Chapter 1
Introduction to Hardware Obfuscation: Motivation, Methods and Evaluation

Bicky Shakya, Mark M. Tehranipoor, Swarup Bhunia and Domenic Forte

1.1 Introduction

While piracy of intellectual property (IP) relating to daily commodities such as clothing, medicine, and fashion items has had a long history, IP violation of technological assets, such as computer software and hardware, has become a recent albeit concerning problem. In April 2016, a Wisconsin grand jury slapped a $940 million penalty on Tata Consultancy Services for allegedly stealing Epic System Inc.'s healthcare database management software and incorporating it into its own products [1]. In 2013, two men, Jason Vuu and Glen Crissman, were indicted by the NY state Supreme Court for allegedly stealing source code from their former employer, flow trader, and a market-trading software provider [2]. In the same year, there was also a high-profile case involving Xilinx, a reputed and well-known FPGA manufacturer, and Flextronics International Ltd., a chip supplier. Xilinx alleged that Flextronics bought Xilinx's FPGA chips at a discounted rate (by lying about the intended end users), remarked the chips as higher grade and sold them for elevated prices, thereby violating Xilinx's IP through misrepresentation and exposing them to liabilities [3]. In 2015, Versara, a Texas-based software company, filed a lawsuit against automotive giant Ford [4]. The lawsuit alleged that Ford developed an in-house tool based on Versara's intellectual property, immediately after terminating its longtime contract with the company which provided Ford with its proprietary vehicle management software. The incidents highlighted above are just a modest sampling of the countless cases in which the electronic intellectual property of companies (and people) was violated, resulting in protracted litigation and massive loss of revenue/reputation.

B. Shakya (✉) · M.M. Tehranipoor · S. Bhunia · D. Forte
University of Florida, Gainesville, FL, USA
e-mail: bshakya@ufl.edu

D. Forte et al. (eds.), *Hardware Protection through Obfuscation*,
DOI 10.1007/978-3-319-49019-9_1

1.1.1 Obfuscation for Intellectual Property Protection

In a world of tough competition, companies often spend a great deal of time and resources in reverse engineering and understanding their competitor's products. This routinely happens in industries ranging from automotive and computer software to electronics. While reverse engineering in itself is not a crime (in fact, it is protected by law), the information derived from reverse engineering could be used in a number of malicious ways. Consider a company that exploits their competitor's intellectual property by incorporating the IP into their own products, without providing any credit or compensation to the IP's rightful owner. Now, think of this scenario in the context of today's global economy where IP protection laws (and the degree of their enforcement) vastly vary from one part of the globe to another. Due to such realities of today's global economy, IP protection can no longer be limited to passive methods such as patents, copyrights, and watermarks. An active approach to intellectual property protection is required, of which obfuscation is a vital part.

1.1.1.1 Definitions

Obfuscation is defined as the technique of obscuring or hiding the true meaning of a message or the functionality of a product, in order to protect the intellectual property (IP) inherent in the product. A more formal definition of obfuscation, in the context of 'circuits' (boolean operators computing a logic function) or programs (that implement an algorithm or a function), has been provided in the field of cryptology. Formally, an obfuscator O is defined as a 'compiler' that transforms a program P into its obfuscated version $O(P)$ that has the same functionality as P yet is unintelligible to an adversary trying to recover P from $O(P)$. The obfuscator O has two key requirements: (i) $O(P)$ computes the same function as P (i.e., it is functionality-preserving) and (ii) anything that can be efficiently computed (in polynomial time) from $O(P)$ can also be computed given 'oracle' access to P (i.e., $O(P)$ can be used as a 'virtual black box') [5].

1.1.1.2 Encryption Versus Obfuscation

Encryption is the most effective way to achieve security and privacy of data and communication, but it cannot provide a full-fledged solution for IP piracy. The overall differences between encryption algorithms and obfuscation are summarized in Fig. 1.1. As is clear from the figure, algorithms, such as Advanced Encryption Standard (AES) and Rivest–Shamir–Adleman (RSA), are cryptographic primitives that transform plaintext data into mathematically random ciphertext given a key (encryption phase). They also perform the reverse operation (ciphertext to plaintext) on application of the same or different key (decryption phase). In contrast to encryption, obfuscation does not necessarily rely on key-based access control. It allows

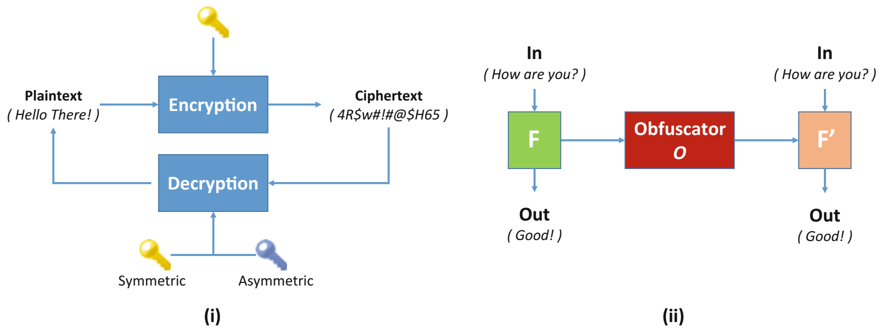


Fig. 1.1 **a** Encryption. **b** Obfuscation

an attacker to use the obfuscated program $O(P)$ as a virtual black box and is only concerned with the protection of the program P. Obfuscators are also commonly used in the context of obfuscating *programs* that implement an algorithm as opposed to general data (e.g., user information/authentication and passwords) that encryption can be used for.

Unfortunately, one major weakness that obfuscation has in relation to encryption is that the security of obfuscation techniques cannot be reduced to mathematically 'hard' problems such as integer factorization for RSA. In a landmark paper [5], it was shown that the general notion of virtual black-box obfuscation is not achievable for all programs. The authors argued that there exists a family of functions (represented as boolean circuits) that were inherently 'unobfuscatable.' In other words, given a program P' that computes the same input–output relationship as P, the attacker can feasibly reconstruct P or extract a secret s from P' about P with non-negligible probability. This means that unlike encryption, which does not prevent any set of data from being encrypted securely, obfuscation is not a universal operation.

1.1.1.3 Alternative Definitions

In spite of the above, later work has shown that although 'all functions' cannot be securely obfuscated (reduced to a 'hard' computational problem), some functions (such as a point function, which can be thought of as a password checking program) can be securely obfuscated [6]. Later, the authors in [7] showed that the virtual black-box property, which implies that the program should leak absolutely no other information than its input–output relationship, might be too strong to be achievable in practice. They proposed a relaxed definition termed as 'best possible obfuscation' which states that an obfuscated program $O(P)$ implementing a function F can leak as little information as any other program that computes the same function F. In other words, an adversary can learn no more information about the obfuscated program $O(P)$ than he or she can learn from any other program computing the same function.

While this does not guarantee what kind of information is securely hidden in $O(P)$, it assures that the obfuscation is literally the best possible [7].

In parallel, another definition of obfuscation, termed as 'indistinguishability obfuscation,' has also been proposed [5]. This definition implies that given two programs (or circuits) C_1 and C_2 which compute the same function F, an indistinguishability obfuscator O exists such that $O(C_1)$ and $O(C_2)$ are indistinguishable. This means that an attacker only has a random chance of being able to figure out whether he has $C1$ or $C2$ given possession of $O(C_1)$ or $O(C_2)$. Candidate constructions for such indistinguishability obfuscators have also been recently proposed using multilinear maps [8]. Note that the definition of indistinguishability obfuscation does not consider the strength of obfuscation of C_1 or C_2, i.e., it only says $O(C_1)$ and $O(C_2)$ are indistinguishable, not how strongly obfuscated they are on their own. It should also be noted that such obfuscators, although provably secure, are nowhere near practical at this point, as shown by a case study on a 16-bit point function (consisting of 15 *AND* gates) which blew up to a 31.1 GB file after running on a 32-core server for 9 h [9]. Nonetheless, the field of program obfuscation has received a great amount of attention in the past few years. More developments in optimization (as well as cryptanalysis) of these techniques can ensure practicality of provably secure obfuscation in the near future. It can also help us to gradually move away from ad hoc obfuscation practices such as code scrambling and white space removal for software obfuscation (see Sect. 1.3), which rely on the highly contested notion of 'security through obscurity.'

1.2 Hardware Obfuscation

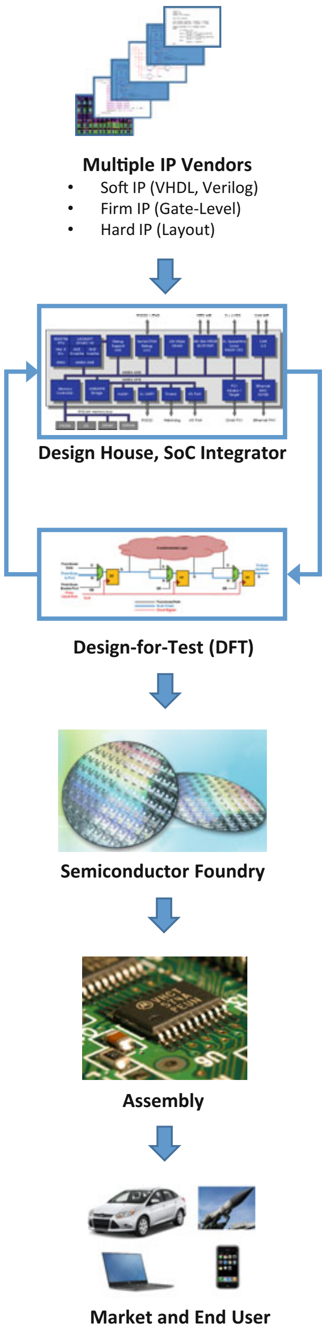
Hardware obfuscation, as it relates to circuits (combinational, sequential, or system-on-chip), is concerned with protecting semiconductor intellectual property (IP). In the context of hardware, IP refers to a reusable unit of logic, cell, or chip layout design that is either licensed to another party or owned and used solely by a single party. IP protection has become a hot topic for research (and practical implementation), especially in light of today's globalized semiconductor production flow where trust between various entities in the supply chain is hard, if not impossible, to maintain. While techniques for achieving hardware obfuscation may be completely different than its software counterpart (see Sect. 1.3), the basic motivation remains the same in both cases: protecting intellectual property from adversaries capable of reverse engineering, piracy, and malicious alteration. Before discussing how hardware obfuscation can be realized, it is vital to understand the semiconductor supply chain and all the threats that are involved in the production of today's integrated circuits (ICs).

1.2.1 Integrated Circuit Supply Chain

Figure 1.2 shows the various steps involved in today's semiconductor supply chain. Each step is carried out in different parts of the globe by different entities (not necessarily under the same company) in order to reduce the insurmountable costs associated with producing state-of-the-art ICs and to reduce time-to-market.

- *IP Vendor*: Integrated circuits today are most commonly in the form of system-on-chips or SoCs. This means that a single silicon die contains intellectual property from several different vendors who could be scattered all across the globe. For example, the power management circuitry may come from an analog IP vendor in Texas, while the cryptographic IP core might come from a separate vendor in Europe. With increasing complexity of today's ICs and short turnaround times, it only makes sense for a design house (or SoC integrator) to buy IPs from several different vendors (usually at a much better rate) than to build the entire IC in-house from scratch. These IPs can generally be classified as (i) soft IP (RTL level, e.g., in the form of Verilog or VHDL), (ii) firm IP (gate-level IPs), and (iii) hard IPs (layout-level IPs, also known as hard macros, e.g., embedded SRAM).
- *Design House/SoC Integrator*: After collecting the necessary IP blocks, the design house puts these IPs in a single design and performs exhaustive simulations/verification to ensure that the overall design functions as intended. During this stage, electronic design automation (EDA) tools, which are commonly purchased from external EDA vendors, are heavily employed in order to perform synthesis, place and route (P&R), timing analysis, and verification. Note that most design houses today have gone 'fabless' meaning that they do not maintain their own silicon production facility (i.e., foundry).
- *Design-for-Test*: Design-for-Test (DFT) is a stage in the IC design process where infrastructures are integrated on-chip for post-manufacturing tests. A few decades ago, it was feasible to comprehensively test bare circuits after manufacturing so that defects generated during fabrication (e.g., short-circuit, damaged gates) could be detected quickly. However, with the sheer scale of ICs today, it is no longer possible to engage in exhaustive logic testing post-fabrication. As a result, the design house can choose to send its entire design (usually in gate-level form) to a separate DFT facility that specializes in inserting test infrastructures into the circuit. This can include specialized flip-flops (FFs) called scan FFs, which can allow a tester to observe and control internal parts of the design (which, otherwise, might not be accessible through the primary inputs of the circuit). Other advanced test structures for self-diagnosis such as built-in-self-test or BIST and test compression can also be incorporated at this stage to ensure good fault coverage of the circuit-under-test.
- *Foundry*: After performing comprehensive tests and design, the SoC integrator generates a final layout file (usually in the form of a GDSII file) and sends it to a semiconductor foundry. The foundry first generates a mask from the file and then etches the patterns from the mask onto an actual silicon wafer to produce the IC die (after dicing the wafer). The foundry may also test the individual die at this point, using manufacturing test patterns that are provided by the design

Fig. 1.2 Semiconductor supply chain



house/DFT facility. It is important to note that the foundry is usually the most cost-intensive stage in the flow, as advanced nodes ($<$22 nm) require state-of-the-art tools for atomic layer deposition (ALD), extreme ultraviolet lithography (EUV), and a large-scale clean room capable of high-volume production. As a result, most foundry facilities are located offshore where labor and operation costs can be kept to a minimum.

- *Assembly*: After a foundry manufactures the IC, they are sent to a separate facility that specializes in packaging the die into a complete chip. The die is first mounted on a substrate after which bond wires or solder bumps (for flip-chip packaging) are used to connect the I/O, clock, and power ports on the die to actual pins on a plastic packaging.[1] After this, the packaged IC is again tested and ready for incorporation within a larger electronic system.
- *Market, End User*: Once the ICs are integrated into a system, it usually reaches a distributor who is then responsible for delivering the product to the final vendors, after which it reaches the market and ultimately the consumer.

3D Integrated Circuits

While the flow described above applies to most ICs in production today, newer technologies such as 3D or 2.5D integrated circuits add a few additional steps to the pre-existing flow. Both 3D and 2.5D ICs allow the integration of multiple dies on the same package (vertically for 3D and horizontally for 2.5D) with high-bandwidth interconnects between them. They offer numerous advantages in terms of reduced footprint for increased computational power, shorter average wire length for reduced parasitics and power dissipation, possibility of heterogenous integration (e.g., MEMs stacking and dies at different technology nodes in the same package), and reduced latency (e.g., when stacking processor and memory). In through-silicon-via (TSV) enabled 3D IC manufacturing, multiple dies (or wafers) with different partitions of the design are fabricated separately. After fabrication, the die (or wafers) is aligned on top of each other and die-to-die (or wafer-to-wafer) high-bandwidth interconnects (called TSVs) are created to connect the partitions. In 2.5D IC, two (or more) dies are placed on top of a single interposer layer and microbumps on the individual die connect to wires in the interposer layer to create a final integrated die. Note that the die-to-die or wafer-to-wafer integration can take place in the same foundry or a separate foundry. More detailed discussions on these emerging integration technologies can be found in their respective chapters in this book.

FPGA Manufacturing

While the IC design flow we presented is geared for ASICs (application-specific integrated circuits), one could also consider the design flow for FPGAs (field-programmable gate arrays), which are widely used today. While FPGA manufacturing follows the same flow as ASICs, the design flow for a FPGA-based product is completely different. In the FPGA product flow, a design house simply buys FPGA chips from a vendor (who specializes in manufacturing FPGAs). The design house

[1]Note that the foundry might also have packaging capabilities.

then combines soft IPs (from multiple vendors as well as the design house itself) and then integrates them into one final wrapper, which is then converted to a bitstream file (using vendor EDA tools). This file configures the lookup tables (basic building blocks of FPGAs) and routing resources of the FPGA to realize the design in silicon. Note that this FPGA flow is much shorter (and simpler) than that of ASICs. However, the high monetary cost associated with FPGAs (per unit), difficulty of high-volume production, lower performance (in terms of clock speed), and higher-power consumption limit the scope of FPGAs despite their flexibility. Nonetheless, the protection of IP contained within the FPGA bitstream is an important area of research.

PCB Manufacturing

Printed circuit board manufacturing, while not as complex as IC manufacturing, also tends to have a distributed supply chain, which is described below.

- *Design House*
  - A design house who manufacturers electronic/embedded systems first creates the layout file of his or her PCB with the help of an electronic design automation tool. During this stage, ICs that are part of the PCB are placed onto a template and wires are created between components by a combination of manual and autorouting.
  - After placement and routing, extensive simulations are performed to check the integrity of signals as they pass through the board (e.g., in terms of cross talk and signal degradation in the case of long wires).
  - The final layout of the PCB (in the form of a Gerber file) is then produced with contains information about (i) the number of layers in the board (modern PCBs have as many as 8–10 layers stacked) and (ii) exact coordinates of vias, wires, and components.
- *PCB Manufacturer*
  - The (Gerber) design is then electronically transmitted to a PCB manufacturer who uses it to produce the final board. The production can involve photolithography, milling, silk-screen printing, or a combination of the three to create the copper wire traces and vias on FR4 layers (the insulating base of a PCB). The PCB manufacturer also usually offers services for mounting + soldering the desired ICs on the board as well. After manufacturing, the PCB can go back to the design house, but it is usually forwarded to an assembly or distributor, who integrates the PCB into a complete electronic system.

1.2.2 Threats in the Supply Chain

It is clear that while the distributed supply chain for IC production has helped to drive cost and time-to-market down, it has also created a number of security and trust concerns among different entities. These threats, as they pertain to silicon IP security, are discussed below.

1.2.2.1 Reverse Engineering

The goal of reverse engineering is for the malicious party to recover the IP. After recovery, the malicious party can (i) use it to create a product that it can then sell to other parties; (ii) use the IP in its own product without compensating the rightful IP owner (breach of contract); (iii) find security vulnerabilities in the IP (e.g., weak random number generation) and exploit it later on; and (iv) insert a targeted backdoor in the IP after gaining a complete white box understanding of the IP. The threats could be present in different stages.

- *Design House*: The design house could potentially reverse engineer the IP core it receives from the IP vendor (firm and/or hard IP).
- *Foundry*: The design house provides the complete design in the form of a GDSII file to the foundry, along with manufacturing test patterns. The foundry could easily recover the netlist from the GDSII file by finding the connectivity information from the layout and coupling it with the standard cell library it had previously provided to the design house.
- *Market*: Once an adversary obtains a manufactured IC (either through the open market or through theft), he or she could try to reverse engineer the chip through destructive means to recover the complete design/IP [10]. Note that since we are concerned with IP protection here, side-channel/noninvasive/semi-invasive attacks that try to recover the secret key will not be considered. For destructive reverse engineering, the attack begins by decapsulating the chip from its packaging (either by mechanical abrasion or by corrosive acids), which exposes the bare die. Once the die is exposed, the attacker begins the slow process of imaging of all the layers of the IC (via scanning/transmission electron microscopy, high-resolution optical microscopy, focused ion beam, etc.). After imaging each layer, the attacker reaches for the next layer by using techniques such as chemical mechanical polishing (CMP) or plasma etching which helps to grind away a specific depth of the chip (and specific materials, depending on the etchant used). Once images for each layer are obtained, they are stitched together using automated image processing algorithms to obtain the full layout of the IC. If a standard cell library is available, the layout can then be converted to a gate-level netlist. While the process seems extensive, there are dedicated companies (e.g., TechInsights and Chipworks in Canada and Integrated Circuit Engineering Corp. in Arizona) that can perform the RE tasks at reasonable price/turnaround time and impressive accuracy. For a more detailed treatment of invasive reverse engineering, we refer the reader to [11].
  - *PCB Reverse Engineering*: Similar kinds of destructive delayering and imaging attacks can also be applied to PCBs in order to recover the design. Since PCB traces are not nanoscale (yet), noninvasive techniques such as X-ray computed tomography can be used to image each layer of the PCB and get the connectivity

information (which can be used to produce a Gerber file) in a matter of hours [12].
– *FPGA Reverse Engineering*: When it comes to an FPGA, the IP that an adversary would try to steal would be the FPGA bitstream, which is usually stored in an onboard/on-chip nonvolatile memory. If the bitstream is unencrypted, the attacker could read out the memory, by either probing or imaging. In case of an encrypted bitstream, side-channel attacks can first be used to recover the encryption key [13]. After recovering the bitstream, the attacker can use it illicitly on another FPGA device. He or she could also convert the bitstream to its corresponding netlist [14].

1.2.2.2 IP Piracy

Apart from reverse engineering, an adversary in the supply chain could also use the IP illicitly as is. With respect to the supply chain, the following threats are involved.

- *Foundry*: A foundry may only be contracted to produce a certain number of the design house's IP. However, since the foundry has the complete working design, they may produce excess copies of the design and sell them in the market (as is or relabeled/remarketed as a 'cloned' product). This effectively allows them to forgo any research and development (R&D) costs and make a profit by illicitly using someone else's IP. This practice is referred to as 'overproduction'.
- *Design House*: A design house may only be licensed to use a vendor's IP core on a limited number of chips (and paying royalties depending on how many chips were manufactured) or for a specified period of time. Unfortunately, if the IP vendor does not have a means to 'meter' the number of chips produced or actively track the IP's license, the design house could engage in IP piracy by 'overusing' the IP. This is in addition to the threat of the design house modifying the IP and selling it under a new name to other unauthorized parties.
- *Design-for-Test*: An offshore untrusted DFT facility may also pirate the IP (by producing a cloned version of the IP and selling it to unauthorized third parties), as it has the complete gate-level design in its possession.

1.2.2.3 Tampering

An adversary in the supply chain could also tamper with the design and introduce vulnerabilities or backdoors (i.e., hardware Trojans) into it. Two types of attacks might be possible in this scenario: targeted and untargeted. In a targeted attack, an adversary, such as an untrusted foundry, design house, DFT facility, or even an untrusted EDA tool, could gain a partial or complete understanding of the IP-under-attack and insert Trojans that bring about a specific malicious effects. For example, a foundry could decrease the entropy produced by a random number generator in a fabricated crypto-core [15]. It could also severely thin down the interconnect on a critical path on the design, such that it fails prematurely. An untrusted DFT facility could increase the observability of an internal node in the design, such that it reveals a critical internal asset directly through its primary outputs.

In an untargeted attack, an adversary's goal is sabotage or denial-or-service attack, by exploiting some critical portions of the design (e.g., power supply net and clock pin) and without gaining a complete understanding of the underlying design (through reverse engineering). For example, a foundry, if it finds unused space in the layout of the design, could implement a Trojan that, once triggered, resets a number of flip-flops or dramatically decreases the clock speed in the design. Similar kind of attacks could also be implemented by an untrusted DFT facility (without any space constraints, since the design is at gate level). It should also be noted that such targeted and untargeted Trojan attacks can also be implemented by an IP vendor (who perhaps has cloned another vendor's design introduced a Trojan into it and then sold it to a design house). For a comprehensive review of hardware Trojan attacks, we refer the readers to [16].

1.2.3 Why Isn't Encryption a Solution?

Due to the convoluted nature of the supply chain, numerous attacks which could compromise an entity's IP rights are possible. On quick thought, one might naively think that the solution to most, if not all of these issues, is to encrypt the IP. However, unlike software, encryption is not a viable option for ICs. While encryption implies a certain storage or speed overhead in software, hardware encryption implies actual gates. Since these gates are cast in silicon and encryption/decryption cores are not exactly area-efficient, encryption becomes unreasonable. Moreover, during most stages in the supply chain (such as foundry, DFT, and design house), the design is a complete white box to the adversary, i.e., information relating to all the gates and their interconnections is available. Since most encryption/decryption cores have repeating structures of arithmetic operations (e.g., AES-128 has 10 identical rounds of permutation/substitution), they are easily identifiable under a white-box attack model. This means that the adversary could simply go in and remove the core, thereby nullifying the security provided by the crypto-core. Lastly, encryption, as we discussed in Sect. 1.1.1.2, requires keys. This implies that a key management infrastructure must be available throughout each stage of the supply chain, and the design would also need to be periodically 'unlocked' in order to perform tests (especially in the case of encrypted soft IP cores—see Sect. 1.2.4.1). This creates further logistical issues to an already complex supply chain. Thus, alternatives are required in order to protect semiconductor intellectual property which could potentially cost billions to develop.

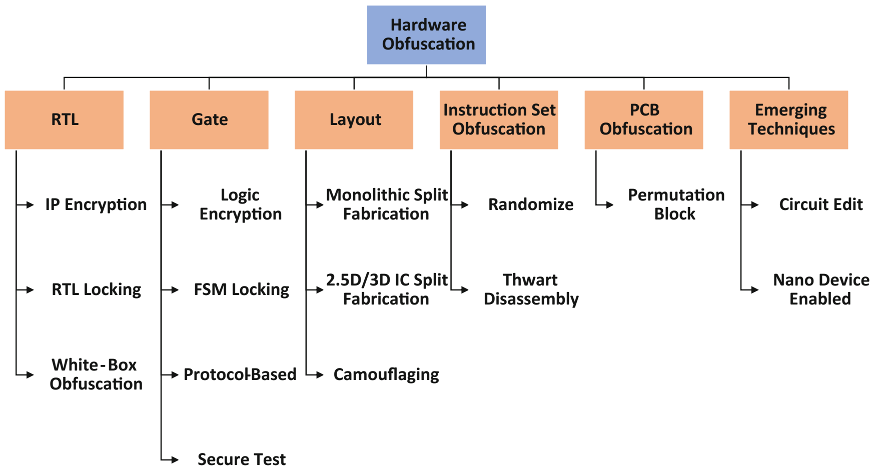


Fig. 1.3 A taxonomy of hardware obfuscation techniques

1.2.4 Techniques for Hardware Obfuscation

With the semiconductor supply chain and its inherent threats in mind, researchers have developed numerous hardware obfuscation techniques over the past decade. A taxonomy of these techniques is presented in Fig. 1.3 and is briefly introduced below.

1.2.4.1 RTL Level

Register-transfer level intellectual properties (IPs), also known as soft IPs, are commonly in the form of Verilog or VHDL code. Soft IP obfuscation can be achieved through the following methods.

- *IP Encryption*: The entire soft IP can be encrypted by common encryption techniques such as AES or RSA. In this setting, key management is usually handled by the EDA tools (which are assumed to be trusted[2]) and the IP buyer simply uses the encrypted IP as a black box. Unfortunately, the technique is limited to flexibility as the buyer/customer might be limited to a particular EDA tool. Recently introduced standards such as the IEEE P1735 encryption standard [17] have attempted to ease the interoperability of encrypted IPs across various EDA tools.
- *RTL-level Locking*: The authors in [18, 19] have proposed separate key-based locking approaches for RTL-based IPs. In these approaches, RTL code is first represented as a data flow [18] or state transition graph [19]. The graph is then

[2]A trusted party is committed to ensuring a proper IC design/fabrication flow (i.e., does not insert Trojans and protects IP confidentiality).

modified with key states, i.e., additional states in the FSM representation of the code that must be traversed with the help of a key sequence [18] or code word [19]. The IP comes into functional mode only when the correct keys are applied; otherwise, the IP is stuck in a non-functional, obfuscated mode. After obfuscating the graph and embedding locking features, the RTL code is regenerated from the graph, resulting in the final obfuscated IP.

- *White-box Obfuscation*: Soft IPs can also be obfuscated in terms of intelligibility and readability. The authors in [20] have utilized techniques such as loop transformations and reordering of statements in order to make a VHDL source code unintelligible yet functionally akin to the original code. The work in [21] explores control flow flattening, where a function or loop is broken into blocks and delegated to 'switch' statements, due to which the control flow of the program becomes much less obvious to an attack (as opposed to a simple 'for' loop where the execution order is obvious). Both of these techniques are more in line with traditional software obfuscation, which will be explored in Sect. 1.3. Unfortunately, such white-box obfuscation does not lock or obfuscate the functionality of the IP, leaving it vulnerable to IP piracy and overuse.

1.2.4.2 Gate Level

Gate level IPs, commonly referred to as firm IPs, are expressed in the form of netlists. In a netlist, the IP is expressed in the form of nets (connections) and a collection of standard logic cells. In order to protect gate-level IP, traditional encryption can be applied. However, this usually comes at the cost of significant hardware overhead and possible 'removal' attacks, as we previously discussed in Sect. 1.2.3. With these challenges in mind, several novel obfuscation techniques at the gate level have been recently proposed, of which the notable ones include the following.

- *Logic Encryption*: In this technique, extra gates such as XOR, XNOR, and MUX are inserted into the netlist of a design [22, 23]. These gates (and their logical output) are controlled by key bits which can be stored in a tamper-resistant nonvolatile

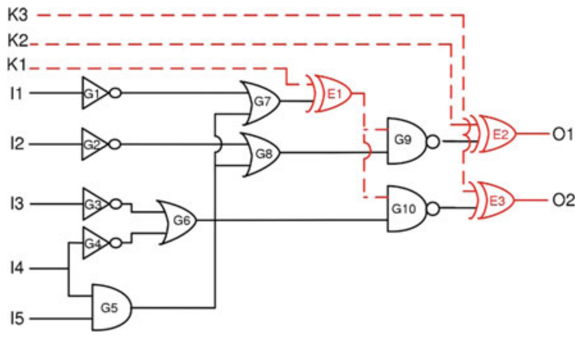


Fig. 1.4 A logic-locked circuit with three key gates [23]

memory or be derived from PUFs (see Fig. 1.4). The security of this approach lies in the fact that only the trusted design house knows and can apply all the correct key bits. Without the correct key bits, incorrect logical values are generated in the internal circuit nodes which eventually lead to faulty outputs. This effectively obfuscates the circuit to a third party who does not possess the correct key. Unfortunately, such locking techniques are vulnerable to Boolean satisfiability (SAT)-based attacks [24] as well as attacks that directly propagate the key bits to the circuit outputs [23] (details regarding both these attacks can be found in Sect. 1.2.5.2). Also, such techniques have only been studied on small-sized benchmark circuits, and their scalability is yet to be assessed.

- *FSM-based locking*: Several finite-state machine (FSM)-based locking techniques that are geared specifically toward sequential circuits have also been proposed. Among the most notable approaches, the authors in [25] have proposed the embedding of an authenticating FSM into a gate-level design. This authenticating FSM has to be traversed by an authorized user through a series of specific state transitions which are triggered by applying a series of input patterns only known to the user. If the chip is not unlocked via such a traversal, faulty values are generated by an additional modification kernel function and injected into the gate-level design in order to obfuscate the functionality of the locked chip. The security of the approach lies in the fact that the whole circuit is resynthesized after embedding the authentication and obfuscation features into the design, leaving an attacker with an insurmountable challenge of identifying (and removing) the implemented obfuscation. Although the authors propose utilizing pre-existing unreachable states in the FSM to incorporate the locking mechanism, the technique remains high in overhead (in terms of area, power, and delay).
- *Protocol-level*: The authors in [22, 26] have also utilized such locking techniques at a protocol level (with key exchange), in order to prevent an untrusted foundry from engaging in IC overproduction and IP piracy. These techniques are also commonly referred to as hardware metering. In [26], the authors utilize the aforementioned FSM-based locking technique, with the addition of a PUF to generate a unique start-up state for each IC. Upon manufacturing, a foundry relays the generated challenge–response pair from the PUFs so that the trusted design house can compute a unique unlocking FSM sequence for each chip. Additionally, the concept of black-hole states are introduced, which are irreversible state transitions from which the FSM cannot be reset. These black-hole states help in tamper detection in case a foundry attempts to randomly traverse the locked FSM. Although a cryptographically secure construction of the locked FSM is provided in [26] (via reduction to multi-point functions), such metering techniques are costly in overhead and do not take into account the testing procedure for ICs, a concern which has been more adequately addressed in [27].
- *Secure Test*: Recent work has shown that crypto-cores (AES, RSA, etc.) are particularly vulnerable to scan-based attacks, i.e., attacks that exploit scan flip-flops (FFs) embedded in a design as part of DFT, in order to reveal internal circuit values (including the secret key itself) [28]. These attacks are all possible because scan FFs are just normal registers in a design that can either be configured as scan or

normal FFs, depending on a 'scan enable bit' that is loaded from a multiplexer driving the *D* pin of the FF. Attackers who gain access to the scan chain can load multiple plaintexts into the cryptographic core (e.g., DES and AES) and switch from normal to scan mode. After doing this, they can flush out the corresponding intermediate values in internal registers and use hamming distance-based analysis to extract the secret key (or at least the individual round keys). Further, if the registers storing the round keys are part of the scan chain, the key could be directly read out without any analysis [29]. In order to prevent the attacker from accessing the scan chain, various obfuscation techniques have been employed. Simple solutions include the use of test compression structures that compress the value of several scan flip-flops into a single output, thereby making the observation of individual FF values unfeasible. Unfortunately, such compression-based obfuscation technique comes at the cost of high area overhead. Locking techniques have also been proposed that allow the designer to scramble the responses of the scan chain (chain of scan FFs) unless a secret key is applied. The lock-and-key technique proposed in [30] breaks up the scan chain into sub-chains. These sub-chains are configured properly and can be fed with the patterns sequentially only when the correct key is applied. If the wrong key is applied, the sub-chains are configured incorrectly by an LFSR, resulting in a 'lock' of the scan chains, which then results in wrong scan-out values. The work in [27] uses a similar concept to combat IC piracy. In the proposed technique, a 'scan locking block' (composed of a scrambling block and an XOR network) is utilized in order to perturb the scan chain responses. These perturbed responses can only be verified as correct by the design house, who can then appropriately chose whether to 'pass' or 'fail' a chip, thereby preventing the foundry from engaging in overproduction and allowing the design house to meter the number of chips produced. This approach also prevents out-of-spec/defective ICs from entering the market by giving the IP owner remote access to test responses. A more detailed treatment of scan-based attack and defenses can be found in the chapter on scan chain security in this book.

1.2.4.3 Layout Level

Layout-level IPs, also known as hard IPs, come in the form of geometrical and spatial information about the design which can be directly fabricated by a foundry. In order to protect the layout from piracy and possible Trojan insertion by an untrusted foundry, several split-manufacturing techniques have been proposed [31–33] (see Fig. 1.5).

- *Monolithic Integration*: In a traditional split-manufacturing flow, an untrusted foundry only fabricates the front-end-of-line (FEOL) layers, which include the expensive and state-of-the-art transistor/active layers. After FEOL fabrication, the design is sent back at the wafer level to the design house, who then uses a trusted foundry in order to complete the less costly back-end-of-line (BEOL) metal layers (see Fig. 1.6). While such techniques hide connectivity information from the foundry, attacks have been mounted on naive split manufacturing, which utilize

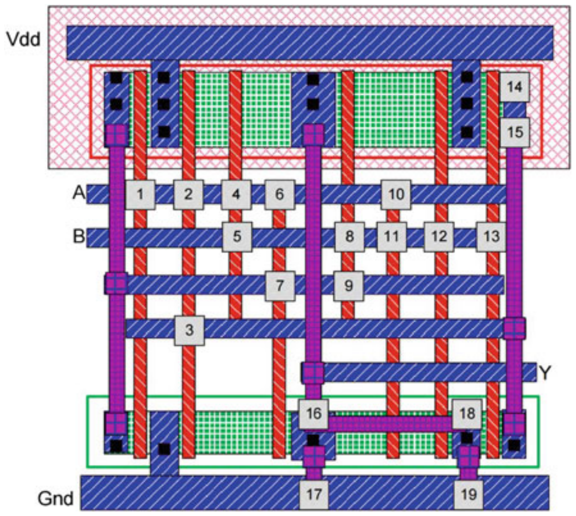


Fig. 1.5 A camouflaged standard cell that can function as a *NAND*, *NOR*, or *XOR*, depending on contact configuration [35]

proximity information from the assumptions that EDA tools use (e.g., gate distance to minimize wire length) [34]. Moreover, the biggest hurdle to split manufacturing is that the design house is still required to maintain a foundry to complete the BEOL, whose cost might be prohibitively expensive depending on the split layer. Further, foundry compatibility and wafer alignment with such monolithic split-fabrication techniques may also hurt IC yield.

- *2.5D/3D IC*: Attempts have also been made to utilize pre-existing 3D/2.5D IC technology to perform split manufacturing, as opposed to the monolithic integration technique proposed in [31]. In [32], wire-lifting is performed on a layout so that the lifted wires can be fabricated as separate layer at a trusted facility and the complete IC can be assembled by through-silicon-via (TSV) bond points in a normal 3D IC design flow. The authors also introduce the notion of *k*-security, by which every gate in the design in the FEOL layers is structurally akin to at least *k* other gates in the same design (as the BEOL information of the upper tier is missing).

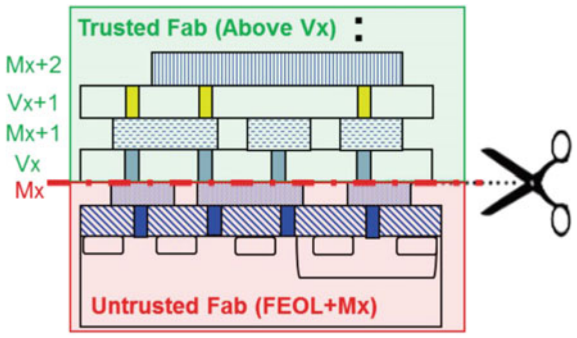


Fig. 1.6 Split manufacturing leveraging monolithic integration [37]

This makes it infeasible for the attacker (untrusted foundry) to identify the gates and thus launch a targeted hardware Trojan attack. In [33], 2.5D IC technology is leveraged in order to securely partition a gate-level design so that two or more partitions of the design can be fabricated at an untrusted foundry and the interposer layer connecting these partitions can be fabricated at a trusted facility. They introduce the concept of 'secure' partitioning, in which gates are iteratively moved from one partition to another until the global objective of a set wire-length penalty and 50% hamming distance (see Sect. 1.3.1) is met. Unfortunately, split manufacturing based on 3D/2.5D IC technology has the same drawbacks of requiring a separate fabrication facility. Further, these techniques require significant amounts of gate-swapping and wire-rerouting operations for obfuscation, leading to large area and delay overheads.

- *Camouflaging*: In order to protect against chip-level reverse engineering, the authors in [35] have proposed the use of special camouflaged standard cells. These cells have a layout that makes them appear the same to an invasive reverse engineer, whether they implement a *NAND*, *NOR*, or *XOR* functions. This is achievable through dummy contacts in the dielectric layer of the gate: Some contacts in the gate go all the way through the dielectric layer and into the metal layers above, while others are cut off. However, to an adversary, the contact looks the same from the top regardless of whether the contact is cut off or joined, giving rise to ambiguity while trying to identify the gate. This prevents the attacker from obtaining the complete netlist of the design. On the downside, the camouflaged gates themselves have a non-negligible power, area, and delay overhead, and a large number of these gates might have to be used in a design to achieve strong security for industrial designs. Further, recent attacks have shown that a design, even with an unrealistic number of camouflaged gates, can be effectively 'decamouflaged.' These attacks leverage SAT solvers and discriminating input patterns to resolve the hidden functionality of the camouflaged gates in negligible time [36].

1.2.4.4 Instruction Set Obfuscation

Every computer system has an underlying instruction set architecture (ISA) associated with it, which dictates the type of commands, data types, address space, and operation codes (opcodes) it can handle. The ISA serves as an intermediary between the software and hardware of the computer and is usually public knowledge. Unfortunately, this also means that for any attacker trying to remotely attack a system (or even invasively, via compromise of the memory unit holding the instructions), the ISA is well defined too, and thus, he or she can plant the attack on all computers using the same ISA. This serves as a basis for attacks such as buffer overflow. To combat the predictability of the ISA, the authors in [38] proposed a technique to scramble each byte of code (using pseudorandom numbers) and reversing the scrambling only when the code is executed in machine. This means that any unauthorized program, which was never scrambled, will be descrambled to random bits, thereby preventing any targeted malicious behavior. A similar approach which XOR's the instructions with a secret key as they are transmitted between the processor and the main memory (as implemented on a simulated x86 environment), was proposed in [39]. An attacker without key access can only inject malicious code which is incorrectly decoded, thereby raising a flag or causing a detectable error.

Another technique for instruction set obfuscation focuses on disrupting the disassembly phase of reverse engineering (i.e., conversion of machine code, in hexadecimal or binary form, to assembly code in human-readable form) [40]. This is achieved by carefully inserting 'junk bytes' in the instruction stream of the code. These junk bytes cause an automatic disassembler to either misinterpret the instructions or the control flow of the program but do not affect the program's functionality (i.e., semantics) as they are unreachable instructions during run-time.

1.2.4.5 PCB Obfuscation

In order to prevent a PCB design from piracy, the authors in [41] propose the inclusion of a permutation block on the board. The permutation block, implemented with a complex programmable logic device (CPLD) or an FPGA, takes in suitable pins (e.g., general-purpose I/O) from programmable component in the design (e.g., microcontrollers) and permutes its pin connections before they reach their destination. The permutation is resolved to the correct configuration only when the correct key is applied to the CPLD or the FPGA (the permutation block).

1.2.4.6 Emerging Techniques

There have also been a variety of novel obfuscation techniques that have been proposed recently. A few of the notable ones are highlighted below.

- *Chip Editor*: The basic idea of circuit edit (CE) is to utilize technologies such as focused ion beam (FIB) in order to edit an integrated circuit after fabrication on a one-by-one basis. CE has been widely used in the semiconductor industry in order to perform failure analysis and circuit/mask repair without undergoing the humongous costs of remaking the IC mask. Recent work has focused on leveraging circuit edit for logic obfuscation and trusted fabrication [42]. The proposed technique called Chip Editor involves the inclusion of extra gates or wires into the IC design which is then fabricated and tested by an untrusted foundry. After fabrication, the IC is returned to the design house (or other trusted party), who then uses circuit edit techniques such as FIB in order to remove the added gates or wires and revert the circuit to its intended functionality. The security of the approach lies in the fact that the foundry is unable to determine the gates or wires that are added in or modified by the design house, even when the added gates have been accommodated with design-for-circuit-edit features such as widened pads. Figure 1.7 shows one example of a gate insertion-based obfuscation enabled by circuit edit, where an *AND* gate is inserted and is edited to a buffer post-fabrication,

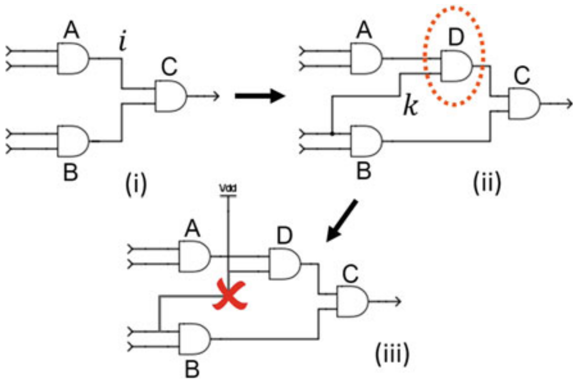


Fig. 1.7 Gate insertion enabled by circuit edit. **a** Original design with selected net *i*. **b** Gate D and wire *k* added. **c** Wire *k* edited to make gate D act as a buffer [42]

by tying one of its inputs to VDD. The authors in [42] also outline several other techniques for obfuscation, such as wire-swapping and insertion with other gate types. Compared to split manufacturing and 2.5D/3D IC obfuscation, the circuit edit technique removes the need for the design house to maintain a costly foundry to complete the BEOL or interposer layers and only requires a moderate cost FIB machine. The need for key management and secure key storage (e.g., in logic encryption) is also alleviated. However, the drawback of the approach is that it can only be utilized for low-volume IC production (since FIB operations take time and can only be done on a one-by-one basis). Similar to split manufacturing and 2.5D/3D IC obfuscation, it also does not prevent the IC from being reverse engineered once it enters the open market, thus limiting its scope to tightly controlled supply chains (e.g., military and aerospace).

- *Nanodevice enabled*: Several other techniques leveraging emerging nanoscale devices for circuit obfuscation have also been proposed. The authors in [43] propose the use of polarity-controllable silicon nanowire FETs (SiNW FETs) in order to make low-overhead IC camouflaging gates. As opposed to traditional camouflaged gates which require as much as 12 transistors, the authors show that it is possible to use only 4 SiNW to achieve a NAND, NOR, XNOR, and buffer functionality from the same gate, resulting in significant area and power savings. They also utilize SiNW FETs to make polymorphic gates, which can be configured to be either a NAND or a NOR gate depending on how the VDD and GND terminals are configured. Unfortunately, the drawback of using these novel devices is that they suffer from high leakage current and fabrication issues (imprecise device characteristics and limited scaling from top-down or traditional fabrication and yield issues from bottom-up or gate-first fabrication) [44].

1.2.5 Key-Based Classification

Hardware obfuscation techniques can also be classified on the basis of whether they employ a key-based locking mechanism or not.

1.2.5.1 Keyless

Keyless hardware obfuscation techniques ensure security by stripping away specific information regarding the design from the adversary. This includes techniques such as (i) white-box obfuscation for RTL (control flow graph obfuscation, removing comments, code compression, etc., to prevent reverse engineering); (ii) split fabrication (take away BEOL connectivity information from foundry so that they do not have the complete design); (iii) 2.5D/3D IC obfuscation (take away partition of a design, in the form of a separate die, from the foundry); (iv) instruction set obfuscation (make reverse engineering of code difficult); (v) IC camouflaging (make a gate incomprehensible for reverse engineers); and (vi) circuit edit (foundry is unable to find the added/modified gates or wires), all of which were discussed in detail in Sect. 1.2.4. While these techniques do not require key management, they usually have a more restrictive attack model (e.g., tightly controlled supply chain so that an attacker does not gain information about the design).

1.2.5.2 Key-Based

Key-based hardware obfuscation techniques include (i) IP encryption (via AES); (ii) RTL/FSM locking (correct sequence of state transitions required to unlock IP/IC); (iii) logic encryption (through embedding of locking key gates into design); and (iv) permutation block PCB locking. These techniques rely on the fact that the key used to unlock the design stays secret. Without the secret key, the design remains non-functional to an adversary. The idealistic assumption is that the adversary is reduced to using brute force in order to guess the correct secret key for the locked design. Assuming a large enough key length, such attacks become quickly unfeasible for most key-based obfuscation techniques (e.g., 2^{128} guesses for 128-bit key). However, several recent attacks have shown that an attacker can do much better than brute force in order to extract the secret key from an unlocked IC (and thereby use it to unlock other ICs) or check the correct key in far fewer tries than brute force. Some of these attacks are discussed below.

- *Boolean satisfiability (SAT) solvers*: Logic encryption techniques have recently been subjected to Boolean satisfiability (SAT) solver-based attacks [24]. In this attack, it is assumed that an adversary gains an unlocked version of the IC from the open market and uses this unlocked IC to query the correct input–output (IO) functionality of the design. Using this correct IO information and a separate locked IC implementing the same functionality, the attacker iteratively tries to rule out multiple incorrect key values. This is done by inputting the circuit in conjunctive normal form (CNF) to a SAT solver. Using the solver, the attacker identifies distinguishing input pairs (DIPs) that can help him or her to rule out multiple wrong keys in a single guess. This drastically reduces the key search space, enabling the attacker to deduce the right keys in a very reasonable amount of time.

- *Key propagation attacks*: The authors in [45] propose an automatic test pattern generation (ATPG)-based attack on logic encryption. In this approach, test patterns are generated, which can selectively ‘mute’ the effect of other gates in a circuit (e.g., by applying non-controlling values, such as logic 0 to an OR gate) and cause the key value on the key gates to propagate all the way to the primary outputs. Thus, through a topological evaluation of the circuit and generation of appropriate test patterns, the attacker can use to obtain the secret key and use it to unlock other ICs.

More details regarding both of these attacks can be found in the following chapter on logic encryption. The chapter also highlights appropriate countermeasures to these attacks, such as interference graph-based logic encryption (allowing key gate insertion at locations which do not allow propagation of the key bits to the output) and SARLock (utilizing the output of one-way random functions such as AES to set the key bits; this prevents an attacker from correlating the AES key to the circuit outputs, making SAT attacks provably unfeasible). However, note that both of these countermeasures are ad hoc in nature, thereby requiring large area overhead.

Besides these attacks based on intelligently stealing the key, key-based HW obfuscation techniques also suffer from an array of issues arising due to key management. Most of the obfuscation techniques rely on the key being stored in some form of non-volatile memory, which themselves are prone to imaging and probing attacks [46]. Further, key management (i.e., exchange of keys through a network allowing remote activation and authentication) is a non-trivial requirement, requiring area overhead for key exchange circuitry (e.g., RSA) and a secure network resistant to man-in-the-middle and replay attacks. An attempt at secure key exchange for remote authentication/activation of chips is presented in [47]. In the proposed FORTIS approach, RSA is utilized for end-point authentication between the chip at the foundry and the design house, and one-time pad (OTP) symmetric encryption is used for protecting the keys during exchange. A detailed treatment of the entire protocol can be found in the chapter on FORTIS in this book.

1.3 Software Obfuscation

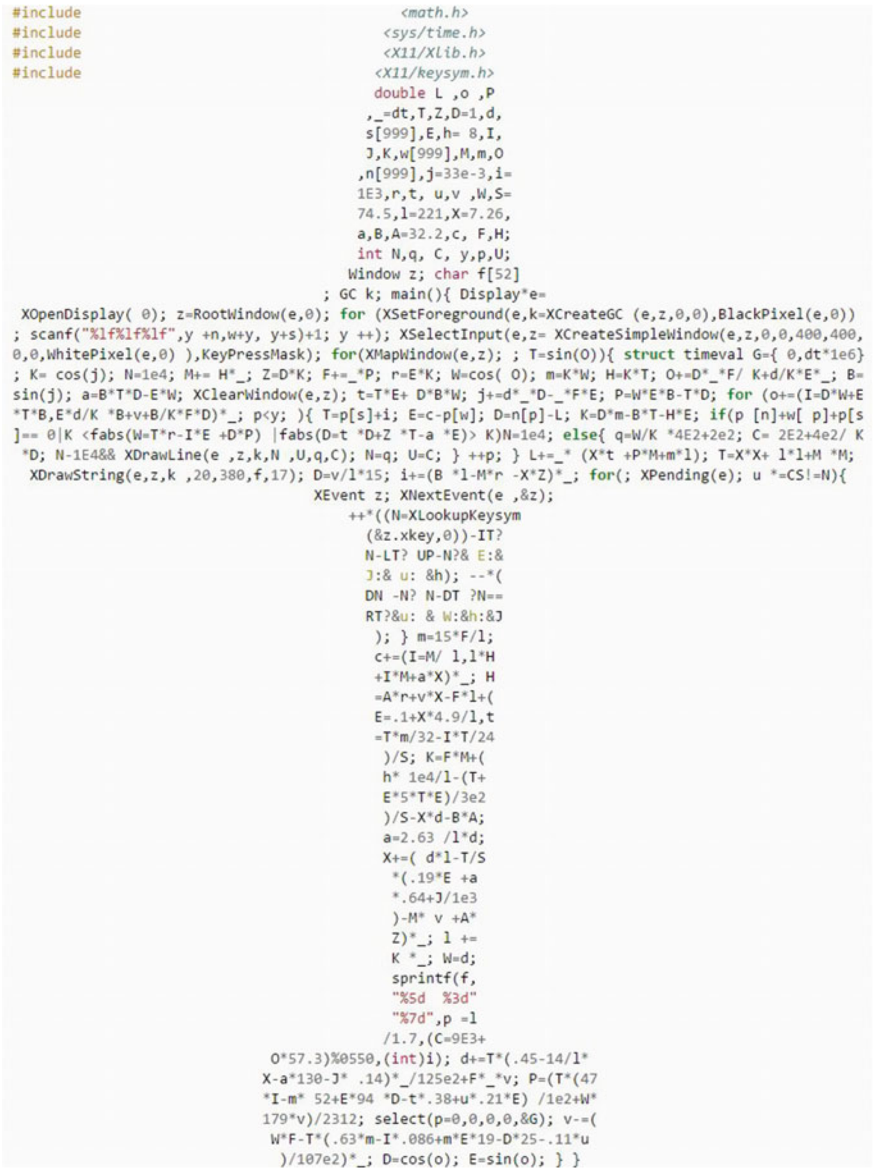
Computer software is the field that the term ‘obfuscation’ is most commonly associated with. While software obfuscation is beyond the scope of this book, we briefly introduce the field and highlight some reasons for utilizing it below.

Informally, software obfuscation relates to the practice of programmers concealing the implementation of their algorithm in code. This can include techniques ranging from simple comment or white space removal to more elaborate techniques such as loop unrolling (from the control flow graph or CFG representation of the program) [48]. While the overarching goal of obfuscation is IP protection (through prevention of reverse engineering), more specific reasons for considering obfuscation include the following.

- *Protection against malicious intent*: Computer vulnerabilities such as malware, virus, and Trojan horses often require the adversary to have a complete white box understanding of the software system they are targeting. For example, a buffer overflow attack requires the attacker to exactly know the instruction set architecture (ISA) of the victim's host system. Obfuscation techniques such as address space randomization [49] can be effective in preventing such attacks. Also, a software company, on discovering a bug in their software, could obfuscate their program with a patch to the bug and release it. This could be done so that an adversary is unable to recover the original bug from the patched program, so that he/she is prevented from exploiting customers of the software without the patch. Conversely, a malware could also employ obfuscation on itself in order to prevent detection by antivirus or intrusion detection packages [50].
- *Protection against IP theft and misuse*: One of the strongest reasons for obfuscation, in terms of IP protection, is to prevent IP theft and misuse. The software piracy cases that we discussed in Sect. 1.1 show the importance of strong obfuscation and why measures need to be taken to actively protect software IP from competitors and even potential customers.
- *Code Minification*: Apart from IP protection, obfuscation is also routinely used in the software industry in order to perform code compaction. Popular tools such as ProGuard [51] work with specific programming languages such as Java in order to simultaneously obfuscate and minimize code by removing unnecessary classes, shortening variable names, etc. This helps to produce executables that are compact in size and ease memory/storage constraints. Note that such 'minification' (and software obfuscation, in general) does not affect the functionality of the original code.
- *Recreational Obfuscation*: On a lighter note, competitions such as the international obfuscated C code contest (IOCCC) encourage participants to reproduce a code or algorithm in the most artistic or esoteric way possible [52] (Fig. 1.8).

1.3.1 Metrics for Hardware Obfuscation

Although we discussed the various techniques used for hardware obfuscation at different levels of abstraction, it is also important to note the metrics that go into (i) implementing and (ii) evaluating these techniques. *Implementation* metrics, which can be used for performing 'good' obfuscation, have two key requirements. First, they should be reasonably fast to compute (e.g., linear complexity) so that they can still be practical when applied to large circuits. Second, they should ideally be able to incorporate overhead (area, power, delay, etc.) and security constraints as the obfuscation technique is iteratively applied to the circuit. In this way, the designer does not have to wait till the entire obfuscation procedure is complete in order to evaluate the resulting security and overheads.



```
#include                                   <math.h>
#include                                 <sys/time.h>
#include                                 <X11/Xlib.h>
#include                                <X11/keysym.h>
                                        double L ,o ,P
                                       ,_=dt,T,Z,D=1,d,
                                        s[999],E,h= 8,I,
                                        J,K,w[999],M,m,O
                                       ,n[999],j=33e-3,i=
                                       1E3,r,t, u,v ,W,S=
                                       74.5,l=221,X=7.26,
                                       a,B,A=32.2,c, F,H;
                                       int N,q, C, y,p,U;
                                      Window z; char f[52]
                                   ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                  XEvent z; XNextEvent(e ,&z);
                                      ++*((N=XLookupKeysym
                                        (&z.xkey,0))-IT?
                                        N-LT? UP-N?& E:&
                                        J:& u: &h); --*(
                                        DN -N? N-DT ?N==
                                        RT?&u: & W:&h:&J
                                         ); } m=15*F/l;
                                         c+=(I=M/ l,l*H
                                         +I*M+a*X)*_; H
                                         =A*r+v*X-F*l+(
                                         E=.1+X*4.9/l,t
                                         =T*m/32-I*T/24
                                          )/S; K=F*M+(
                                          h* 1e4/l-(T+
                                          E*5*T*E)/3e2
                                          )/S-X*d-B*A;
                                          a=2.63 /l*d;
                                          X+=( d*l-T/S
                                           *(.19*E +a
                                           *.64+J/1e3
                                           )-M* v +A*
                                           Z)*_; l +=
                                           K *_; W=d;
                                           sprintf(f,
                                           "%5d  %3d"
                                           "%7d",p =l
                                          /1.7,(C=9E3+
                               O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                              X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                              *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                              179*v)/2312; select(p=0,0,0,0,&G); v-=(
                               W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                                )/107e2)*_; D=cos(o); E=sin(o); } }
```

Fig. 1.8 IOCCC Flight Simulator: Winning entry of the 1998 International Obfuscated C Code Contest [53]

Evaluation metrics can be grouped into two categories: preobfuscation and post-obfuscation. *Preobfuscation* evaluation metrics can be used to judge the difficulty in obfuscating a circuit before the actual technique is applied to it. These kinds of metrics help the designer to gauge the effort required (e.g., overheads that might need to be committed and computation time) in order to obfuscate the circuit before performing the actual obfuscation. They could also help in deciding from an array of candidate obfuscation techniques. *Post-obfuscation* evaluation metrics are applied to a circuit after the obfuscation is complete. In order to use these metrics, a golden circuit (i.e., the unobfuscated circuit) is required which is then compared to the obfuscated version in order to evaluate the resultant security and overheads.

Some of the metrics frequently which recur across different obfuscation techniques and can be used for implementation and/or evaluation are briefly discussed below.

- Implementation Metrics
  - *Fault Impact*: The authors in [23] have used fault impact in order to judge the appropriate locations to insert key gates into a circuit. The fault impact for a gate can be expressed as the product of the total number of test patterns that detect a stuck-at-0 fault at the gate's output and the number of outputs that get affected by the stuck-at-0 fault at the gate's output (the total fault impact is the sum of stuck-at-0 fault impact and stuck-at-1 fault impact). This metric roughly tells how likely it is that a fault (which, in this case, is the bit-flip induced by applying a wrong key at the key gate) can propagate to the outputs and cause an output error. The shortcoming of this approach is that fault detection is a computationally hard problem, and generating the patterns for detecting the faults would require exponential run-time as the circuit size scales. The authors in [33] have utilized a similar metric, which they term as mean observe and control values (MOV/MCV). As opposed to a stuck-at fault detection approach which was used by fault impact, MOV/MCV tracks the number of bit-flips in a wire and the corresponding number of bit-flips of the outputs in output cone (for MOV) or inputs in the input cone (for MCV) of the same wire. MOV/MCV has the same linear computational complexity as logic simulation.
  - *k-security*: The authors in [32] introduce the notion of k-security for indistinguishability of a gate in a circuit fabricated by split manufacturing. Given that an attacker has the complete netlist of the design and a partial netlist recovered from the FEOL layer, the authors claim that a design is k-secure if for every gate in the FEOL design, there exists at least k subgraph isomorphisms, i.e., k distinct gates in the complete netlist that the gate can be mapped to. For example, if a gate in the FEOL netlist has $k = 2$, it implies that the attacker has to randomly guess between two gates in the original netlist in order to identify the gate in the FEOL netlist. Unfortunately, the authors showed that determining whether a circuit is $k - secure$ is NP-complete. Therefore, they employed a SAT solver to gradually lift wires from the FEOL to the BEOL and heuristically check graph isomorphism after each lifting operation to construct a k-secure circuit. This also unfortunately leads to impractical area overheads.

- Evaluation Metrics
  - *Hamming Distance*: Hamming distance (HD) is a metric used to evaluate the difference between two given bitstreams b_1 and b_2. It performs a bit-by-bit comparison of two bitstreams and uses a percentage figure to describe how many bits are different between b_1 and b_2. For circuits, the output bits of a combinational circuit (or a sequential circuit at the same time instance) can be thought of as a bitstream. An average HD of 50% between b_1, the output of a circuit and b_2, the output of its obfuscated or locked counterpart, tells us that the responses between the two circuits can be the same only with a probability that is as good as random chance. In other words, the obfuscated circuit's response is completely different than that of the original circuit. This metric has been widely used for evaluating gate-level obfuscation techniques such as logic locking [23] and also camouflaging [35]. It is either calculated once post-obfuscation or recursively calculated after each change (e.g., after one key or camouflaged gate insertion). The drawback of this metric is that it is based on outputs and requires logic simulation which does not scale well with the number of inputs and size of the circuit. Most techniques employing HD estimate the figure by a random sampling of the input space (e.g., 1000 randomly selected input vectors).
  - *Verification Failure*: The authors in [25, 42] have used percentage verification failure as a metric for evaluating the performance of their obfuscation techniques. To calculate this metric, formal verification tools such as Synopsys Formality are used to perform logical equivalence checking between the obfuscated design and its original counterpart. The equivalence checking involves the use of proprietary static analysis techniques on the logic cones of the two designs to compare their output ports and flip-flop outputs (pseudo-output ports). The final verification failure figure is expressed as a percentage of failing comparison points (ports that failed equivalence checking) to the total number of comparison points (total no. of ports). The advantage of this metric over simulation-based Hamming distance is that it is much faster and scalable and does not suffer from the inaccuracies or coverage issues that arise due to simulation with a limited set of vectors/patterns. However, for purely combinational circuits, the metric might not be applicable due to the increased size of the logic cones (extending from the primary inputs all the way to the primary outputs, without any flip-flops in between). Note that verification failure can only be used as a post-obfuscation evaluation metric, as it requires the complete obfuscated circuit and the original circuit.
  - *Entropy*: Entropy refers to the amount of information contained in a system. In terms of obfuscation, entropy is used to determine the extent of information that can be non-trivially attained by an adversary by observing the obfuscated version of the circuit. The authors in [54] have used entropy as a measure of how easy it is for the attacker to gain information about the functionality of the circuit from the distribution of gate types. For example, a circuit synthesized with only two types of gates will have a very high entropy compared to a circuit synthesized with 30 different types of gates, from which the attacker might deduce clues

(e.g., a collection of XOR gates might hint to the addRoundKey stage of AES). Along the same lines, the authors also proposed a complimentary metric they term as 'standard cell composition bias.' This metric analyzes the proportion of standard cells (such as XOR, flip-flops, or arithmetic gates) in the design. A design with high bias (e.g., with a lot of XORs and few FFs/arithmetic gates) might indicate a cryptographic core, while a design with significantly more FFs might indicate a state machine logic. The goal is to synthesize the design with low bias, i.e., with equal proportion of different types of standard cells so that the attacker cannot make a generalized guess about the high-level functionality or purpose of the circuit. Both entropy and composition bias can be used as preobfuscation and post-obfuscation evaluation metrics.
– *Neighbor connectedness*: The authors in [54] introduce neighbor connectedness, which gives us an idea of how connected a cell in a design (in layout form, with respect to split manufacturing) is. If a cell is connected to a lot of other cells in its neighborhood (e.g., a 4×4 grid within a small radius R), its functionality/purpose could be deduced from the connectivity information. On the flip side, if connected cells are more 'spread out' (i.e., R is increased), an attacker without BEOL information could wrongly assume that a cell is connected to another functionally unrelated cell. Therefore, a design with low neighbor connectedness (i.e., where connected cells are far apart) would increase the reverse engineering required by the adversary (as he or she would keep making wrong connections based on distance (mis)information). Note that low neighbor connectedness also unfortunately implies an increase in wire length. This metric can be used for both preobfuscation and post-obfuscation evaluation.

Thus, there are an array of metrics that have been proposed in order to implement as well as evaluate hardware obfuscation techniques. The key issue with most of the proposed metrics is that they are impractical, either in terms of computing the metric itself or implementing the design guided by the metric. For example, evaluation metrics such as HD require logic simulation, which are unscalable on large designs, have significant errors when estimated with a small set of random vectors, and cannot be calculated as is by partitioning the design (as we discussed above). On the other hand, metrics such as k-security end up being too strong and an unfeasible notion of security, as is evident by the unacceptable area overhead that arises in trying to meet the metric. Thus, there exists a delicate trade-off between the computational complexity involved in metric computation and the overhead that results from adopting a particular metric to guide obfuscation.

1.3.1.1 Software Obfuscation Metrics

Although an in-depth treatment of software obfuscation is beyond the scope of this chapter/book, a few relevant metrics for software obfuscation, which indicate the level of complexity in reverse engineering or comprehending a program, are highlighted below.

- *Cyclomatic complexity* relates to the control flow graph (CFG) representation of a program. A program without any control statements (e.g., *IF*) would be assigned a complexity of 1, whereas a program with one *IF* statement and one evaluation condition would be assigned a complexity of 2 (one part evaluating to *TRUE* and another evaluating to *FALSE*, for a total of two linearly independent paths in the CFG). The complexity increases as more control flow statements are introduced in the program, indicating an increase in the test cases required to comprehend the program functionality.
- *Halstead complexity metric* defines a suite of measures such as program length, difficulty, and effort, which are all based on the number of distinct operators and operands that are utilized in the program [55]. Since this metric solely relies on the operators + operands, it is language-independent and has no notion of control flow. Nonetheless, the metric directly correlates to program execution time and amount of time a reverse engineer has to spend to evaluate the program.

1.3.2 Hardware Obfuscation Benchmarks

In order to show the efficacy of their obfuscation techniques, researchers frequently utilize 'benchmark circuits' on which they apply the technique and present results on the incurred area/delay/power overhead and security metric utilized. For gate-level techniques, popular examples of used benchmarks include the ISCAS '85 [56], '89 [57], and ITC '99 [58] benchmark sets (which are widely available in synthesized netlist form). Researchers have also utilized the placed-and-routed layout of these benchmarks to explore split-manufacturing obfuscation. These benchmarks were initially created as example designs (to be used as functional black boxes) for researchers to explore test pattern generation, scan chain insertion, fault coverage, and other VLSI test-related topics. As a result, their use in HW obfuscation has been more or less ad hoc. Moreover, most of these benchmarks date back decades and are only a few thousands in gate count. Due to these reasons, it becomes hard to argue about the scalability of HW obfuscation techniques implemented on these benchmarks to the million gate designs that are commonplace today.

1.4 Conclusion

In this chapter, we presented a general overview of hardware as well as software obfuscation. Software obfuscation focuses on developing general-purpose obfuscating compilers that can work for all programs. However, hardware obfuscation can vary in technique and scope, depending on the threat model and level of abstraction. Due to this, there cannot be a one-size-fits-all solution for all hardware obfuscation problems.

We also explored the various threats involved in each stage of the integrated circuit supply chain and presented a brief review of obfuscation techniques that have been developed to counter these threats. Lastly, we reviewed relevant metrics for HW obfuscation that the reader will encounter throughout the rest of this book.

References

1. Weber J (2016) Epic systems wins $940 mln US jury verdict in Tata trade secret case, reuters. http://www.reuters.com/article/us-tata-epic-verdict-idUSKCN0XD135. Accessed April 2016
2. Kirk J (2013) Three indicted in alleged source code theft from trading house, PC world. http://www.pcworld.com/article/2053020/three-indicted-in-alleged-source-code-theft-from-trading-house.html. Accessed Oct 2013
3. Rosenblatt J (2013) Xilinx sues Flextronics alleging fradulent chip resale, bloomberg technology. http://www.bloomberg.com/news/articles/2013-12-11/xilinx-sues-flextronics-alleging-fraudulent-chip-resale. Accessed Dec 2013
4. Bunkley N (2015) Ford accused by software maker of intellectual property theft, automotive news. http://www.autonews.com/article/20150604/OEM06/150609919/ford-accused-by-software-maker-of-intellectual-property-theft. Accessed June 2015
5. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S, Yang K (2001) On the (im) possibility of obfuscating programs. Annual international cryptology conference. Springer, Heidelberg, pp 1–18
6. Canetti R, Dakdouk RR (2008) Obfuscating point functions with multibit output. Annual international conference on the theory and applications of cryptographic techniques. Springer, Heidelberg, pp 489–508
7. Goldwasser S, Rothblum GN (2007) On best-possible obfuscation. Theory of cryptography conference. Springer, Heidelberg, pp 194–213
8. Garg S, Gentry C, Halevi S, Raykova M, Sahai A, Waters B (2013) Candidate indistinguishability obfuscation and functional encryption for all circuits. In: IEEE 54th annual symposium on foundations of computer science (FOCS). IEEE, pp 40–49
9. Apon D, Huang Y, Katz J, Malozemoff AJ (2014) Implementing cryptographic program obfuscation. IACR Cryptol ePrint Arch 2014:779
10. Torrance R, James D (2009) The state-of-the-art in ic reverse engineering. In: Cryptographic Hardware and Embedded Systems-CHES. Springer, pp 363–381
11. Quadir SE, Chen J, Forte D, Asadizanjani N, Shahbazmohamadi S, Wang L, Chandy J, Tehranipoor M (2016) A survey on chip to system reverse engineering. ACM J Emerg Technol Comput Syst (JETC) 13(1):6
12. Asadizanjani N, Shahbazmohamadi S, Tehranipoor M, Forte D (2015) Non-destructive PCB reverse engineering using x-ray micro computed tomography. In: 41st International symposium for testing and failure analysis, ASM, 1–5 November 2015
13. Moradi A, Barenghi A, Kasper T, Paar C (2011) On the vulnerability of fpga bitstream encryption against power analysis attacks: extracting keys from xilinx virtex-ii fpgas. In: Proceedings of the 18th ACM conference on Computer and communications security, pp. 111–124. ACM, 2011
14. Note JB, Rannaud E (2008) From the bitstream to the netlist. In: Proceedings of the 16th international ACM/SIGDA symposium on field programmable gate arrays, series FPGA 2008, New York, USA. ACM, pp 264–264. http://doi.acm.org/10.1145/1344671.1344729
15. Becker GT, Regazzoni F, Paar C, Burleson WP (2013) Stealthy dopant-level hardware trojans. In: International workshop on cryptographic hardware and embedded systems. Springer, pp 197–214
16. Tehranipoor M, Koushanfar F (2010) A survey of hardware trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25

17. IEEE computer society, IEEE recommended practice for encryption and management of electronic design intellectual property. https://standards.ieee.org/findstds/standard/1735-2014.html. Accessed December 2014
18. Chakraborty RS, Bhunia S (2010) RTL hardware IP protection using key-based control and data flow obfuscation. In: 2010 23rd international conference on VLSI design. IEEE, pp 405–410
19. Desai AR, Hsiao MS, Wang C, Nazhandali L, Hall S (2013) Interlocking obfuscation for anti-tamper hardware. In: Proceedings of the eighth annual cyber security and information intelligence research workshop. ACM, p 8
20. Brzozowski, M, Yarmolik VN (2007) Obfuscation as intellectual rights protection in VHDL language. In: 6th International conference on computer information systems and industrial management applications, CISIM 2007. IEEE, pp 337–340
21. Kainth M, Krishnan L, Narayana C, Virupaksha SG, Tessier R (2015) Hardware-assisted code obfuscation for FPGA soft microprocessors. In: Proceedings of the 2015 design, automation and test in Europe conference and exhibition. EDA Consortium, pp 127–132
22. Roy JA, Koushanfar F, Markov IL (2008) Epic: ending piracy of integrated circuits. In: Proceedings of the conference on design, automation and test in Europe. ACM, pp 1069–1074
23. Rajendran J, Pino Y, Sinanoglu O, Karri R (2012) Logic encryption: a fault analysis perspective. In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, pp 953–958
24. Subramanyan P, Ray S, Malik S (2015) Evaluating the security of logic encryption algorithms. In: IEEE international symposium on hardware oriented security and trust (HOST) (2015). IEEE, pp 137–143
25. Chakraborty RS, Bhunia S (2009) Harpoon: an obfuscation-based soc design methodology for hardware protection. IEEE Trans Comput Aided Des Integr Circuits Syst 28(10):1493–1502
26. Koushanfar F (2012) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans Inf Forensics Secur 7(1):51–63
27. Contreras GK, Rahman MT, Tehranipoor M (2013) Secure split-test for preventing IC piracy by untrusted foundry and assembly. In: IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFTS). IEEE, pp 196–203
28. Yang B, Wu K, Karri R (2004) Scan based side channel attack on dedicated hardware implementations of data encryption standard. In: Proceedings of the ITC international test conference on 2004. IEEE, pp 339–344
29. Nahiyan A, Xiao K, Yang K, Jin Y, Forte D, Tehranipoor M (2016) AVFSM: a framework for identifying and mitigating vulnerabilities in FSMS. In: Proceedings of the 53rd annual design automation conference. ACM, p 89
30. Lee J, Tehranipoor M, Patel C, Plusquellic J (2007) Securing designs against scan-based side-channel attacks. IEEE Trans Dependable Secure Comput 4(4):325–336
31. Vaidyanathan K, Liu R, Sumbul E, Zhu Q, Franchetti F, Pileggi L (2014) Efficient and secure intellectual property (IP) design with split fabrication. In: IEEE international symposium on hardware-oriented security and trust (HOST) 2014. IEEE, pp 13–18
32. Imeson F, Emtenan A, Garg S, Tripunitara M (2013) Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In: Presented as part of the 22nd USENIX security symposium (USENIX security 2013), pp 495–510
33. Xie Y, Bao C, Srivastava A (2015) Security-aware design flow for 2.5D IC technology. In: Proceedings of the 5th international workshop on trustworthy embedded devices. ACM, pp 31–38
34. Rajendran JJ, Sinanoglu O, Karri R (2013) Is split manufacturing secure? In: Proceedings of the conference on design, automation and test in Europe. EDA Consortium, pp 1259–1264
35. Rajendran J, Sam M, Sinanoglu O, Karri R (2013) Security analysis of integrated circuit camouflaging. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security. ACM, pp 709–720
36. El Massad M, Garg S, Tripunitara MV (2015) Integrated circuit (IC) decamouflaging: reverse engineering camouflaged ICS within minutes. In: NDSS

37. Vaidyanathan K, Das BP, Sumbul E, Liu R, Pileggi L (2014) Building trusted ICS using split fabrication. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 1–6, May 2014
38. Barrantes EG, Ackley DH, Palmer TS, Stefanovic D, Zovi DD (2003) Randomized instruction set emulation to disrupt binary code injection attacks. In: Proceedings of the 10th ACM conference on Computer and communications security. ACM, pp 281–289
39. Kc GS, Keromytis AD, Prevelakis V (2003) Countering code-injection attacks with instruction-set randomization. In: Proceedings of the 10th ACM conference on computer and communications security. ACM, pp 272–280
40. Linn C, Debray S (2003) Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the 10th ACM conference on computer and communications security. ACM, pp 290–299
41. Guo Z, Tehranipoor M, Forte D, Di J (2015) Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In: Proceedings of the 52nd annual design automation conference, series DAC 2015, New York, NY, USA. ACM, pp 114:1–114:6. http://doi.acm.org/10.1145/2744769.2744862
42. Shakya B, Asadizanjani N, Forte D, Tehranipoor M (2016) Chip editor: leveraging circuit edit for logic obfuscation and trusted fabrication. In: IEEE/ACM international conference on computer-aided design (ICCAD)
43. Bi Y, Shamsi K, Yuan J-S, Gaillardon P-E, Micheli GD, Yin X, Hu XS, Niemier M, Jin Y (2016) Emerging technology-based design of primitives for hardware security. ACM J Emerg Technol Comput Syst (JETC) 13(1):3
44. Cui Y, Zhong Z, Wang D, Wang WU, Lieber CM (2003) High performance silicon nanowire field effect transistors. Nano Lett 3(2):149–152
45. Rajendran J, Zhang H, Zhang C, Rose GS, Pino Y, Sinanoglu O, Karri R (2015) Fault analysis-based logic encryption. IEEE Trans Comput 64(2):410–424
46. Skorobogatov SP (2005) Semi-invasive attacks: a new approach to hardware security analysis, Ph.D. dissertation, Citeseer
47. Guin U, Shi Q, Forte D, Tehranipoor MM (2016) Fortis: a comprehensive solution for establishing forward trust for protecting IPS and ICS. ACM Trans. Des. Autom. Electron. Syst., 21(4):63:1–63:20. http://doi.acm.org/10.1145/2893183
48. Collberg C, Thomborson C, Low D (1997) A taxonomy of obfuscating transformations. The University of Auckland, New Zealand, Technical report, Department of Computer Science
49. Bhatkar S, DuVarney DC, Sekar R (2003) Address obfuscation: an efficient approach to combat a broad range of memory error exploits. Usenix Secur 3:105–120
50. You I, Yim K (2010) Malware obfuscation techniques: a brief survey. In: 2010 international conference on broadband, wireless computing, communication and applications (BWCCA), pp 297–300
51. Lafortune E et al. (2004) Proguard. http://proguard.sourceforge.net
52. Noll LC, Cooper S, Seebach P, Leonid AB (2005) The international obfuscated C code contest
53. IOCCC, IOCCC flight simulator. In: International obfuscated C code contest (1998). http://www.ioccc.org/1998/banks.c
54. Jagasivamani M, Gadfort P, Sika M, Bajura M, Fritze M (2014) Split-fabrication obfuscation: metrics and techniques. In: 2014 IEEE international symposium on hardware-oriented security and trust (HOST), pp 7–12
55. Halstead MH Elements of software science, vol 7
56. Hansen MC, Yalcin H, Hayes JP (1999) Unveiling the iscas-85 benchmarks: a case study in reverse engineering. IEEE Desi Test 16(3):72–80
57. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In: IEEE international symposium on circuits and systems. IEEE, pp 1929–1934
58. Corno F, Reorda MS, Squillero G (2000) RT-level ITC 1999 benchmarks and first ATPG results. Ieee Des Test Comput 17(3):44–53