [40].

(iii) *Entanglement*: Since the adversary can alter the gate-level design obfuscated with combinational logic locking, an attempt to remove the key gates to obtain the original design is very likely. However, such removal is feasible only if the attacker can assume the impact of the key gate on the obfuscated net in the presence of a correct (or wrong) key input. For instance, assuming that the key gates are always inserted to invert the response of an obfuscated signal, the correct key input could be guessed by observing if the key gate is XOR or XNOR. However, due to resynthesis of the obfuscated design, the gates around obfuscated areas are often rearranged with a different combination of gates, making such observation very difficult. Nevertheless, the design should be assessed against such attacks after obfuscation.

(iv) *Output corruptibility*: Although the Hamming distance between output responses of the design in the locked and unlocked states is an important parameter, this high corruptibility of output should be consistent for all possible patterns of wrong keys and for each wrong key pattern, a large number of input patterns should produce wrong outputs. If for a given faulty key pattern, only a few input patterns create a faulty output, those input patterns could be bypassed [45]. Using the knowledge of all possible correct input–output pairs from an unlocked IC purchased from the market, a “bypass attack” on the obfuscated netlist could be done by connecting an additional comparator-like logic with the primary input that detects the input pattern causing a faulty output for a specific faulty key pattern (yet provides correct output for most input patterns). The comparator logic will trigger a payload circuit connected to the primary output that fixes the output to a correct one.

(v) *Resistance against key-guessing attacks*: Various key-guessing attacks on combinational logic locking can be used to reveal the key using observation from input–output pairs for different key patterns. In addition to that, attackers could also use the knowledge of correct output for any input by obtaining an unlocked chip from the market. Key-guessing attacks become easier when the key gates are placed in such a way that the number of input–output pair observation required is small. Ideally, the number of required observations should increase exponentially with key size for preventing such attacks.

(vi) *Failing verification points*: This metric is useful for assessing the structural modification after state-space alteration [21]. To calculate this, the gate-level design and associated library of the original and obfuscated design are provided to an equivalence checking software tool. The tool compares the two designs based on a set of verification points [e.g., Flip-Flop (FF) input or primary outputs]. If a majority of the total verification points fail to match, it indicates a good structural obfuscation.

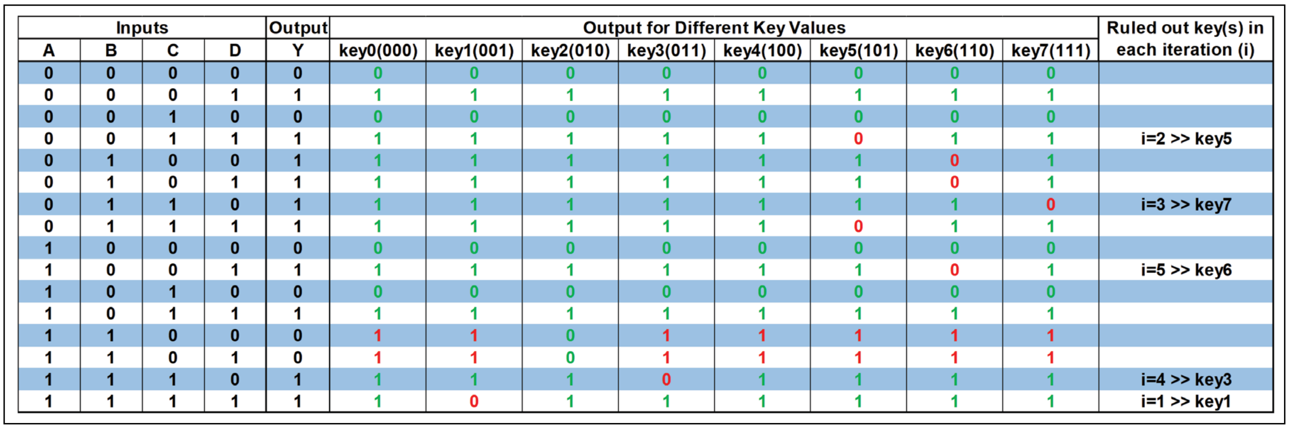
(vii) *Bitstream Hamming distance*: Inter- and intra-bitstream hamming distances can be used for assessing the quality of FPGA bitstream obfuscation [37], [38]. The inter-bitstream distance can be calculated as the average distance between comparable LUT contents (i.e., frames) in the original and obfuscated configurations. The intra-bitstream distance can be calculated as the average pairwise distance between LUT contents within a obfuscated bitstream file.

Attacks on obfuscation

A number of attack methodologies exist that can compromise different forms of combinational logic-locking techniques. The primary target of these attacks is to reveal the obfuscation key. Structural deobfuscation-based attack has also been performed to revert the original structure from the obfuscated one. A brute-force attack on any obfuscation technique simply tries all possible key values, unless the obfuscated design starts working as a functional one. For the comparison purposes an unlocked IC that produces correct outputs for all possible inputs can be obtained from the market. For combinational logic locking, if N_l different key bits are present, a 2^{N_l} different trials could be needed in the worst case. For sequential locking, such as HARPOON, an N_s bit key can be applied as i bit inputs to the FSM for c clock cycles such that $N_s = i*c$. The required number of worst-case attempts to brute force would be $(2^i)^c = 2^{(i*c)} = 2^{Ns}$. Since the required number of attempts is an exponential function of the key size, a reasonably large key size (e.g., 256 bits or larger) would be good enough to render a brute-force attempt computationally infeasible. Note that an obfuscated design could produce a correct response for a few numbers of input vectors even when an incorrect key is applied. Hence, to ensure that a given key is correct, the design has to be compared with the unlocked one for a large number of input patterns. This further increases the difficulty of brute-force attacks.

Boolean SAT attacks: The SAT attack [6] is a useful technique to reveal the correct key from a design obfuscated by combinational logic locking. SAT attack can be applied by modeling it as a Boolean SAT problem, and using heuristic-based algorithms (in the form of their efficient software implementations) to solve it. This attack requires an unlocked IC or golden functional outputs. Compared to the brute-force technique, SAT attack is very fast as it reduces the search space for the correct key. It eliminates incorrect key values by using the idea of distinguishing input patterns (DIPs). DIP can be defined as the input value for which at least two different key values produce dissimilar responses. A single DIP could eliminate more than one incorrect key values in some instances. Unless any specific heuristic is used by the SAT engine, the SAT attack randomly chooses the DIPs. If a single DIP eliminates a large number of invalid keys simultaneously, the number of patterns required to reveal the correct key would be small.

To better understand the idea of SAT attack, let us consider the obfuscated circuit in Figure 5b. Along with the obfuscated netlist, the attacker would require an unlocked IC (or golden functional response) that can be purchased from the open market once the product is available. The attack begins with the identification of input–output pairs from the functional IC. Then, a SAT solver tool will randomly choose the DIPs and will try to rule out the wrong key values. Figure 10 shows the correct functional outputs for different input patterns. It also shows the outputs for different combinations of input and key values for the circuit in Figure 5b. The eight possible key values for the three-bit key are denoted as $key0$, $key1$,..., $key7$, where $key = \{K1, K2, K3\}$ and $key0 = \{0, 0, 0\}$, $key1 = \{0, 0, 1\}$, ..., $key7 = \{1, 1, 1\}$, and so on. The outputs for different key



Inputs				Output	Output for Different Key Values								Ruled out key(s) in
A	B	C	D	Y	key0(000)	key1(001)	key2(010)	key3(011)	key4(100)	key5(101)	key6(110)	key7(111)	each iteration (i)
0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	1	1	1	1	1	1	1	1	1	1	
0	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	1	1	1	1	1	1	1	1	0	1	1	i=2 >> key5
0	1	0	0	1	1	1	1	1	1	1	0	1	
0	1	0	1	1	1	1	1	1	1	1	0	1	
0	1	1	0	1	1	1	1	1	1	1	1	0	i=3 >> key7
0	1	1	1	1	1	1	1	1	1	0	1	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	1	1	1	1	1	1	1	1	0	1	i=5 >> key6
1	0	1	0	0	0	0	0	0	0	0	0	0	
1	0	1	1	1	1	1	1	1	1	1	1	1	
1	1	0	0	0	1	1	0	1	1	1	1	1	
1	1	0	1	0	1	1	0	1	1	1	1	1	
1	1	1	0	1	1	1	1	0	1	1	1	1	i=4 >> key3
1	1	1	1	1	1	0	1	1	1	1	1	1	i=1 >> key1

Figure 10. Elimination of invalid key values under SAT attack to reduce the key search space.

values are marked in green and red colors. Red and green denote wrong and correct outputs, respectively. In the first iteration, we assume that the DIP is chosen to be 1111. In this iteration, the SAT solver finds *key*1 as a wrong key combination since the output for this key combination does not match with the functionally correct output. Hence, it removes *key*1 from the list of valid key values. In the second iteration, 0011 is chosen as the DIP and *key*5 is ruled out in the same way as in iteration-1. The SAT solver tool goes through multiple iterations to gradually rule out wrong key values. In this example, after five iterations, five key values (*key*1, *key*3, *key*5, *key*6, and *key*7) are ruled out, and only three key combinations are left from which an attacker can easily find the correct key by trying them in the obfuscated netlist. However, if the SAT solver considers 1100 or 1101 as the DIP, all the incorrect key values will be ruled out in just one iteration. In a practical scenario, the SAT solver does not consider each of the key bits. It chooses a part of the key bits, e.g., for the above example, two-key bits will be set to a known value considering other ones as "don't care." At this point, if the SAT solver finds a DIP, it will just rule out those values of key bits to be incorrect and in this way, the correct key search space can be significantly minimized.

Key sensitizing attacks: This attack uses an unlocked IC obtained from the market and takes advantage of the automatic test pattern generation (ATPG) tools that are commonly used for generating patterns to sensitize faults in the design. Vector generation for fault sensitization is a well-researched task that has been automated by commercial CAD tools available from various vendors (e.g., Synopsys, Cadence, and Mentor). The goal of such a pattern is to propagate the impact of a fault to the primary output. If patterns are not carefully generated, such faults may never propagate to the output and stay unobserved throughout the testing process. Using the same principle of pattern generation, sensitizing vectors could be generated for a gate-level design to propagate the value of a key-bit to the primary output [46]. The key-bit can be sensitized without masking or corruption if other key-bits have no interference in the sensitized path. By observing the output, it is possible to determine the value of the sensitized keybits. Once an attacker determines an input pattern, which can sensitize a key-bit to the output without interference, he/she can then apply it to the functional IC. At this point, this pattern will be used to sensitize the correct value of the key-bit to an output. Key sensitizing attack (KSA) has been successfully mounted on certain combinational locking approaches.

Structural analysis using machine learning: A structural attack on combinational logic locking called structural analysis using machine learning (SAIL) can reconstruct the obfuscated gate-level design using machine learning [47]. Since key-gate-inserted designs are vulnerable to the removal of key gates and key guessing attacks, designs are synthesized to ensure that the obfuscated design is structurally modified. However, even after synthesis, changes are localized around the key gates and sparse (nonoverlapping). Therefore, these deterministic changes can be learned and any resynthesized obfuscated design can be retrieved to its presynthesized version. As shown in Figure 11, after obfuscation, an original design *C* is converted to *O*. After synthesis, *O* experiences localized structural changes around its key gate area and transforms to *S*. A machine learning model that is trained to identify the possible mapping rules of such transformations can be used to retrieve *O* from *S*, from which *C* can be obtained with removal or key-guessing attacks. This attack has noticeable advantages over SAT-based attacks for the following reasons. First, unlike SAT, it does not require golden responses from an unlocked IC; second, while SAIL can be readily applied to both combinational and sequential designs, SAT is difficult to apply on sequential ones; third, SAT attack fails for designs with certain functions, e.g., multiplier; however, SAIL has no dependency on the underlying boolean functions of the design; and fourth, for large designs obfuscated with large key size, SAT suffers from the scalability issue, whereas SAIL is more scalable since the computation time primarily depends on the number of key gate areas to reconstruct.

The major steps to apply the SAIL attack is presented in Figure 12. From a given obfuscated design, a change prediction model is first created. It is a trained model that understands the changes in key gate localities due to the synthesis process. The obfuscated design itself is reobfuscated and resynthesized to generate example changes to train from. The localities before and after logic synthesis are represented as subgraphs and provided to the model for training. Once the model is trained, from the actual obfuscated design, subgraphs of key gate localities are identified. Key gate localities that do not have any change, i.e., the key gates appear to be intact, are ignored and the rest of the subgraphs are provided to the change prediction model. The model generates the reconstructed subgraph and the presynthesized design is retrieved.

Attacks specific to HARPOON: Even though the aforementioned attacks (SAT, KSA, removal, etc.) have not been applied to HARPOON, there is an inherent difficulty in applying these attacks on sequential locking techniques. Application of SAT on HARPOON would require unrolling of the design, which is very challenging due to the presence of nested feedback loops in the design. The identification of the required number of unrolling cycles is also a challenge as the attacker does not know the number of key sequences. KSA could be used for attacking the modification cells. However, since the modification cells in HARPOON are not connected to the primary inputs (like key gates in combinational logic locking), it is difficult to separate these cells in the design. However, by analyzing the fan-out of the design, an attacker could try to identify the separately included OFSM responsible for driving a large number of modification cells. Again, these enabled signals could be observed after identification

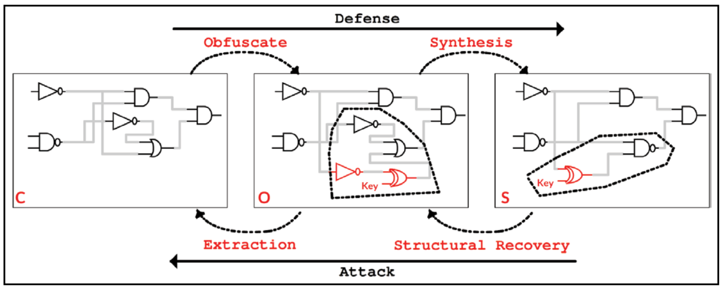


Figure 11. Design changes due to the synthesis process on localized areas around key gates of the obfuscated design can be learned using machine learning. The trained model can be used to retrieve the presynthesized version of the obfuscated design which does not hide the original design intent [47].

to understand if they stay in a particular state during the obfuscated mode to reveal the unlocking value (the inverse one).

A security analysis of various sequential locking schemes has been presented in [48]. To extract the unlocking key sequence of HARPOON, the authors first obtained the STG from the locked netlist. Since HARPOON does not generate any state transition paths from the original state space to the preinitialization state space, the two FSMs can be isolated by observing the clustered state spaces in the STG of the overall design. Once the states belonging to the original FSM are identified, it is straightforward to find the input sequence to reach them. Furthermore, if the initial state of the original FSM is known, the preinitialization state space can be completely bypassed by patching the desired state bits as the initial bits of state elements [48]. The applicabilities of various attacks on combinational locking and HARPOON are compared in Figure 13.

Attacks on the key: More recently the researchers have emphasized that the security of the IP is not only limited to the robustness of the obfuscation technique. Rather, snooping the transfer of the unlocking

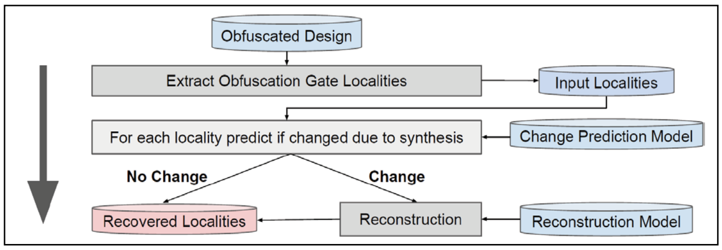
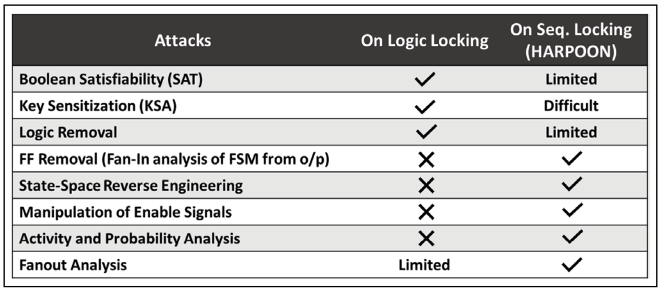


Figure 12. Overview of major steps in the SAIL attack flow [47].



Attacks	On Logic Locking	On Seq. Locking (HARPOON)
Boolean Satisfiability (SAT)	✓	Limited
Key Sensitization (KSA)	✓	Difficult
Logic Removal	✓	Limited
FF Removal (Fan-In analysis of FSM from o/p)	✗	✓
State-Space Reverse Engineering	✗	✓
Manipulation of Enable Signals	✗	✓
Activity and Probability Analysis	✗	✓
Fanout Analysis	Limited	✓

Figure 13. Comparison between combinational logic locking and HARPOON with respect to various attacks.

key between the source of the key and the key-gates through invasive and noninvasive methods is viable attack vectors [49], [50]. For instance, Rahman et al. [50] demonstrate the feasibility of extracting the obfuscation key by tapping the data transfer between the memory containing the key and the key-gates using optical probing. A design obfuscated with a combinational locking method mapped onto an FPGA was targeted in the optical probing experiment. In the presence of such a probing facility, the security of any logic-locking framework gets reduced to the difficulty of finding the signals of interest (signals carrying the key values) in the IC, unless preventive measures against the probing attempt are incorporated [49].

Countermeasures to attacks: To make an obfuscated design resistant to key-guessing attacks, key-gates should be inserted in such a way that an attacker cannot propagate the output of a single key-gate [51], i.e., the observed output should be a function of multiple key-gates. Key-gates can be inserted judiciously so that they block each other's sensitization path. This method forms a "clique." With the increase in the size of the clique, the attacker's effort will also increase. To achieve resistance against SAT attacks, some cryptographic primitives or SAT-resistant logic circuits called anti-SAT blocks can be incorporated into the design. The idea is to prevent an SAT solver from efficiently minimizing the keyspace, i.e., forcing it to go through an exponentially large number of iterations. Anti-SAT block is a small circuit which takes inputs from internal nodes of the original circuit along with some key inputs, and it exponentially increases the key search space. It corresponds to a high level of difficulty for an SAT solver to find the correct key values.

Stripped functionality logic locking (SFLL) was proposed as a secure combinational logic-locking technique that is resistant to attacks, including SAT, sensitization, and removal [52]. This technique modifies the design by removing a part of its functionality such that it produces wrong output only for selected input patterns. Additional logic is integrated to the modified design such that the output for the selected input vectors is correct only when the unlocking key is preset. However, one version of SFLL, known as SFLL-hd, has been compromised through the structural analysis of the obfuscated netlist that reveals the protected pattern(s) from which the secret key is recovered [53].

Timing-based obfuscation has been explored to thwart Oracle-guided attacks such as SAT and KSA. In a delay logic locking (DLL), gates with key-controlled delay values are inserted along with the regular XOR- and/or XNOR-based key gates [54]. Incorrect key input to the tunable delay key-gate (TDK) causes a setup/hold time violation and corrupts the output of the IP. Since delay is not a logical behavior, it is hard to translate the TDK behavior to a SAT assignment problem that is understandable by SAT. However, Azar et al. [55] were able to deploy the SAT modulo theory (SMT)-based attack to model nonfunctional properties like delay. The SMT attack translates the obfuscated netlist to its graph representation and uses a graph-theory solver to reveal the key. Zhang et al. [56] present a timing camouflage technique where some of the state elements in the design are removed intentionally to introduce unconventional timing paths in the circuit that cannot be resolved using SAT attack. However, Li et al. [57] demonstrated the feasibility of applying SAT attack to decamouflage such timing-based strategies through a novel transformation procedure.

To thwart probing attacks that could leak the key from an unlocked IC, various defensive solutions can be integrated. Optical probing can be detected with the integration of an optically active layer that is opaque to photoemission from silicon devices and can detect any modification to the backside of the IC [58], [59]. Contact-based probing can be detected and circumvented using active shielding [60] that is placed on the top layer(s) of the IC. The signals being transmitted on this layer are monitored at runtime to detect modification of the shield by an attacker to create space for probe insertion. The probe attempt detector (PAD) is another solution that detects a probing attempt by measuring the extra capacitance introduced by the probe on security critical wires [61]. Routing critical signals to lower metal layers is suggested as well to make contact-based probing attacks more challenging [62].

The field of hardware obfuscation is advancing fast, as various novel techniques are proposed and inventive methods of attacks are studied to challenge them. We have emphasized the need for hardware IP obfuscation by discussing the threats present in the distributed semiconductor supply chain. An overview of various obfuscation techniques applicable to different stages of the design cycle is presented. We directed our discussion on a few prime obfuscation methods and provided a comparative analysis of the existing attacks in compromising them. As more ingenious methods to weaken these obfuscation techniques emerge, the need for quantifying the resistance against all known attacks is becoming an important requirement. A metric-driven analysis of strong and weak features of structures within the design (e.g., gates and connectivity) is needed at design time to employ efficient obfuscation under minimal design overhead. Furthermore, establishing protocols to enable secure application, storage, and management of the obfuscation key in the field or third-party facility requires further research. ■

Acknowledgments

This work was funded in part by the National Science Foundation (NSF) under Grant 1603483. We appreciate the valuable input from our colleague Parker Difuntorum to improve the paper.

References

[1] Semico Research. (2018). *Semiconductor IP Market to Exceed $8 Billion by 2020*. [Online]. Available: https://semico.com/content/

[2] CNN. (2018). *How Much Has the US Lost From China's IP Theft?* [Online]. Available: https://money.cnn.com/2018/03/23/technology/chinaus-trump-tariffs-ip-theft/index.html

[3] P. Mishra, S. Bhunia, and M. Tehranipoor, *Hardware IP Security and Trust*. Springer International Publishing, 2017.

[4] D. Forte, S. Bhunia, and M. M. Tehranipoor, *Hardware Protection Through Obfuscation*. Springer International Publishing, 2017.

[5] M. Yasin et al., "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, early access, Aug. 21, 2017, doi: 10.1109/TETC.2017.2740364.

[6] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[7] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[8] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Des. Test. Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.

[9] R. Torrance and D. James, "The state-of-the-art in IC reverse engineering," in *Cryptographic Hardware and Embedded Systems—CHES*. Berlin, Heidelberg: Springer, 2009, pp. 363–381.

[10] J.-B. Note and É. Rannaud, "From the bitstream to the netlist," in *Proc. 16th Int. ACM/SIGDA Symp. Field Program. Gate Arrays (FPGA)*, vol. 8, 2008, p. 264.

[11] A. Moradi et al., "On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from Xilinx virtex-II FPGAs," in *Proc. 18th ACM Conf. Comput. Commun. Secur. (CCS)*, 2011, pp. 111–124.

[12] J. Robertson and M. Riley. (Oct. 2018). *The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies*. [Online]. Available: https://www.bloomberg.com/news/features/2018-10-04/the-bighack-how-china-used-a-tiny-chip-to-infiltrate-america-s-top-companies

[13] P. Swierczynski et al., "FPGA Trojans through detecting and weakening of cryptographic primitives," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1236–1249, Aug. 2015.

[14] R. S. Chakraborty et al., "Hardware Trojan insertion by direct modification of FPGA configuration bitstream," *IEEE Design Test. Comput.*, vol. 30, no. 2, pp. 45–54, Apr. 2013.

[15] A. Chhotaray et al., "Standardizing bad cryptographic practice: A teardown of the IEEE standard for protecting electronic-design intellectual property," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1533–1546.

[16] R. S. Chakraborty and S. Bhunia, "RTL hardware IP protection using key-based control and data flow obfuscation," in *Proc. 23rd Int. Conf. VLSI Design*, Jan. 2010, pp. 405–410.

[17] A. R. Desai et al., "Interlocking obfuscation for anti-tamper hardware," in *Proc. 8th Annu. Cyber Secur. Inf. Intell. Res. Workshop (CSIIRW)*, 2013, p. 8.

[18] M. Brzozowski and V. N. Yarmolik, "Obfuscation as intellectual rights protection in VHDL language," in *Proc. 6th Int. Conf. Comput. Inf. Syst. Ind. Manage. Appl. (CISIM)*, Jun. 2007, pp. 337–340.

[19] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, Oct. 2010.

[20] J. Rajendran et al., “Logic encryption: A fault analysis perspective,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2012, pp. 953–958.

[21] R. S. Chakraborty and S. Bhunia, “HARPOON: An obfuscation-based SoC design methodology for hardware protection,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1493–1502, Oct. 2009.

[22] J. Dofe and Q. Yu, “Novel dynamic state-deflection method for gate-level design obfuscation,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 273–285, Feb. 2018.

[23] Y. Alkabani and F. Koushanfar, “Active hardware metering for intellectual property protection and security,” in *Proc. USENIX Secur. Symp.*, 2007, pp. 291–306.

[24] B. Yang, K. Wu, and R. Karri, “Scan based side channel attack on dedicated hardware implementations of data encryption standard,” in *Proc. Int. Conf. Test*, 2004, pp. 339–344.

[25] J. Lee et al., “Securing designs against scan-based side-channel attacks,” *IEEE Trans. Dependable Secure Comput.*, vol. 4, no. 4, pp. 325–336, Oct. 2007.

[26] G. K. Contreras, M. T. Rahman, and M. Tehranipoor, “Secure split-test for preventing IC piracy by untrusted foundry and assembly,” in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFTS)*, Oct. 2013, pp. 196–203.

[27] K. Vaidyanathan et al., “Building trusted ICs using split fabrication,” in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust (HOST)*, May 2014, pp. 1–6.

[28] F. Imeson et al., “Securing computer hardware using 3D integrated circuit (IC) technology and split manufacturing for obfuscation,” in *Proc. USENIX Secur. Symp.*, 2013, pp. 495–510.

[29] Y. Xie, C. Bao, and A. Srivastava, “Security-aware design flow for 2.5D IC technology,” in *Proc. 5th Int. Workshop Trustworthy Embedded Devices (TrustED)*, 2015, pp. 31–38.

[30] J. Rajendran et al., “Security analysis of integrated circuit camouflaging,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 709–720.

[31] E. G. Barrantes et al., “Randomized instruction set emulation to disrupt binary code injection attacks,” in *Proc. 10th ACM Conf. Comput. Commun. Secur. (CCS)*, 2003, pp. 281–289.

[32] G. S. Kc, A. D. Keromytis, and V. Prevelakis, “Countering code-injection attacks with instruction-set randomization,” in *Proc. 10th ACM Conf. Comput. Commun. Secur. (CCS)*, 2003, pp. 272–280.

[33] C. Linn and S. Debray, “Obfuscation of executable code to improve resistance to static disassembly,” in *Proc. 10th ACM Conf. Comput. Commun. Secur. (CCS)*, 2003, pp. 290–299.

[34] M. Fyrbiak et al., “Hybrid obfuscation to protect against disclosure attacks on embedded microprocessors,” *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 307–321, Mar. 2018.

[35] R. Karam et al., “Mixed-granular architectural diversity for device security in the Internet of things,” in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Oct. 2017, pp. 73–78.

[36] Z. Guo et al., “Investigation of obfuscation-based anti-reverse engineering for printed circuit boards,” in *Proc. 52nd Annu. Design Autom. Conf. (DAC)*, 2015, p. 114.

[37] R. Karam et al., “MUTARCH: Architectural diversity for FPGA device and IP security,” in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 611–616.

[38] R. Karam et al., “Robust bitstream protection in FPGA-based systems through low-overhead obfuscation,” in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.

[39] P. Swierczynski et al., “Protecting against cryptographic Trojans in FPGAs,” in *Proc. IEEE 23rd Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, May 2015, pp. 151–154.

[40] J. Rajendran et al., “Fault analysis-based logic encryption,” *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 410–424, Feb. 2015.

[41] S. Dupuis et al., “A novel hardware logic encryption technique for thwarting illegal overproduction and hardware Trojans,” in *Proc. IEEE 20th Int. On-Line Test. Symp. (IOLTS)*, Jul. 2014, pp. 49–54.

[42] J. A. Roy, F. Koushanfar, and I. L. Markov, “EPIC: Ending piracy of integrated circuits,” in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1069–1074.

[43] Y.-W. Lee and N. A. Touba, “Improving logic obfuscation via logic cone analysis,” in *Proc. 16th Latin-American Test Symp. (LATS)*, Mar. 2015, pp. 1–6.

[44] T. Hoque et al., “Hidden in plaintext: An obfuscation-based countermeasure against FPGA bitstream tampering attacks,” *ACM Trans. Des. Autom. Electron. Syst. (TODAES)*, vol. 25, no. 1, p. 4, 2019.

[45] B. Shakya et al., “Chip editor: Leveraging circuit edit for logic obfuscation and trusted fabrication,” in *Proc. Int. Conf. Comput.-Aided Design*, 2016, p. 30.

[46] M. Yasin et al., “On improving the security of logic locking,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 9, pp. 1411–1424, Sep. 2016.

[47] P. Chakraborty, J. Cruz, and S. Bhunia, “SAIL: Machine learning guided structural analysis attack on hardware obfuscation,” 2018, *arXiv:1809.10743*. [Online]. Available: http://arxiv.org/abs/1809.10743

[48] M. Fyrbiak et al., “ On the difficulty of FSM-based hardware obfuscation,” *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 293–330, 2018.

[49] S. Engels, M. Hoffmann, and C. Paar, “The end of logic locking? A critical view on the security of logic locking,” IACR Cryptol. ePrint Arch., vol. 2019, p. 796, 2019.

[50] M. T. Rahman et al., “The key is left under the mat,” IACR Cryptol. ePrint Arch., Tech. Rep., 2020.

[51] J. Rajendran et al., “Security analysis of logic obfuscation,” in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 83–89.

[52] M. Yasin et al., “Provably-secure logic locking: From theory to practice,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1601–1618.

[53] F. Yang, M. Tang, and O. Sinanoglu, “Stripped functionality logic locking with hamming distance-based restore unit (SFLL-hd)—Unlocked,” *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 10, pp. 2778–2786, Oct. 2019.

[54] Y. Xie and A. Srivastava, “Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction,” in *Proc. 54th Annu. Design Autom. Conf. (DAC)*, 2017, p. 9.

[55] K. Z. Azar et al., “SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks,” *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, no. 1, pp. 97–122, 2019.

[56] G. L. Zhang et al., “TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing,” in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2018, pp. 91–96.

[57] M. Li et al., “TimingSAT: Decamouflaging timing-based logic obfuscation,” in *Proc. IEEE Int. Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[58] E. Amini et al., “Assessment of a chip backside protection,” *J. Hardw. Syst. Secur.*, vol. 2, no. 4, pp. 345–352, Dec. 2018.

[59] X. T. Ngo et al., “ Cryptographically secure shield for security IPs protection,” *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 354–360, Feb. 2017.

[60] S. Briais et al., “Random active shield,” in *Proc. Workshop Fault Diagnosis Tolerance Cryptography*, Sep. 2012, pp. 103–113.

[61] S. Manich, M. S. Wamser, and G. Sigl, “Detection of probing attempts in secure ICs,” in *Proc. IEEE Int. Symp. Hardware-Oriented Secur. Trust*, Jun. 2012, pp. 134–139.

[62] H. Wang et al., “Probing attacks on integrated circuits: Challenges and research opportunities,” *IEEE Des. Test. Comput.*, vol. 34, no. 5, pp. 63–71, Oct. 2017.

Tamzidul Hoque is pursuing a PhD in electrical and computer engineering (ECE) at the University of Florida, Gainesville, FL, under the supervision of Dr. Swarup Bhunia. He worked as a PhD intern within Cisco’s Advanced Security Research & Government Group, Research Triangle Park, NC. His research is focused on hardware security, with emphasis on hardware Trojan countermeasures and hardware intellectual property protection.

Rajat Subhra Chakraborty is an Associate Professor in the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur (IIT Kharagpur), India. His primary area of research is hardware security, VLSI design (especially low-power and robust design), digital watermarking, and digital forensics. He received a PhD in computer engineering from Case Western Reserve University, Cleveland, OH.

Swarup Bhunia is the Semmoto Endowed Professor of IoT at the Electrical and Computer Engineering (ECE) Department, University of Florida (UF), Gainesville, FL. He is currently serving as the Director of Warren B. Nelms Institute for the Connected World at UF. He received a PhD in electrical engineering from Purdue University, West Lafayette, IN.

■ Direct questions and comments about this article to Tamzidul Hoque, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611 USA; thoque@ufl.edu.