Testing the Trustworthiness of IC Testing: An Oracle-less Attack on IC Camouflaging

Muhammad Yasin†, Ozgur Sinanoglu‡ and Jeyavijayan (JV)ξ Rajendran
yasin@nyu.edu†, ozgursin@nyu.edu‡, jv.ee@utdallas.eduξ,
† Electrical and Computer Engineering, NYU Tandon School of Engineering, NY, USA
‡ Electrical and Computer Engineering, New York University Abu Dhabi, Abu Dhabi, U.A.E.
ξ Erik Jonsson School of Engineering & Computer Science, The University of Texas at Dallas, TX, USA

***Abstract*—Test of integrated circuits (ICs) is essential to ensure their quality; the test is meant to prevent defective and out-of-spec ICs from entering into the supply chain. The test is conducted by comparing the observed IC output with the expected test responses for a set of test patterns; the test patterns are generated using automatic test pattern generation algorithms. Existing test-pattern generation algorithms aim to achieve higher fault coverage at lower test costs. In an attempt to reduce the size of test data, these algorithms reveal the maximum information about the internal circuit structure. This is realized through sensitizing the internal nets to the outputs as much as possible, unintentionally leaking the secrets embedded in the circuit as well.**

In this paper, we present HackTest, an attack that extracts secret information generated in the test data, even if the test data does not explicitly contain the secret. HackTest can break the existing intellectual property (IP) protection techniques, such as camouflaging, within two minutes for our benchmarks using only the camouflaged layout and the test data. HackTest applies to all existing camouflaged gate-selection techniques and is successful even in the presence of state-of-the-art test infrastructure, i.e. test data compression circuits. Our attack necessitates that the IC test data generation algorithms be reinforced with security. We also discuss potential countermeasures to prevent HackTest.

I. Introduction

Fabrication of integrated circuits (ICs) is not an entirely controlled process; a percentage of the manufactured ICs may not function as per the design specifications. Distribution of low-quality ICs could not only result in unreliable consumer products that jeopardize the reputation of a company, but also lead to catastrophic failures if the ICs are used in safety-critical applications. Thorough testing of ICs is essential to ensure the reliability of electronic products. Each manufactured IC, therefore, passes through a test that identifies whether the chip is defective or defect-free. Additionally, many of these ICs suffer from run-time defects that arise during in-field operations. Thus, they are often appended with test structures to enable test on the fly.

Apart from the Boolean logic gates that perform the desired function, around 5% of gates are added in an IC design to facilitate IC testing. The cost of IC testing occupies around 20-30% of the overall cost of an IC that includes its design, fabrication, and testing [1].

Specialized test structures are added to an IC to support the test conducted at the test facility. These test structures enable the control and observation of the signals internal to an IC. Many design-for-testability (DfT) techniques that achieve high test quality in a cost-effective manner exist [2]. The most commonly used DfT technique is scan testing [1]. In scan testing, the flip-flops in the design are connected to form one or more scan chains that enable access to internal nodes in the IC. At the test facility, an IC is connected to the Automatic Test Equipment (ATE), which stores the test data in its memory. Test patterns are shifted in through the scan chains; the test responses are shifted out and compared with the expected responses.

The test patterns and the expected responses are computed using *Automatic Test Pattern Generation (ATPG)* algorithms [3]. The objective of the ATPG algorithms is to achieve maximum test quality at minimum test cost, which includes the DfT hardware, the test data volume, the time required for testing, and the power consumed during test [1]. Test quality is measured in terms of the *fault coverage*, which is the percentage of faults in the circuit that can be detected by the test data. Effectively, scan-based test structures turn every flip-flop into a one-bit input-output unit during testing, (1) enabling the use of computationally-efficient *combinational* ATPG algorithms to generate test data on sequential designs, and (2) attaining high test quality as well.

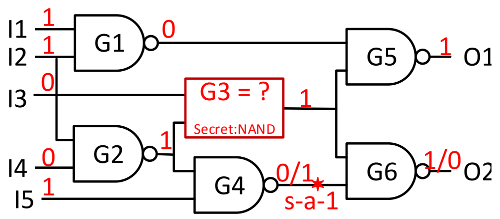


Fig. 1. A test vector to detect the G4 stuck-at-1 fault. ‘1/0’ at the output O2 indicates that the circuit output for fault-free/defective circuit are 1 and 0, respectively. The test vector 11001 along with the expected response 11 is provided to the test facility to test all the manufactured ICs (by a test set including this test vector). The test data implies that G3 cannot be NOR.

Test data, generated under the assumption that the target IC contains test structures that deliver deep access into the design, naturally embeds critical information about the design. An attacker in the test facility can therefore maliciously *extract design secrets by exploiting test data*, though such secrets are

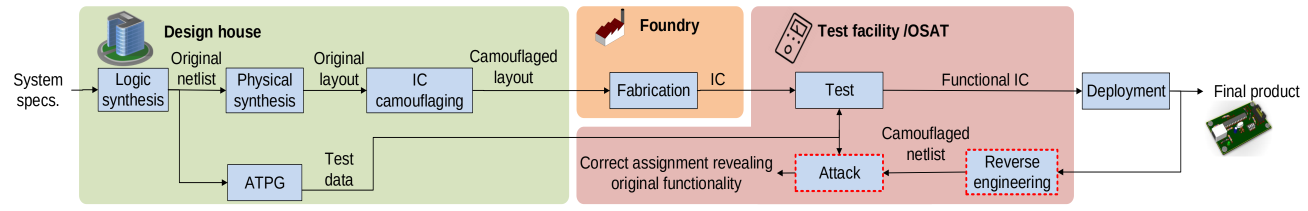


Fig. 2. IC camouflaging in the IC design flow. The test data generated during ATPG is sent to the test facility and used during the test. An adversary in the test facility can misuse data to compromise the security of IC camouflaging.

TABLE I. DEFINITION OF KEY TERMS USED.

Term	Description
Original netlist	A network of Boolean gates obtained after logic synthesis. ATPG is conducted on the original netlist to generate test data.
Original layout	The geometric (GDS-II) representation of the original netlist that is obtained after physical synthesis.
Camouflaged layout	Selected gates in the original layout are replaced with their camouflaged counterparts to create *camouflaged layout*.
Functional IC	The ICs that pass the manufacturing test conducted at the test facility. These ICs are deployed in electronic products. The scan infrastructure of these ICs may be locked.
Camouflaged netlist	The gate-level netlist obtained after imaging-based reverse engineering a functional IC. The functionality of camouflaged gates is unknown in this netlist.
Correct assignment	An assignment refers to functionality assigned to the camouflaged gates (see Section III-B for an illustrative example). On correctly assigning the functionality of all the camouflaged gates in the camouflaged netlist, the camouflaged netlist becomes functionally equivalent to the original netlist.

not explicitly embedded in the test data.

Example: Figure 1 provides a circuit along with one test vector and its response. This test vector was generated for a particular fault in the netlist, but can be misused by the test facility to unveil the type of the gate G3 (design secret: NAND) that the IP owner wants to protect. For example, the adversary can easily rule out the possibility of G3 being a NOR gate, as that would result in an expected response of 10 rather than 11.

In this work, we describe why the test data can implicitly embed design-critical information and how it can be misused to undermine the security of the chip. More specifically, we demonstrate how an attacker can extract secret information using the test data; we call our attack *HackTest*. We demonstrate HackTest using IC camouflaging as a case study [4]–[7]. IC camouflaging is a technique that companies use to prevent layout-level reverse engineering. Figure 2 depicts IC camouflaging in the context of IC design, manufacturing, and test.

A. *Attack Model and Assumptions*

The cost of owning and maintaining a foundry has become expensive, forcing many design companies to outsource their fabrication process to foundries. Such outsourcing has introduced several security vulnerabilities, including hardware Trojans, reverse engineering, and piracy. Over the last decade, a gamut of solutions has been developed to detect and/or prevent attacks from a rogue element in the foundry [8]–[11].

Similar to outsourcing fabrication, many design companies have been outsourcing their testing and assembly phases to offshore companies, such as Amkor [12], ASE [13], SPIL [14], and STATS ChipPAC [15]. These companies are distributed throughout the globe. Unfortunately, the security implications of untrusted testing facilities have not been analyzed in great detail. In this paper, we analyze security implications of untrusted testing facilities on an IP protection technique, namely, IC camouflaging.

In IC camouflaging, the design secret is the functionality of the camouflaged gates. The technique assumes that the foundry is trusted and needs its cooperation to fabricate chips with camouflaged cells built in. IC camouflaging assumes that entities in the supply-chain post-fabrication are untrusted; IC end-users have been best suited for reverse engineering an IC. We define the important terms used throughout the paper in Table I.

In our threat model, the attacker or his/her accomplice is a rogue element in the test facility, consistent with the IC camouflaging threat model, with access to:

1) **The camouflaged netlist** obtained by reverse engineering the target IC; tools/techniques are available for this purpose [6], [7], [16].
2) **Test stimuli and responses** leaked from/by the test facility.

Table II highlights the assets needed by each of the IC Camouflaging attacks.

TABLE II. ASSETS EACH ENTITY HAS ACCESS TO AND CAMOUFLAGING ATTACK CLASSIFICATION BASED ON ACCESS TO THE REQUIRED ASSETS. ASSET REQUIRED (NOT REQUIRED) FOR AN ATTACK IS MARKED WITH A ✓ (✗).

Entity	Test facility	End-user	Reverse Engineer
Asset	Test data	Functional IC	Camouflaged netlist
Sensitization [17]	✗	✓	✓
DeCamo [18], [19]	✗	✓	✓
HackTest	✓	✗	✓

B. *Why HackTest is More Dangerous*

All attacks, including the previous ones [20], [18] as well as the proposed attack, need the reverse-engineered (camouflaged) netlist for simulations. While previous attacks on IC camouflaging [20], [18] have used the functional IC as oracle assuming physical access to the test structures on the IC, HackTest uses only the test data, which already contains the fruits of such access; test data is generated by the designer with deep access into the netlist. Yet HackTest does not require physical access to the test structures on the IC, which are often times protected.

Physical access to the chip through its test infrastructure is available only at the test facility, as this access is blocked/restricted upon the completion of the test. Therefore, this leaves the existing attacks with a limited window of opportunity, during which it is very difficult to obtain the reverse-engineered netlist. First, all tested chips, failing or passing, must be returned to the designer, resulting in no chips to reverse engineer. Second, by the time the chips are available in the market and one can be obtained for reverse engineering, it is too late to launch the attack as the access to test infrastructures is no longer available, i.e., the window of opportunity has already expired. The *simultaneous* access to oracle (working chip) and reverse-engineered netlist that the existing attacks need is therefore unrealistic; *the existing attacks are difficult to launch.*

The proposed attack, on the contrary, has no deadlines, as it does not require physical access to the chips (or its test infrastructure). All it needs is test data in addition to the reverse-engineered netlist that contains camouflaged gates. When the reverse-engineered netlist becomes available, which can be long after the testing of chips has been completed by the test facility, the *leaked test data* can be used to successfully obtain the functionality of the camouflaged gates. *The proposed attack is therefore more flexible and realistic.*

C. *Contributions*

The contributions of this paper are:

1) We highlight the security vulnerabilities associated with the test data and the ATPG algorithms. We show that HackTest can break camouflaged benchmark circuits of realistic logic depth *within two minutes*.
2) We demonstrate that HackTest can extract the correct circuit assignment for a camouflaged IC:
   a) irrespective of the camouflaged gate-selection technique,
   b) even when the tools/algorithms employed for ATPG and for the attack are different,
   c) and despite the presence of industrial scan compression circuitry, i.e., when applied on compressed test data.
3) We demonstrate potential countermeasures that a designer can employ in an effort to thwart HackTest, albeit at the loss of test quality.

II. Preliminaries: Testing

A. *Test Pattern Generation*

The test patterns used in IC testing are generated by ATPG algorithms, which aim to maximize fault coverage. To model the physical defects, such as a short or an open, various fault models are used. The most prevalent model is the single stuck-at fault model. This model assumes that the fault location is tied to a logical value (0 or 1), and at most a single fault can be present in the IC.

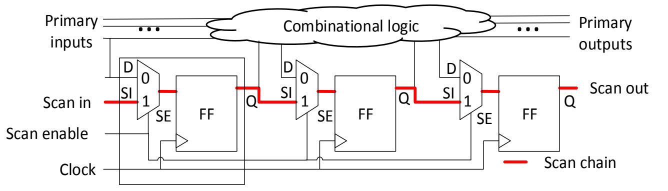


Fig. 3. An example scan chain with three scan cells. The flip-flops are configured into scan cells by adding multiplexers at the flip-flop inputs. The signal SE selects between the shift operation and normal operation.

Detection of a stuck-at fault involves *fault activation* and *fault propagation*. Fault activation necessitates setting the fault location to a value opposite to that of the stuck-at value, e.g., a value of 1 to detect a s–a–0 fault. Fault propagation entails forwarding the effect of the fault along a *sensitization path*[1] to a primary output. An input-output pattern that detects a given fault by accomplishing both fault activation and propagation is referred to as a *test pattern*.

For example, the G4 s–a–1 fault in the circuit shown in Figure 1 is activated by setting G4 to 0. To propagate the fault to the primary output O2, G3 must be 1. An input pattern that detects the fault is 11001. The output O2 will be 1 in the fault-free circuit and 0 in the presence of G4 s–a–1 fault; this is represented using the notation 1/0. A single test pattern can detect multiple faults, which are assumed to occur at most one at a time in the *single* stuck-at fault model. ATPG algorithms aim at maximizing the number of faults detected per pattern, in order to reduce the number of test patterns, and hence, the volume of the test data and test time.

While the traditional ATPG algorithms, such as D-algorithm, PODEM, and FAN, have focused on structural properties of a circuit [1], modern ATPG algorithms make use of techniques such as Boolean satisfiability [21]. The ATPG algorithms can be applied directly and scale well for combinational circuits; however, the computational complexity of these algorithms increases significantly for sequential circuits, where it is difficult to control and/or observe internal signals because of the presence of memory elements. Specialized DfT structures, such as scan chains, are inserted in the sequential circuits to improve the controllability and observability of the internal signals, enabling efficient ATPG.

B. *Scan-Based Testing*

DfT structures are inserted in an IC, early in the design cycle, to enable high-quality testing. The most commonly deployed DfT structures are the scan chains. In scan testing, the flip-flops in a design are reconfigured as scan cells. Effectively, every flip-flop is controllable and observable via

[1]Sensitization of a net to an output denotes the bijective mapping between the two.

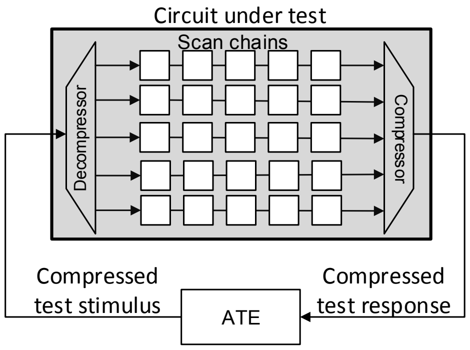


Fig. 4. Test data compression and decompression to reduce test data volume. Single channel supports five scan chains; $CR = 5$.

shift operations. *Consequently, test generation algorithms can treat the flip-flops as inputs and outputs*; the sequentiality is therefore eliminated, enabling the use of combinational test generation algorithms at reduced computational complexity.

As shown in Figure 3, each scan cell comprises a flip-flop and a multiplexer. The select line of the multiplexer, scan enable (SE) signal, decides the inputs to the flip-flops. When SE = 1, the flip-flops behave as a shift register, and each flip-flop is loaded with the output value of the previous flip-flop in the scan chain. Test operations involve 1) scanning in the test pattern, 2) capturing the response of the combinational logic into scan cells, and 3) scanning out the captured response. The scanned-out response is then compared with the expected response to decide whether the IC is defective or functional [1].

C. *Test Compression*

The test patterns are applied to the IC using an ATE. For larger designs, test data volume and the ATE pin count can become enormous, adding to the overall production cost. On-chip compression and decompression circuitry are used to reduce the pin count and the test data volume. Scan chains are accessed through the compression and decompression circuits, which reduce the test data volume and test time. As shown in Figure 4, the decompressor decompresses the stimulus coming from the tester and broadcasts/spreads it to the scan chains; the compressor compresses the responses prior to sending them to the ATE [22]. Various test compression schemes are available, which include code-based and linear schemes [22]. Linear schemes employ only XOR-networks and Linear Feedback Shift Registers (LFSRs).

The compression ratio, CR, is denoted as the ratio of the number scan chains to the number of input (or output) pins attached to the tester. In every cycle, the compressor compresses the response in one group of scan cells, while the decompressor simultaneously expands compressed stimulus into one group of scan cells. This group of scan cells is referred to as a *scan slice*. The number of scan slices is referred to as the *scan depth*.

While test compression reduces the test data volume, it also leads aliasing and encodability problems due to reduced controllability and observability [23], [24]. *Aliasing* refers to the phenomena where the compressor maps multiple responses onto the same value, leading to a lossy compression; a reduction in the fault coverage is the end result as faulty responses corresponding to some of the faults can no longer be differentiated from the expected responses at the output of the decompressor [23]. On the decompressor side, uncompressed test stimulus is computed as a linear combination of the input bits (i.e., compressed stimulus). As a consequence, certain input patterns may not be *encodable* through the decompressor. Because of the exclusion of the unencodable input patterns from the test input space, certain faults may remain undetected, resulting in a loss of fault coverage [25]. Consequently, test compression-induced controllability and observability loss reflects into test quality loss; this loss increases for larger CR values (more aggressive compression).

III. Preliminaries: IP protection

A. *Reverse Engineering*

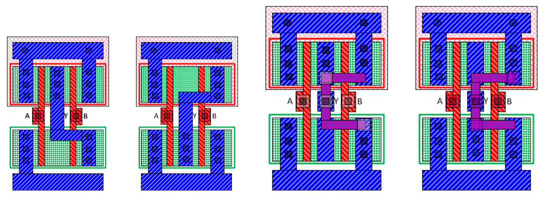
IP piracy is a major concern for the semiconductor industry, which loses billions of dollars each year due to IP infringement [26]. A major enabler for IP piracy attacks is reverse engineering [27], [28]. Reverse engineering an IC involves de-packaging, delayering and imaging the individual layers, annotating the images, and extracting the netlist of the design [27]. Reverse engineering can identify IC's functionality, the device technology used in the IC, or its design features [27]. Many commercial ICs, such as TI 4377401 baseband processor an Intel's 22nm Xeon processor, have been reported to be successfully reverse engineered [6], [7]. Commercial and open-source tools for reverse engineering are available [29], [16].

B. *IC Camouflaging*

IC camouflaging is a layout-level countermeasure against imaging-based reverse engineering [4], [5]. It introduces cells that look alike from the top view, but can implement one of many functions. On reverse engineering a camouflaged IC, an attacker cannot infer the correct functionality of the camouflaged cells by inspecting the layout through imaging techniques [27]. Selected gates in the layout can therefore be replaced with their camouflaged counterparts in order to generate ambiguity for a reverse engineer. Camouflaged cells incur higher area, power, and delay overhead over their regular counterparts, thus constraining the number of gates that can be camouflaged [20].

IC camouflaging can be performed by using dummy contacts [4], filler cells [5], or diffusion programmable standard cells [30], [31]. IC camouflaging using dummy contacts is illustrated in Figure 5, where each camouflaged cell can implement one of two functions: NAND or NOR.

Example. An example of a camouflaged circuit is shown in Figure 6. The original circuit is denoted as C_{orig}, and its camouflaged version as C_{camo}. Both C_{orig} and C_{camo} have n inputs and m outputs. L represents the set of possible functionalities that a camouflaged gate can implement. k denotes the number of gates that have been camouflaged.



(a) (b) (c) (d)

Fig. 5. Layout of typical 2-input (a) NAND and (b) NOR gates. The metal layers look different from the top, and it is easy to distinguish by visual inspection. Camouflaged layout of 2-input (c) NAND and (d) NOR gates [20]. The metal layers are identical, and the two gates cannot be distinguished from the top view. Source: [20].

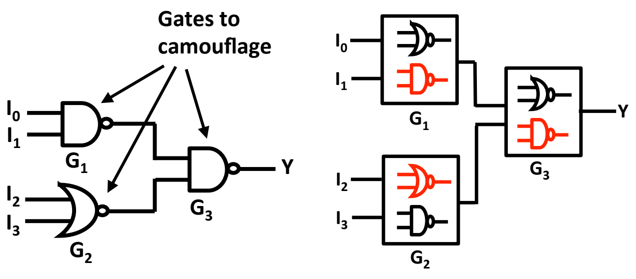


Fig. 6. a) Original circuit C_{orig}, b) Camouflaged circuit C_{camo}, with each gate implementing either NAND or NOR. Gates in red depict the actual functionality. Source: [18].

For C_{camo} in Figure 6, $n = 4$, $m = 1$, and $k = 3$. Further, L = {*NAND,NOR*}, i.e., a camouflaged gate will be either a NAND or a NOR. The correct function of each camouflaged gate is illustrated in red.

The number of possible functions that C_{camo} can implement is $|\mathrm{L}|^{\mathrm{k}}$, only one of which is the function implemented by C_{orig}[2]. An *assignment* refers to the mapping of a function from the set L to one of the camouflaged gates in C_{camo}. The set of assignments to all the camouflaged gates in C_{camo} is referred to as a *circuit assignment*. A circuit assignment that leads to a correct circuit output for all inputs i, i.e., $\forall i \in \{0,1\}^n, C_{camo}(i) = C_{orig}(i)$, is referred to as a *correct circuit assignment*[3]. In Figure 6(b), *(NAND,NOR,NAND)* is the correct circuit assignment for the camouflaged gates (G_1, G_2, G_3), respectively.

C. Camouflaged Gate-selection Techniques

An important step in IC camouflaging is to select the gates to be camouflaged. *Random selection (RS)* of gates is not secure [20] (elaborated in Section III-D). To increase security, *clique-based selection (CBS)* camouflages a set of gates such that the output of a camouflaged gate in the set cannot be sensitized to an output without accounting for the other camouflaged gates in the set; this set of camouflaged gates is referred to as a clique [20]. *Output corruption-based selection (OCS)* maximizes the corruption at the circuit's output when an attacker makes a random circuit assignment, by selecting gates that have high impact on circuit's outputs. CBS and OCS can be integrated to maximize both clique size and output corruptibility, resulting in *OCS+CBS* [20].

D. Attacks on Camouflaging

Sensitization attack utilizes the VLSI test principle of sensitization to break random IC camouflaging [20]. Sensitization of a net requires setting the side inputs of each gate on the path from the wire to the output to their non-controlling values[4]. The attack needs (1) a functional IC: the IC that has passed the manufacturing test, and (2) a *camouflaged netlist*: the netlist obtained through reverse engineering the IC. In the camouflaged netlist (e.g. the netlist shown in Figure 1), the functionality of the camouflaged cells is unknown. The attacker analyzes the camouflaged netlist and computes the input patterns that sensitizes the output of a camouflaged gate to an output. By applying the computed patterns to the functional IC and analyzing the responses, the attack can infer the correct assignment to the camouflaged gates, iteratively.

DeCamo attack breaks all existing camouflaged gate-selection techniques using Boolean satisfiability (SAT) based techniques[5] [18], [19]. DeCamo attack, similar to the sensitization attack, needs a functional IC and a *camouflaged netlist*. DeCamo attack generates and uses *discriminating input patterns (DIP)* [18]. Each DIP, when used in conjunction with the correct output of the functional IC, has the ability to eliminate one or more incorrect circuit assignments. By repeatedly applying the DIPs, an attacker can eliminate all incorrect assignments and find the correct assignment [18], [19]. The computation of DIPs can be formulated as a Boolean formula, which can be solved using a SAT solver. In each iteration of the attack, the Boolean formula grows as new clauses based on the previous DIPs are appended to it. The number of DIPs needed for a successful DeCamo attack depends on the camouflaged circuit under attack and dictates the attack time.

IV. HACKTEST-V0 – BASIC SCAN

As the ATPG process is dependent on the structure of the netlist, information about the netlist structure is embedded in the generated test patterns. A designer who wants to keep certain nodes in the circuit as a secret is faced with the problem of information leakage through these test patterns. The existing ATPG algorithms are not designed to protect design secrets. Instead, they have been developed to expose/reveal maximum information about the circuit structure for every test pattern, so that a high fault coverage can be achieved using a smaller number of test patterns. The test data is therefore the key enabler of HackTest.

[2]Here, we do not consider the associative, commutative, and distributive properties of the functions.

[3]An assignment that is not correct is referred to as an incorrect assignment.

[4]The controlling value of a gate, when applied to one of its inputs, determines the gate output regardless of the values applied to the other inputs of the gate. The non-controlling value of gate is the opposite of the controlling value, e.g., 1 for the AND gate and 0 for the OR gate.

[5]Decamouflaging refers to the identification of the functionality of a camouflaged gate.

A. Threat Model

The attacker has the following capabilities:

1) **Access to a camouflaged netlist.** The attacker obtains the gate-level camouflaged netlist by reverse engineering the target IC. To this end, he/she can use existing reverse-engineering techniques [6], [7] and tools [16].
2) **Information on test structures.** From the netlist, he can identify the test structures: scan chains, compressor, and decompressor. The structure of scan flip-flops is different than that of combinational logic gates, thus making them easier to detect. Furthermore, their connectivity through wires and buffers reveals the scan-chains.
3) **Test stimuli S and responses R.** An attacker in the test facility can access the test stimuli and their expected responses because the designer has provided them to the attacker.

B. Attack Methodology

The objective of the attacker is to determine the correct assignment of the camouflaged circuit using the knowledge of test data and test structures. To achieve the objective, he/she performs the following steps:

1) Generate an equivalent gate-level netlist $CamoCkt$ from the camouflaged netlist C_{camo}, where the possible assignments to the camouflaged gates are represented using *assignment vector* A [18], [19].
2) Apply the test stimuli S as input constraints, the test responses R as output constraints to $CamoCkt$, and solve for the assignment vector A that satisfies the given I/O constraints and maximizes the fault coverage FC under the constraints, as represented in Equation 3.

Problem formulation. Let A be the correct assignment of functionalities to a camouflaged circuit $CamoCkt$. In this paper, the type of fault $t \in \{s\text{-}a\text{-}0, s\text{-}a\text{-}1\}$, while it can easily be extended for other fault models as well (see Section IX). A fault $f_{g,t}$ at the output of a gate g of type t is detected, if there exists an input i for which the outputs of fault-free circuit and circuit with fault $f_{g,t}$ are different. Detectability of a fault $f_{g,t}$ is

$$\mathrm{fd_{g,t}} = \begin{cases} 1 & \exists i \; CamoCkt(i, A, \cdot) \oplus CamoCkt(i, A, f_{g,t}) \\ 0 & otherwise \end{cases} \tag{1}$$

The fault coverage FC for the camouflaged circuit with N gates and T types of faults is

$$FC = \frac{\sum\limits_{i=1}^{N} \sum\limits_{j=1}^{T} \mathrm{fd_{i,j}}}{N \times T} \tag{2}$$

The attack is an optimization problem: the objective is to maximize the fault coverage FC with M test stimuli (S) and responses (R) and described as follows:

$$\begin{aligned} \text{maximize} \quad & FC \\ \text{subject to} \quad & CamoCkt(S_1, A, \cdot) = R_1 \\ & CamoCkt(S_2, A, \cdot) = R_2 \\ & \vdots \\ & CamoCkt(S_M, A, \cdot) = R_M \\ \text{solve for} \quad & A \end{aligned} \tag{3}$$

Equation 3 formulates a system of Boolean equations that can be solved using techniques such as mixed integer linear programming. ATPG algorithms are capable of solving a system of Boolean equations while simultaneously maximizing fault coverage even in the presence of unknown values; ATPG is, therefore, a natural candidate for solving the optimization problem in Equation 3. Here, the unknown values are the assignments to the camouflaged gates. Computing a set of test patterns that maximizes the fault coverage in a circuit is an NP-hard problem [32]. However, practical circuits exhibit certain structural properties, such as limited circuit depth, that make it possible to solve the ATPG problem efficiently [33].

The rationale for the attack to be successful is:

1) In ATPG, the objective is to maximize the fault coverage through minimal amount of test data. ATPG applied on the original netlist produces test patterns that will maximize the fault coverage for the correct circuit assignment. The same set of test patterns may fail to detect certain faults when an incorrect assignment is made to the circuit, leading to a reduction in the fault coverage. Thus, an attacker can use fault coverage as a guiding metric for the attack.
2) In IC camouflaging, the test patterns are generated by conducting ATPG on the original netlist, as depicted in Figure 2. Thus, the expected test responses match the correct IC output and can provide a hint to the attacker in distinguishing incorrect assignments from the correct one. Thus, an attacker can use test stimulus-response pairs to guide the attack.

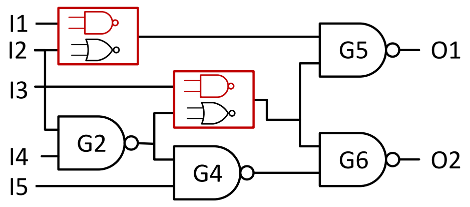


Fig. 7. Camouflaged circuit C_{camo}, with each gate implementing either a NAND or a NOR. Gates in red depict the correct assignment to the camouflaged gate.

Example. On performing ATPG on the circuit shown in Figure 7, six test patterns are generated, as listed in Table III; the corresponding stuck-at fault coverage is 100%.

In the camouflaged circuit shown in Figure 7, two gates, G1 and G3, are camouflaged using NAND/NOR camouflaged cells. There are four possible circuit assignments. The correct assignment is $(NAND, NAND)$. Table IV reports the fault

TABLE III. TEST PATTERNS FOR THE NETLIST IN FIGURE 7.

Stimulus (S)	Response (R)
10100	10
11010	11
10011	01
11100	00
01101	11
01111	00

coverage for different assignments to the camouflaged gates by using the test patterns listed in Table III. The attacker computes the fault coverage using the test stimuli and responses. As shown in the table, the fault coverage is maximum for the correct assignment, because the test data has been generated to maximize fault coverage for the original netlist (i.e., the correct assignment). Therefore, an attacker can use fault coverage as a metric to guide his/her attack and extract the correct circuit assignment.

TABLE IV. FAULT COVERAGE ACHIEVED FOR DIFFERENT ASSIGNMENTS TO THE NETLIST IN FIGURE 7. CORRECT ASSIGNMENT: (NAND,NAND).

G1	G2	Fault coverage (%)
NAND	NAND	100
NAND	NOR	90.9
NOR	NAND	90.9
NOR	NOR	59.1

C. *Computational Complexity of HackTest*

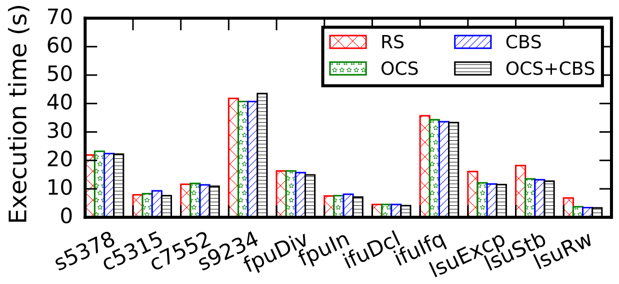
Theorem 1: The complexity of HackTest is NP-hard.
Proof: See Appendix 1.

D. *Experimental Results*

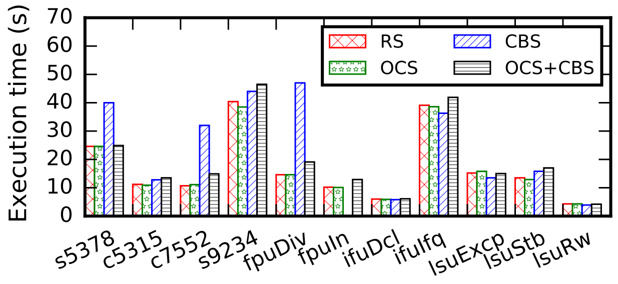
Experimental setup. We performed HackTest on ISCAS [34] benchmark circuits and the controllers of OpenSPARC processor [35]. In the OpenSPARC processor, `fpuDiv` is the controller of the floating-point divider, and `fpuIn` manages the operands of the floating-point divider. `ifuDcl` and `ifuIfq` are in the instruction fetch unit of the processor controlling the decoder logic and fetch queue, respectively. `lsuExp`, `lsuStb`, and `lsuRw` are in the load-store unit managing the exceptions, store-buffer units, and the read-write units. These circuits have been used to benchmark the performance of traditional and modern ATPG tools, as the circuit structure is representative of the industrial circuits [36].

TABLE V. OUTPUT OF THE NETLIST SHOWN IN FIGURE 7 FOR DIFFERENT TEST PATTERNS. EACH COLUMN REPRESENTS A CIRCUIT ASSIGNMENT. THE INCORRECT OUTPUTS ARE SHOWN IN GRAY.

Test stimulus	Assignment			
	NAND, NAND	NAND, NOR	NOR, NAND	NOR, NOR
10100	10	11	10	11
11010	11	11	11	11
10011	01	01	11	11
11100	00	01	10	11
01101	11	11	11	11
01111	00	01	10	11



(a)



(b)

Fig. 8. Execution time (s) of HackTest-v0 on circuits with a) 64 camouflaged gates b) 128 camouflaged gates. The attack completes within one minute for any given circuit.

Table VI lists the circuits used in experiments, number of gates in each circuit (# gates), and the logic depth of the circuit. Logic depth denotes the maximum number of gates on any path between two flip-flops in a circuit. The higher the logic depth, the higher the complexity of ATPG.

The number of faults for the circuit, the number of test patterns generated during the ATPG (# patterns) and the corresponding fault coverage values are also shown, assuming full scan. It can be noted that the ATPG achieves almost 100% fault coverage in almost all the circuits. The number of camouflaged gates in each circuit is either 64 or 128. Synopsys Tetramax ATPG [37] is used to generate the test patterns during the ATPG phase and to perform the attack described in Equation 3. Since, in IC camouflaging, the ATPG is performed on the original netlist, the same set of test patterns will be generated and sent to the test facility, regardless of the:

- IC camouflaged gate-selection technique, such as RS, CBS, and OCS,
- technique[6] employed for camouflaging, such as dummy contacts or programmable cells, and
- number of gates camouflaged.

HackTest results. The success of HackTest is measured by the number of camouflaged gate assignments that are retrieved correctly. **HackTest successfully retrieves the correct assignment for 100% (64 out of 64, and 128 out of 128) of the camouflaged gates in each circuit, irrespective of the camouflaged gate-selection technique.** This is because

[6]Although camouflaged cells have different physical structures compared to standard cells, test generation algorithms/tools target faults on the circuit wires. Thus, the same test data is generated irrespective of the camouflaged cell structures.

TABLE VI. Statistics of the benchmarks.

Benchmark	# gates	Logic depth	# faults	# patterns	Fault coverage (%)
s5378	1110	21	7012	245	99.9
c5315	1252	30	7770	118	99.9
c7552	1290	48	7820	165	99.4
s9234	1573	27	9672	331	99.9
fpuDiv	1197	34	6836	181	100
fpuIn	916	37	5234	124	100
ifuDcl	771	40	4384	110	100
ifuIfq	2027	36	11290	267	100
lsuExp	936	42	5640	168	100
lsuStb	1360	38	7566	174	100
lsuRw	899	26	5338	65	100

HackTest exploits the principle behind the ATPG algorithms, i.e., maximizing fault coverage.

The execution time of HackTest is shown in Figure 8 for circuits with 64 and 128 camouflaged gates. The execution time is less than a minute for all the circuits, though it varies across the benchmarks and for different camouflaged gate-selection techniques. The attack is formulated as a constrained optimization problem; the execution time of the attack depends on the number of patterns (shown in Table VI), which dictates the number of constraints. The execution time for `s9234` is the highest as its number of test patterns is also the highest. The circuit `ifuIfq`, which has the highest number of gates and the second highest number of test patterns, has the second highest execution time.

As the number of camouflaged gates is increased from 64 to 128, HackTest's execution time increases by 24% on average. The maximum increase is observed for circuits camouflaged with the CBS technique — 44% on average; in CBS, camouflaged gates maximally interfere each other, resulting in linearly inseparable constraints.

E. *Special Case: Attack ATPG $\neq$ Test Generation ATPG*

In practical scenarios, the attacker does not know which ATPG tool has been used for test pattern generation, or an attacker may not have access to the same ATPG tool. To illustrate the effectiveness of HackTest in these scenarios, we generated test patterns using Atalanta, an open-source ATPG tool [38], and performed HackTest using Synopsys Tetramax ATPG [37]. **HackTest is again successful on 100% of the circuits**. The execution time of HackTest for circuits with 128 camouflaged gates is shown Figure 9. The ratio of execution time of HackTest when performed using Atalanta and when performed using Tetramax is around 0.9 for most circuits, indicating that the attack time slightly decreases. This is because, compared to Tetramax, Atalanta generates fewer test patterns, as listed in Table VII.

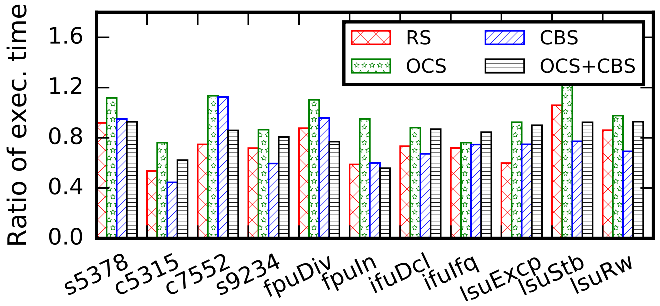


Fig. 9. Execution time of HackTest-v0 for test patterns generated using Atalanta normalized with respect to the execution time when the patterns are generated using Tetramax. The attack is launched using Tetramax.

V. HackTest-v1 – test compression

A. *Threat Model*

Till now, we assumed that the attacker has access to raw test data that is loaded to the IC. Modern ICs employ compressor and decompressor circuits at the start and the end of the scan chains. Thus, HackTest needs to be modified to operate on the compressed test data.

The compressor and decompressor structures are public knowledge and can be identified easily [1]. Having proprietary test infrastructure does not guarantee security, as even such structures have been successfully reverse engineered [39]. Let C and D be the compressor and decompressor functions, respectively, obtained by reverse engineering these structures. Now, detectability of a fault $f_{g,t}$ is

$$\mathrm{fd_{g,t}} = \begin{cases} 1 & \exists i \; C(CamoCkt(D(i), A, \cdot)) \oplus \\ & C(CamoCkt(D(i), A, f_{g,t})) \\ 0 & otherwise \end{cases} \tag{4}$$

B. *Attack Methodology*

As explained in Section II-C, the decompressor and the compressor lead to controllability and observability loss, which degrades test quality. From the attacker's perspective, the decompressor poses no inconvenience, as it can be reverse engineered, and the uncompressed data that the scan chains receive can be computed from the compressed stimuli. The attacker can therefore perform the attack as if decompressor was absent. The compressor, on the other hand, hampers the attack. The secret that the test data leaks is transformed by the compression operation, resulting in an extra effort for the attacker. The attacker has to effectively consider the compressor as part of the netlist under attack. The compressor needs to be instantiated within the netlist as many times as the number of scan slices, as response fragments in slices are compressed individually (for combinational compressors). The attack complexity is therefore expected to increase and the success rate

TABLE VII. THE RATIO OF # OF PATTERNS GENERATED BY ATALANTA TO THE # OF PATTERNS GENERATED BY TETRAMAX.

Benchmark	s5378	c5315	c7552	s9234	fpuDiv	fpuIn	ifuDcl	ifuIfq	lsuExcp	lsuStb	lsuRw
# patterns	0.82	0.75	0.63	0.69	0.72	0.75	0.74	0.74	0.68	0.67	0.63

TABLE VIII. IMPACT OF TEST COMPRESSION ON # PATTERNS. A COMPRESSION RATIO OF 1 IMPLIES NO COMPRESSION.

Benchmark	1	10	20	30
s5378	245	63	41	30
c5315	118	47	29	18
c7552	165	61	41	28
s9234	331	153	96	61
fpuDiv	181	44	32	21
fpuIn	124	22	19	12
ifuDcl	110	32	17	11
ifuIfq	267	58	44	30
lsuExp	168	52	30	23
lsuStb	174	66	26	17
lsuRw	65	38	28	25

TABLE IX. IMPACT OF TEST COMPRESSION ON FAULT COVERAGE (%). A COMPRESSION RATIO OF 1 IMPLIES NO COMPRESSION.

Benchmark	1	10	20	30
s5378	99.9	62.0	57.1	53.1
c5315	99.9	73.7	68.8	63.3
c7552	99.4	77.8	69.4	61.0
s9234	99.9	78.2	70.7	60.4
fpuDiv	100	66.0	63.9	55.8
fpuIn	100	49.5	45.4	43.6
ifuDcl	100	68.8	60.9	54.5
ifuIfq	100	51.5	47.8	44.3
lsuExp	100	72.2	60.7	56.3
lsuStb	100	70.9	57.4	51.9
lsuRw	100	82.9	77.8	73.6

is expected to decrease. The more aggressive the compression (the larger the CR), the more challenging the attack is expected to become. HackTest constraints in the presence of compressor and decompressor are $\underset{1\leq i\leq M}{\forall} C(CamoCkt(D(S_i), A, \cdot)) = R_i$. The rest of the HackTest formulation remains the same.

C. *Experimental Results*

We used CR values of 10, 20, and 30, and camouflaged 64 gates. The compressor and the decompressor are XOR networks. The experimental setup is the same as described in Section IV-D. We first report the number of test patterns and the fault coverage as a function of CR; these parameters are independent of camouflaged gate-selection techniques. Table VIII presents the number of test patterns generated for different values of CR, and Table IX presents the corresponding fault coverage. The trend with more aggressive compression is a decrease in fault coverage as well as the number of patterns. The average number of patterns is 177, 57, 36, and 14 for compression ratio of 1, 10, 20 and 30, respectively. The corresponding average fault coverage is 99.9%, 68.5%, 61.8%, and 56.2%, respectively.

Table X lists the attack success when the ATPG and test are conducted in the presence of scan compression. Attack success is shown in terms of the number of assignments that are retrieved correctly by the attack. On average, the attack retrieves 52 out of 64 assignments for $CR = 10$. The attack success reduces drastically for CR values of 20 and 30, with the average number of assignments retrieved being 32 and 26. This can be attributed to the associated loss in fault coverage with aggressive test data compression. **Thus, the attack success correlates well with test quality**; the higher the test quality, the higher the attack success rate. OCS [20] exhibits the highest resistance against the attack. The average number of assignments retrieved correctly in the case of OCS is only 30 for $CR = 10$.

The execution time of the attack, as a function of CR, is shown in Figure 10. The execution time decreases as CR gets larger, due to a smaller number of patterns. The execution time, on average, is the highest for OCS [20].

VI. POTENTIAL COUNTERMEASURE 1: HACKTEST-V2 (ON SECRET-OBLIVIOUS ATPG) – BASIC SCAN

A designer/defender may consider a simple countermeasure that hides the secret during the ATPG, resulting in test generation in a *secret-oblivious* manner. This way, the attacker will only be able to utilize the input-output constraints driven by the test vectors and expected responses. In this section, we evaluate if such a simple countermeasure can thwart HackTest. We also evaluate the ability of current ATPG tools to support such countermeasures.

Secret-oblivious ATPG can be performed by hiding the functionality of the camouflaged gates from the ATPG tool. They can be black-boxed for this purpose. The ATPG tool then sets the output of the camouflaged gates to unknown values, denoted as x's. The set of test vectors generated in the presence of x's will fail to attain the same fault coverage level as that obtained on the original netlist with all the secrets exposed to the ATPG tool. Test generation in the presence of unknown x's, and the associated controllability and observability challenges have been well studied and understood in VLSI testing [40].

The **example** in Figure 11 illustrates the challenging task of test generation in the presence of the unknown (secret) G3 functionality. The stuck-at-1 fault at the output of G4 remains undetected, as it cannot be propagated to O2 in the presence of the unknown generated by G3. The expected response will be 11 regardless of the functionality of G3, while the faulty response will depend on G3's functionality; chips that contain a defect corresponding to the targeted fault may or may not be detected depending on G3's functionality. The ATPG tool will conservatively assume that the fault remains undetected and make other attempts to detect it; for this netlist, there is no test that can detect the G4 stuck-at-1 fault in the presence of the unknown.

For the same example, the generated test pattern does **not** reveal any information about the secret, as the expected response given to the test facility is 11 irrespective of the functionality of G3. A test vector that propagates an x to the output as the expected response, however, leaks information

TABLE X. SUCCESS RATE OF HACKTEST-V1. NUMBER OF ASSIGNMENTS (OUT OF 64) RETRIEVED CORRECTLY BY THE ATTACK ON COMPRESSED TEST DATA FOR DIFFERENT VALUES OF CR.

Benchmark	RS [20]			OCS [20]			CBS [20]			OCS+CBS [20]		
	10	20	30	10	20	30	10	20	30	10	20	30
s5378	43	41	42	58	58	0	52	51	50	54	54	49
c5315	58	0	0	57	0	0	53	53	0	56	50	0
c7552	60	0	0	0	0	0	58	0	0	56	53	0
s9234	62	57	49	62	0	57	63	58	58	57	49	49
fpuDiv	50	53	42	61	59	0	53	0	0	51	51	47
fpuIn	50	43	38	54	0	0	43	0	35	47	33	38
ifuDcl	48	0	43	43	0	35	0	0	0	45	39	36
ifuIfq	48	53	53	57	54	0	54	51	0	46	45	41
lsuExp	60	54	49	63	0	0	54	0	0	50	50	43
lsuStb	53	40	42	50	0	0	53	0	0	49	52	44

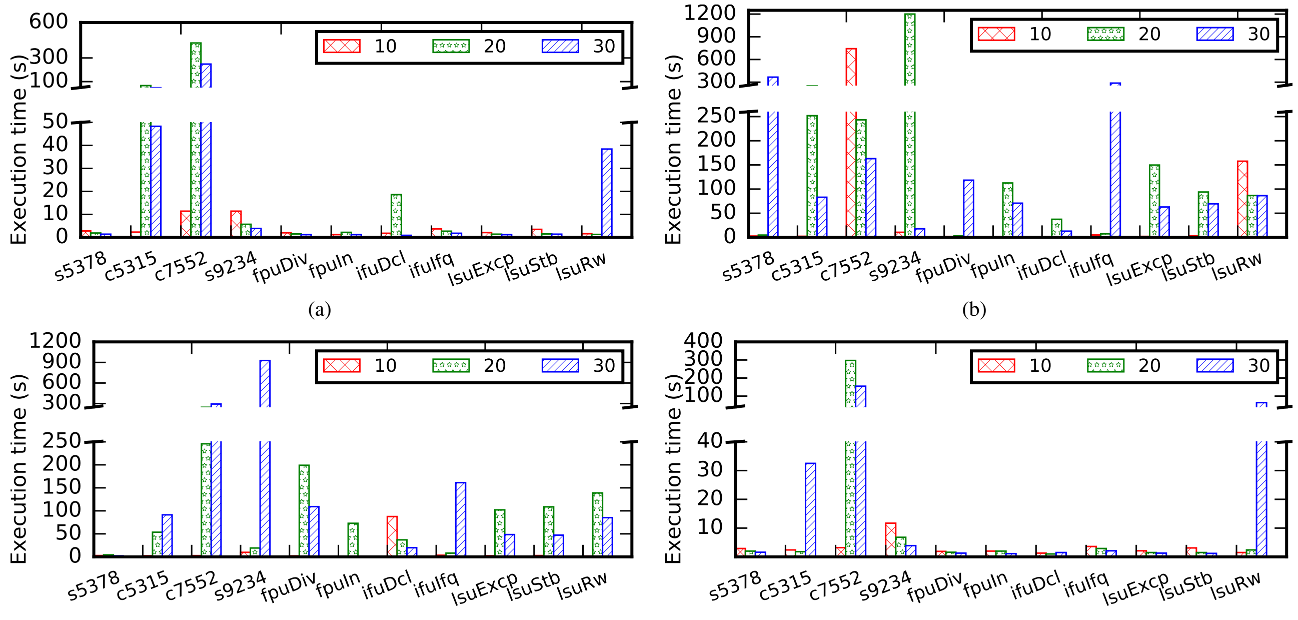


Fig. 10. Execution time of HackTest-v1 on compressed test data for different compression ratios. a) RS [20], b) OCS [20], c) CBS [20], and d) OCS+CBS [20].

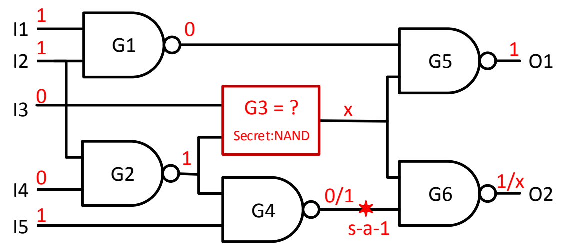


Fig. 11. Secret-oblivious ATPG. G3 is set to an unknown value, denoted as x. G4 stuck-at-1 remains undetected.

about the secret. The test facility will be provided with the responses with the x's turned into known values, so they can conduct the test; otherwise, x's in the responses would mask out defective chips, lowering test quality further.

From the security perspective, the attacker will then be obtaining a set of test vectors that maximizes fault coverage in a secret-oblivious manner. This will prevent the attacker from using the fault coverage maximization criterion to guide the attack. The test vectors and the expected responses may still provide useful information to the attacker – if and only if they propagate secret x's to the outputs in the expected response – and can still be used in the form of input and output constraints to guide the attack.

The threat model is identical to that in HackTest-v0. The attack formulation, however, is different. The attack problem becomes a decision problem that solves a set of constraints with no fault coverage maximization objective as follows.

$$
\begin{aligned}
&\text{solve for} \quad A \\
&\text{subject to} \qquad CamoCkt(S_1, A, \cdot) = R_1 \\
&\qquad\qquad\qquad CamoCkt(S_2, A, \cdot) = R_2 \\
&\qquad\qquad\qquad\qquad \vdots \\
&\qquad\qquad\qquad CamoCkt(S_M, A, \cdot) = R_M
\end{aligned}
\tag{5}
$$

Experimental results. As can be seen in Table XI, HackTest-v2 is almost 100% successful even with secret-oblivious

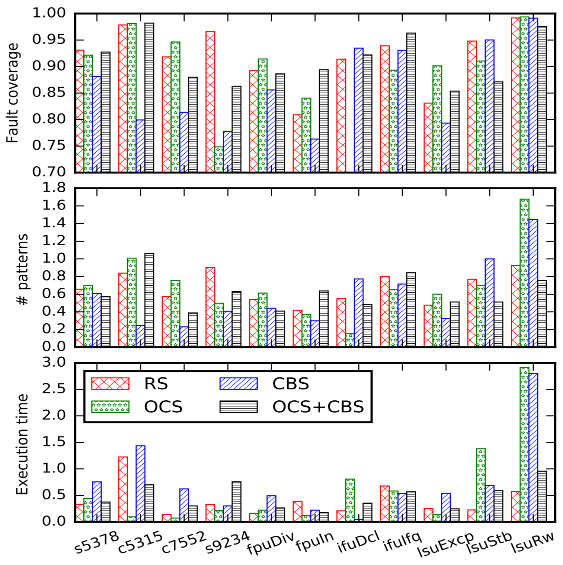


Fig. 12. Normalized fault coverage, # patterns, and execution time for HackTest-v2 on secret-oblivious ATPG. Fault coverage and # patterns are normalized with respect to the values shown in Table VI. Execution time is normalized with respect to the time displayed in Figure 8(a).

ATPG. Secret-oblivious ATPG affects the attack success for some of the circuits, where the number of assignments retrieved correctly is less than 64. The reason for the high attack success is the fact that secret-oblivious ATPG results in a lot of x's in the test responses. The designer specifies these x's for the test facility so the test can be applied in a meaningful way, indirectly revealing the design secret in the process.

The fault coverage for secret-oblivious ATPG is shown in Figure 12, normalized with respect to the fault coverage for the regular secret-aware ATPG, shown in Table VI. The fault coverage is 75% or higher in all the circuits, and is sufficient for the attack success. The same figure also displays the number of patterns, which is $\sim$50% of the number of patterns for regular ATPG.

TABLE XI. HACKTEST-V2 SUCCESS RATE IN TERMS OF NUMBER OF ASSIGNMENTS RETRIEVED CORRECTLY (OUT OF 64).

Benchmark	RS [20]	OCS [20]	CBS [20]	OCS+CBS [20]
s5378	64	64	64	64
c5315	64	64	64	64
c7552	63	64	64	60
s9234	60	64	64	63
fpuDiv	60	64	64	62
fpuIn	62	63	64	61
ifuDcl	63	64	62	59
ifuIfq	63	64	64	64
lsuExp	61	62	64	61
lsuStb	58	64	64	63
lsuRw	64	64	64	64

VII. Potential Countermeasure 2: HackTest-v3 (on secret-oblivious ATPG) – test compression

Unknown x's produced in a test compression environment further exacerbate the test quality loss [40]. When the attack is performed on a compressed test set produced in a secret-oblivious manner, the attack complexity is therefore expected to increase, lowering the attack success rate. The threat model is identical to that in HackTest-v1; however, the attack is a decision problem, rather than an optimization problem, as there is no fault coverage metric guiding the attack. The following constraints replace the ones in Equation 5:

$$\underset{1\leq i\leq M}{\forall} C(CamoCkt(D(S_i), A, \cdot)) = R_i.$$

Experimental results. Table XII lists the impact of secret-oblivious ATPG on fault coverage for compressed test data. For some designs, the fault coverage is zero, indicating that the constraints imposed on the ATPG to hide the secret are so restrictive that the ATPG is unable to detect any fault in the circuit. The number of patterns is also zero in these cases, and the attack is not successful as shown in Table XIII; the ICs cannot be tested either, and thus it is of no use to the designer. For $CR = 10$, the attack can retrieve 37 (out of 64) assignments, on average, compared to 51 assignments in the case of secret-aware compression-based ATPG (HackTest-v1). For the same CR value, the fault coverage achieved, on average, is only 56.0%, as compared to 68.1% for secret-aware compression-based ATPG.

VIII. Comparison with existing attacks

Beyond the limitations explained in Section 1.2, the sensitization and DeCamo attacks have several other limitations, as listed in Table XIV.

- **Sequentiality**. Neither the sensitization nor DeCamo attack is efficient on sequential circuits, even though their effectiveness has been illustrated on combinational circuits. These attacks have not been applied on sequential circuits, although most of the real-world designs are sequential circuits. To make these attacks applicable, design sequentiality can be overcome in two ways, each leading to a fundamental problem for the sensitization and DeCamo attacks.

  To eliminate sequentiality, these attacks can unroll the sequential circuit $d+1$ times, in a way similar to bounded model checking, where d is the diameter of the finite state machine (FSM) of the circuit [18]. However, an attacker, who is trying to decamouflage a circuit does not know the circuit FSM and consequently the value of d. Extracting the FSM for a sequential circuit from its netlist is an NP-complete problem [41]. d can be large in sequential circuits. As the circuit has to be unrolled d times, DeCamo and sensitization attacks do not scale well for sequential circuits. Moreover, the exact value of d is often computationally infeasible to determine and needs a QBF solver [42]. Approximations using recurrence diameter (rd) as upper bound on d are typically used. However, even computing rd for large circuits is NP-complete and computationally infeasible for large circuits even with use of heuristic approaches [43].

TABLE XII. Fault coverage for secret-oblivious ATPG. A fault coverage value of zero indicates that no faults could be detected in an attempt to hide the secrets.

Benchmark	RS [20]			OCS [20]			CBS [20]			OCS+CBS [20]		
	10	20	30	10	20	30	10	20	30	10	20	30
s5378	64.7	59.3	31.3	60.3	0	0	62.4	44.7	0	63.4	57.1	0
c5315	63.6	50.4	39.5	31.2	0	0	52.7	36.1	0	63.0	57.9	0
c7552	67.5	45.0	49.8	58.5	48.4	0	56.5	47.7	0	70.5	47.7	37.7
s9234	70.6	64.7	34.8	54.6	37.9	0	57.4	0	0	74.4	61.9	0
fpuDiv	62.0	56.8	38.0	44.2	0	0	53.6	0	0	62.8	46.3	0
fpuIn	39.6	28.4	0	20.8	0	0	37.9	0	0	32.1	25.0	0
ifuDcl	50.8	36.7	0	38.9	31.1	0	38.0	16.3	0	47.6	0	0
ifuIfq	53.8	50.2	36.9	48.8	0	0	52.5	45.1	0	52.3	48.8	0
lsuExp	68.8	44.9	45.1	45.6	22.4	22.4	53.0	0	0	66.5	54.9	0
lsuStb	56.5	47.9	36.1	53.2	36.0	36.0	56.3	37.4	0	55.8	45.7	36.0
lsuRw	82.0	72.8	38.5	66.4	36.6	0	76.9	42.1	0	74.5	36.3	0

TABLE XIII. HackTest-v3 success rate in terms of number of assignments retrieved correctly (out of 64).

Benchmark	RS [20]			OCS [20]			CBS [20]			OCS+CBS [20]		
	10	20	30	10	20	30	10	20	30	10	20	30
s5378	43	39	37	57	0	0	52	0	0	51	46	39
c5315	54	43	34	40	0	0	52	0	0	52	0	37
c7552	51	31	27	0	0	0	0	31	0	51	25	32
s9234	53	52	49	0	0	0	59	0	0	56	46	45
fpuDiv	53	43	41	51	0	0	46	0	0	49	44	0
fpuIn	45	38	0	39	0	0	37	0	0	40	31	0
ifuDcl	39	37	0	29	0	0	31	34	0	35	0	0
ifuIfq	47	50	41	56	0	0	55	0	0	46	43	39
lsuExp	49	44	43	0	0	40	48	0	0	50	45	29
lsuStb	48	41	36	55	0	35	50	40	0	45	44	33
lsuRw	53	45	0	0	0	0	44	0	0	57	37	0

TABLE XIV. Comparison with related work

Technique	Sequentiality	No. of ICs	Divide & Conquer	Oracle Access
Sensitization [20]	Yes	2+	No	Yes
DeCamo [18]	Yes	2+	No	Yes
This work	No	1	Yes	No

- **Blocked access to test structures**. The sequentiality problem can also be overcome by gaining physical access to the scan chains. An attacker with such access can consider a sequential circuit as chunks of combinational blocks. However, modern ICs do not provide unauthorized scan access to users. Thus, an attacker cannot subvert the sequentiality problem using scan chains. A HackTest attacker does not need to record the output from the IC. He only needs test data and camouflaged netlist. Even if the scan chains are disabled/locked, the IC can be reverse engineered to obtain camouflaged netlist.
  Our attack does not require oracle (input-output) access to the IC (or its scan chains), but rather uses the test data, which has been given to the malicious test facility and generated for the camouflaged design with the scan access.
- **Number of ICs**. Sensitization and DeCamo attacks require at least two camouflaged ICs – one to reverse engineer and obtain the camouflaged netlist, and the other to apply input patterns and observe the responses. Camouflaged ICs are mainly used in defense applications, wherein an attacker cannot always obtain two ICs. For instance, consider a camouflaged IC being used in a drone or in a remote sensor. In most real-life scenarios, an attacker captures only one copy of an equipment [44]. Thus, the sensitization and DeCamo attacks do not work in these scenarios. Our attack requires only one IC and is thus applicable in real-life scenarios.
- **Scalability**. Sensitization and DeCamo attacks consider the design monolithically. When the scan chain is locked, the sole access they have to rely on is through the primary inputs and outputs. These attacks do not scale well for modern designs, which have millions of gates, as the attacks are NP-hard problems [18]. On the contrary, our attack HackTest can employ a modular approach enabled by the test structures. Test structures, such as test wrappers, usually divide a large design into multiple isolated blocks (i.e., cores) to facilitate concurrent testing, thereby reducing test time and test power [45]. Thus, although our attack is an NP-hard problem, it can operate independently on test data fragments, each generated for testing a core. HackTest can thus extract the secret of individual cores independently. In short, HackTest solves multiple but smaller instances of NP-hard problems rather than one big NP-hard problem.

IX. Discussion

Differences between the attack patterns used by HackTest and DeCamo. One may consider that HackTest is successful as the patterns generated by ATPG and the patterns generated by DeCamo attack overlap. However, our experiments show that there is no intersection between the two sets of patterns, each generated with a completely different objective. The objective of patterns used by the DeCamo attack

TABLE XV. FAULT COVERAGE (%) ACHIEVED BY ATPG GENERATED PATTERNS AND DECAMO ATTACK [18] PATTERNS.

Benchmark		s5378	c5315	c7552	s9234	fpuDiv	fpuIn	ifuDcl	ifuIfq	lsuExcp	lsuStb	lsuRw	Average
ATPG		99.9	99.9	99.4	99.9	100	100	100	100	100	100	100	99.9
DeCamo	RS [20]	75.8	80.0	83.9	72.5	75.0	78.8	77.4	69.2	66.9	79.1	72.8	76.4
	OCS [20]	62.6	65.8	61.8	64.2	65.1	73.9	66.6	54.2	57.2	58.4	70.4	64.5
	CBS [20]	63.9	63.5	70.0	53.3	66.8	73.4	63.8	56.1	51.4	61.8	64.3	64.5
	OCS+CBS [20]	75.1	76.4	83.1	72.5	77.4	80.1	77.0	66.1	68.2	79.2	74.8	75.4

TABLE XVI. HACKTEST RESULTS ON CAMOUFLAGED COMPRESSION CIRCUITRY. NUMBER OF ASSIGNMENTS (OUT OF 64) RETRIEVED CORRECTLY BY THE ATTACK WHEN 32 GATES ARE CAMOUFLAGED IN THE DECOMPRESSOR CIRCUIT AND 32 IN THE ORIGINAL CIRCUIT. $CR = 10$.

Benchmark	s5378	c5315	c7552	s9234	fpuDiv	fpuIn	ifuDcl	ifuIfq	lsuExp	lsuStb	lsuRw
RS [20]	59	61	64	63	59	58	62	60	61	62	60
OCS [20]	63	64	64	59	61	63	64	62	64	62	59
CBS [20]	61	61	0	64	58	62	64	64	62	64	58
OCS+CBS [20]	64	64	62	63	63	61	59	60	62	63	64

is to find assignments to the camouflaged gates, whereas, the objective of patterns generated by ATPG is to detect faults at minimal cost. Table XV presents the fault coverage obtained by the ATPG patterns (independent of camouflaged gate-selection techniques), and the fault coverage obtained by DeCamo attack patterns. The average fault coverage for DeCamo attack patterns is only $\sim 70\%$ as compared to $\sim 100\%$ for the ATPG patterns. Another important difference is that in DeCamo attack, the attacker generates the attack patterns; whereas, in HackTest the defender generates the test patterns using ATPG tools and hands them over to the attacker.

Extending HackTest to other fault models. Multiple fault models such as stuck-at fault, bridging fault, transition fault and delay fault model are used in IC testing. Since HackTest internally uses an ATPG solver, it can accommodate all fault models that are supported by existing ATPG tools [37]. As an example, in case of transition faults, two patterns need to be applied in consecutive clock cycles to detect a single transition fault. The first pattern initializes the fault location to a value ϕ; $\phi = 0$ to detect a slow-to-rise fault, and $\phi = 1$ to detect a slow-to-fall fault). The second pattern serves to detect a stuck-at-ϕ fault in the circuit [46]. To accommodate the transition fault model, each HackTest constraint will consist of a pair of test stimuli, and single test response value.

Hiding test responses to prevent HackTest. In an attempt to thwart misuse of test responses, a designer can choose to send only the test stimuli (and not the test responses) to the test facility. The test facility will have to return the test responses of each chip to the designer who later decides whether the chip is functional or faulty. In secret oblivious ATPG, the x's in the test responses may be left as such, so the attacker cannot get information about the correct responses. However, none of these methods thwarts HackTest. The attacker in the test facility has access to multiple chips, most of which are fault free (assuming yield is high). The attacker can record the responses of multiple ICs and find the correct responses through majority voting. The fact that the ICs are in the test facility for a limited period does not stop the attacker from recording the responses of a few ICs; the task is not that time-consuming. Moreover, each IC has to be tested, and the ATE records the responses of each IC. Note that the IC may be not be reverse engineered during the limited window of opportunity available to the attacker and DeCamo/sensitization attack may not be launched. When the functional IC is available in the market, the scan infrastructure will be locked, thus preventing the retrieval of responses from the scan cells.

Camouflaging compression circuitry. Another countermeasure against HackTest is camouflaging the compression circuitry. Table XVI presents the HackTest success rate for the scenario when 32 gates are camouflaged in the compression circuitry, and 32 in the original circuit. Our preliminary results indicate that camouflaging the compression circuity does not prevent HackTest. It is in fact detrimental for security. HackTest becomes more effective as shown by a high percentage ($\sim 90\%$) of assignments retrieved correctly for most of the circuits, as compared to HackTest-v1 (see Table X), where all the gates to be camouflaged are selected from the original circuit.

X. RELATED WORK

Other misuse of test infrastructure. In the past, researchers have also examined how test infrastructure itself can be misused to jeopardize the security of ICs [47], but from an entirely different perspective than ours. For example, it has been demonstrated that the secret keys for cryptographic algorithms, such as DES and AES, can be leaked through the use of scan chains [48]. Several scan based attacks and countermeasures have been developed in the last decade with an emphasis on the security of hardware implementations of cryptographic algorithms [49], [47].

While most attacks assume standard, publicly known test infrastructure such as scan chains, extraction of secrets using undocumented proprietary test infrastructure has also been demonstrated [39]. The attacker, at first, extracts the details of proprietary test infrastructure using reverse-engineering techniques and later retrieves the secret information stored on the IC by exploiting the knowledge of the test infrastructure.

The existing research efforts focus only on the security vulnerabilities of the test infrastructure but do not take into account the vulnerabilities associated with the test data, i.e., the test patterns and the responses, and the ATPG algorithms. More importantly, they require input-output access to the chip. HackTest exploits test data instead.

Other IP protection techniques. IC camouflaging is one IP protection technique that prevents reverse engineering by a malicious user. Split manufacturing is another IP protection technique that prevents reverse engineering by a malicious foundry, but not by a malicious user [50]. Logic encryption thwarts piracy, reverse engineering, and overbuilding at malicious foundry [51], by encrypting the design with a key. Logic encryption can be broken if the test patterns are generated with the knowledge of the correct key during ATPG [52], in a security-oblivious manner. HackTest is the first attack that exploits the test data to break the IC camouflaging – that utilizes a trusted foundry model – in realistic settings, i.e., with scan compression and different tools used during ATPG and by the attack.

XI. Conclusion

IP protection techniques are becoming essential to thwart various attacks, such as reverse engineering and piracy. To deliver the protection they promise, these techniques must be resilient against attacks.

In this work, we consider IC camouflaging as a case study. IC camouflaging has been broken via recent attacks, although these attacks make very strong assumptions, and thus may be impractical for realistic ICs [20], [18]. The fundamental assumption for the existing attacks to work is access to scan chains, which is typically protected. Without this access, the existing attacks are not applicable to sequential designs due to complexity and scalability problems.

We propose an attack, HackTest, that does not require oracle (input-output) access to scan chains. The proposed attack rather operates on test data, raw or uncompressed, to extract the design secret. Because the test data is generated by ATPG tools, under the assumption that deep access to the IC is delivered by on-chip test structures, an analysis of the test data effectively provides the same access in a "soft" manner, i.e., without oracle access.

Over the last few decades, a significant amount of effort has been invested into ATPG algorithms in order to maximize test quality. So far, the focus has been on the generation of test data that "reveals" (the faults) rather than "protects" (the secret). This is the fundamental reason why HackTest is so effective in extracting design secrets from the test data. Our experimental results confirm the success of HackTest in realistic scenarios ranging from basic scan to compressed scan, and from regular ATPG to secret-oblivious ATPG. We validate the underlying reason of HackTest success by observing the close correlation between test quality and HackTest success. We demonstrate that without truly incorporating security into ATPG, all the designer can do is trade test quality for protection against HackTest. As compromise in test quality is neither desirable nor acceptable, further research in developing secret-protecting ATPG algorithms is a must.

References

[1] M. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-signal VLSI circuits*, vol. 17. Springer Science & Business Media, 2000.

[2] T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 2–15, 1982.

[3] T. Kirkland and M. R. Mercer, "Algorithms for Automatic Test-Pattern Generation," *IEEE Design and Test of Computers*, no. 3, pp. 43–55, 1988.

[4] SypherMedia, "SypherMedia Library Circuit Camouflage Technology." http://www.smi.tv/solutions.htm. [Aug 10, 2015].

[5] J. P. Baukus, L. W. Chow, R. P. Cocchi, P. Ouyang, and B. J. Wang, "Camouflaging a standard cell based integrated circuit," *US Patent no. 8151235*, 2012.

[6] Chipworks, "Texas Instruments 4377401 Baseband Processor TSMC 65nm Process Transistor Characterization." http://www.chipworks.com/TOC/TI_4377401_TSMC_Bb_Processor_TCR-0703-801_TOC.pdf.

[7] Chipworks, "Intel's 22-nm Tri-gate Transistors Exposed." http://www.chipworks.com/blog/technologyblog/2012/04/23/intels-22-nm-tri-gate-transistors-exposed/, 2012.

[8] R. S. Wahby, M. Howald, S. Garg, abhi shelat, and M. Walfish, "Verifiable ASICs." Cryptology ePrint Archive, Report 2015/1243, 2015. http://eprint.iacr.org/.

[9] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building Stealthy and Malicious Hardware," in *IEEE Symposium on Security and Privacy*, pp. 64–77, 2011.

[10] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[11] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *USENIX Security*, pp. 291–306, 2007.

[12] M. Berry and G. John, "Outsourcing Test – What are the most valuable engagement periods?." http://www.amkor.com/go/outsourcing-test, 2014. [May 16, 2016].

[13] A. P. Room, "Silicon Laboratories and ASE announce milestone shipment of 10 million tested integrated circuits." http://www.aseglobal.com/en/News/PressRoomDetail.aspx?ID=45, 2014. [May 16, 2016].

[14] "Test Overview." http://www.spil.com.tw/services/?u=2*0, 2005. [May 12, 2016].

[15] "Test Services Overview." http://www.statschippac.com/en/services/testservices.aspx, 2005. [May 10, 2016].

[16] Degate. http://www.degate.org/documentation/.

[17] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2016. to be published.

[18] M. E. Massad, S. Garg, and M. V. Tripunitara, "Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes," in *Network and Distributed System Security Symposium*, 2015.

[19] D. Liu, C. Yu, X. Zhang, and D. Holcomb, "Oracle-guided Incremental SAT Solving to Reverse Engineer Camouflaged Logic Circuits," in *Design, Automation Test in Europe*, pp. 433–438, 2016.

[20] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," in *ACM SIGSAC Conference on Computer and Communications Security*, pp. 709–720, 2013.

[21] S. Eggersglüß and R. Drechsler, *High Quality Test Pattern Generation and Boolean Satisfiability*, ch. ATPG Based on Boolean Satisfiability, pp. 59–70. Boston, MA: Springer US, 2012.

[22] N. Touba *et al.*, "Survey of Test Vector Compression Techniques," *IEEE Design and Test of Computers*, vol. 23, no. 4, pp. 294–303, 2006.

[23] K. Chakrabarty, "Zero-Aliasing Space ompaction Using Linear Compactors with Bounded Overhead," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 5, pp. 452–457, 1998.

[24] I. Bayraktaroglu and A. Orailoglu, "Test Volume and Application Time Reduction through Scan Chain Concealment," in *IEEE/ACM Design Automation Conference*, pp. 151–155, 2001.

[25] O. Sinanoglu, "Align-Encode: Improving the Encoding Capability of Test Stimulus Decompressors," in *IEEE International Test Conference*, pp. 1–10, 2008.

[26] SEMI, "Innovation is at Risk Losses of up to $4 Billion Annually due to IP Infringement," 2008. [June 10, 2015].

[27] R. Torrance and D. James, "The State-of-the-Art in Semiconductor Reverse Engineering," in *IEEE/ACM Design Automation Conference*, pp. 333–338, 2011.

[28] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[29] Chipworks, "Reverse engineering software." http://www.chipworks.com/en/technical-competitive-analysis/resources/reerse-engineering-software. [Jan 10, 2016].

[30] M. Shiozaki, R. Hori, and T. Fujino, "Diffusion Programmable Device: The Device to Prevent Reverse Engineering," *IACR Cryptology ePrint Archive*, p. 109, 2014.

[31] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy Dopant-Level Hardware Trojans," in *Cryptographic Hardware and Embedded Systems*, pp. 197–214, 2013.

[32] B. Krishnamurthy and S. B. Akers, "On the complexity of estimating the size of a test set," *IEEE Transactions on Computers*, no. 8, pp. 750–753, 1984.

[33] M. R. Prasad, P. Chong, and K. Keutzer, "Why is ATPG Easy?," in *ACM/IEEE Design Automation Conference*, pp. 22–28, ACM, 1999.

[34] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design and Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[35] "Sun Microsystems, OpenSPARC T1 Processor," *http://www.opensparc.net/opensparc-t1/index.html.*

[36] I. Polian, C. Miller, P. Engelke, T. Nopper, A. Czutro, and B. Becker, "Benchmarking Academic DFT Tools on the OpenSparc Microprocessor," in *IEEE International Test Conference*, pp. 1–1, 2008.

[37] S. U. Manual, "TetraMAX ATPG User Guide," *Version X-2005.09*, pp. 249–264, 2005.

[38] H. Lee and D. Ha, "Atalanta: An efficient atpg for combinational circuits," *Deptartment of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA, Technical Report*, 1993.

[39] M. Kammerstetter, M. Muellner, D. Burian, C. Platzer, and W. Kastner, "Breaking Integrated Circuit Device Security through Test Mode Silicon Reverse Engineering," in *ACM SIGSAC Conference on Computer and Communications Security*, pp. 549–557, 2014.

[40] I. Pomeranz, S. Kundu, and S. M. Reddy, "On Output Response Compression in the Presence of Unknown Output Values," in *IEEE/ACM Design Automation Conference*, pp. 255–258, 2002.

[41] A. L. Oliveira, "Robust Techniques for Watermarking Sequential Circuit Designs," in *IEEE/ACM Design Automation Conference*, pp. 837–842, 1999.

[42] D. Kroening, M. Lewis, and G. Weissenbacher, "Proving Safety with Trace Automata and Bounded Model Checking," in *International Symposium on Formal Methods*, pp. 325–341, Springer, 2015.

[43] E. Dubrova, M. Teslenko, and L. Ming, "Finding Attractors in Synchronous Multiple-valued Networks using SAT-based Bounded Model Checking," in *IEEE International Symposium on Multiple-Valued Logic*, pp. 144–149, 2010.

[44] T. Guardian, "Iran claims to have reverse-engineered US spy drone." http://www.theguardian.com/world/2012/apr/22/iran-reverse-engineer-spy-drone, 2014. [Jan. 10, 2016].

[45] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing Embedded-Core-Based System Chips," *Computer*, vol. 32, no. 6, pp. 52–60, 1999.

[46] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits," *Kluwer Academic Publishers, Boston*, 2000.

[47] J. Da Rolt, A. Das, G. Di Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test Versus Security: Past and Present," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 1, pp. 50–62, 2014.

[48] B. Yang, K. Wu, and R. Karri, "Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard," in *IEEE International Test Conference*, pp. 339–344, 2004.

[49] B. Yang, K. Wu, and R. Karri, "Secure Scan: A Design-for-Test Architecture for Crypto Chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2287–2293, 2006.

[50] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," *USENIX Conference on Security*, pp. 495–510, 2013.

[51] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[52] M. Yasin, S. M. Saeed, J. Rajendran, and O. Sinanoglu, "Activation of Logic Encrypted Chips: Pre-test or Post-test?," in *Design, Automation Test in Europe*, pp. 139–144, 2016.

XII. Appendix 1 — Complexity of HackTest

Lemma 1. A clause ψ_i in a CNF formula ϕ can be represented by an OR gate G_i with output o_i. ψ_i is satisfied iff $o_i = 1$.

Proof: The clause ψ_i is a disjunction of m_i literals, l_{ij}. A clause is satisfied if at least one of the m_i literals evaluates to 1 for a given assignment. Let each literal is represented by a wire that drives an m_i-input OR gate G_i whose output is o_i. Then, $o_i = 1$ iff at least one of the inputs wire takes a value of 1.

Lemma 2. A *s-a-*0 fault on the output o_i of an OR gate G_i can be detected iff $o_i = 1$.

Proof: According the principle of fault activation in VLSI testing, the fault location must be set of a value opposite to the fault. Thus, to detect a *s-a-*0 fault on o_i, $o_i = 1$.

Lemma 3. A clause ψ_i in a CNF formula ϕ is satisfied iff a *s-a-*0 fault on the output o_i of the OR gate G_i is detected.

Proof: According to Lemma 2, a *s-a-*0 fault on o_i is detected iff $o_i = 1$. According to Lemma 1, ψ_i can be represented by an OR gate G_i with output o_i, and $o_i = 1$ implies that ψ_i is satisfied.

Definition 1. *HackTest instance* A HackTest instance is a 3-tuple comprising $(circuit, fault\text{–}set, test\text{–}set)$. The $circuit$ has n camouflaged gates whose functionality is unknown. HackTest is the problem of finding an assignment A that maximizes the number of faults detected in the $fault\text{–}set$ for a given $test\text{–}set$.

Definition 2. *MaxSAT* Consider a CNF formula ϕ containing N clauses over n Boolean variables $x_1, x_2, \ldots, x_n$: $\phi = \bigwedge_{i=1}^{N} \psi_i$. The i^{th} clause ψ_i is represented as: $\bigvee_{j=1}^{m_i} l_{ij}$, where the literal l_{ij} is a variable x_i or its complement $\neg x_i$.

MaxSAT over the CNF formula ϕ is the problem of finding an assignment v that maximizes the number of clauses satisfied in ϕ:

$$\max \left\{ \sum_{i=1}^{N} \|\psi_i\|_v \;\middle|\; v \in \{0,1\}^n \right\} \tag{6}$$

where, $\|\psi_i\|_v = \begin{cases} 1, & \text{if } v \models \psi_i \\ 0, & \text{otherwise} \end{cases}$

Theorem 1: The complexity of HackTest is NP-hard.

Proof: We show that HackTest is NP-hard by reducing the MaxSAT problem to HackTest in linear time. The complexity of MaxSAT is NP-hard. We show mapping from a MaxSAT instance to individual components of a HackTest instance.

$Circuit$. The MaxSAT problem ϕ can be mapped to a Boolean circuit C_ϕ in the following way. According to Lemma 1, each clause ψ_i in ϕ can be represented as an m_i-input OR gate G_i with output o_i, as shown in Figure 13. o_i is also the primary output of the circuit C_ϕ. There are N OR gates each corresponding to a clause.

Each variable in ϕ is represented as a wire in C_ϕ. If a variable appears in its true form in a clause, it is connected directly to an input of the OR gate representing that clause, as shown in Figure 13. If a variable appears in its complement form in a clause, it is connected to an inverter whose output is connected to an input of the OR gate representing that clause.

For each variable x_i in ϕ, a 2-input camouflaged gate is added. The functionality of the camouflaged gate can be NAND or NOR. The inputs of the i^{th} camouflaged gate is z_{2i-1} and z_{2i}. Its output is x_i, which represents a variable in the MaxSAT problem ϕ, as shown in Figure 13. Each z_i is a primary input of the circuit C_ϕ.

$Fault\text{-}set$. The fault set consists of s-a-0 faults on the outputs of C_ϕ. Taking only s-a-0 faults into account, the FC (defined in Equation 2) can be re-defined as $FC = \frac{1}{N}\sum_{i=1}^{N} \mathrm{fd_i}$.

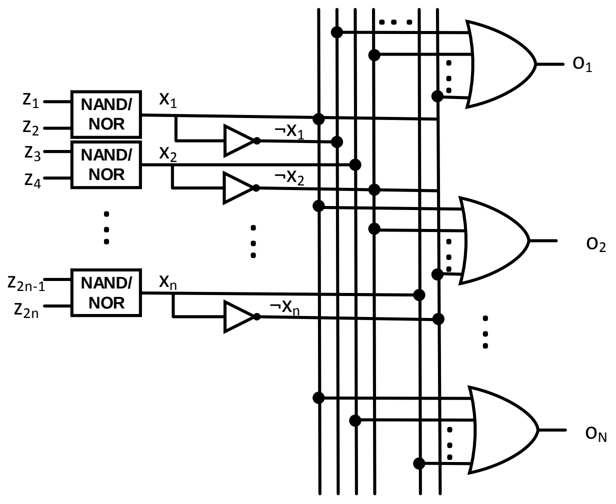


Fig. 13. Proof of Theorem 1. C_ϕ circuit equivalent of CNF formula ϕ that reduces the MaxSAT problem to HackTest. C_ϕ has $2n$ inputs, N outputs, and n camouflaged gates. Each OR gate represents a clause in ϕ.

$Test\text{-}set$. For each NAND/NOR camouflaged gate, we can apply either 10 or 01 as an input since the outputs of NAND and NOR gates differ only for these input combinations; the output is 1 for a NAND gate, and 0 for a NOR gate. Thus, the value of two inputs to a camouflaged gate must be different, i.e., for the i^{th} camouflaged gate, $z_{2i-1} \oplus z_{2i} = 1$.

Let A be an assignment to the functionality of all camouflaged gates in C_ϕ. HackTest on C_ϕ is defined as:

$$\begin{aligned} &\text{maximize} && FC \\ &\text{subject to} && \underset{1\le i\le n}{\forall} z_{2i-1} \oplus z_{2i} = 1 \\ &\text{solve for} && A \end{aligned} \tag{7}$$

The solution to HackTest is an assignment A that leads to maximum fault coverage for C_ϕ. Solving for A also solves for the assignment v in Equation 6, since according to Lemma 3, a s-a-0 fault on o_i is detected iff the clause ψ_i is satisfied. Thus, fault coverage of C_ϕ is maximized iff the number of clauses satisfied in ϕ is maximized. If the assignment of a camouflaged gate is NAND in C_ϕ, the corresponding variable in ϕ is assigned 1; if it is NOR, the corresponding variable in ϕ is assigned 0. Thus, we can solve a MaxSAT problem by reducing it to HackTest.

The reduction can be performed in linear time. Replacing N clauses in ϕ with N OR gates takes $O(N)$ time; inserting n NOT gates and n camouflaged gates for each literal takes $O(n)$ time. Since $n \leq N$, the computational complexity of reduction is $O(N)$.

Complexity of MaxSAT is NP-hard. As MaxSAT $\leq_P$ HackTest, the complexity of HackTest is also NP-hard.